



Red Hat Directory Server 10

Plug-in Guide

Updated for Directory Server 10.3

Red Hat Directory Server 10 Plug-in Guide

Updated for Directory Server 10.3

Marc Muehlfeld
Red Hat Customer Content Services
mmuehlfeld@redhat.com

Petr Bokoč
Red Hat Customer Content Services

Tomáš Čapek
Red Hat Customer Content Services

Ella Deon Ballard
Red Hat Customer Content Services

Legal Notice

Copyright © 2018 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide provides a reference of plug-ins, parameter blocks, and data structure available in the Directory Server API.

Table of Contents

PREFACE	15
1. REQUIRED CONCEPTS	15
PART I. INTRODUCTION TO DIRECTORY SERVER PLUG-INS	16
CHAPTER 1. AN OVERVIEW OF DIRECTORY SERVER PLUG-INS	17
1.1. ABOUT DIRECTORY SERVER PLUG-INS	17
1.2. HOW DIRECTORY SERVER PLUG-INS WORK	17
1.3. TYPES OF DIRECTORY SERVER PLUG-INS	19
1.4. USING DIRECTORY SERVER PLUG-IN APIS	21
1.5. DEPRECATED FUNCTIONS AND REPLACEMENTS	22
CHAPTER 2. WRITING AND COMPILING PLUG-INS	25
2.1. WRITING A PLUG-IN FUNCTION	25
2.2. WRITING PLUG-IN INITIALIZATION FUNCTIONS	29
2.3. COMPILING A DIRECTORY SERVER PLUG-IN	32
CHAPTER 3. CONFIGURING PLUG-INS	34
3.1. CREATING A PLUG-IN CONFIGURATION FILE	34
3.2. PLUG-IN LOGGING FEATURES	40
3.3. LOADING THE PLUG-IN CONFIGURATION FILE	40
3.4. PASSING EXTRA ARGUMENTS TO PLUG-INS	40
3.5. SETTING THE LOG LEVEL OF THE SERVER	41
CHAPTER 4. AN EXAMPLE PLUG-IN	43
4.1. WRITING THE PLUG-IN EXAMPLE	43
4.2. COMPILING THE PLUG-IN EXAMPLE	48
4.3. REGISTERING THE PLUG-IN	48
4.4. RUNNING THE PLUG-IN	49
PART II. WRITING FUNCTIONS AND PLUG-INS	50
CHAPTER 5. FRONTEND API FUNCTIONS	51
5.1. LOGGING MESSAGES	51
5.2. ADDING NOTES TO ACCESS LOG ENTRIES	51
5.3. SENDING DATA TO THE CLIENT	52
5.4. DETERMINING IF AN OPERATION WAS ABANDONED	52
5.5. WORKING WITH ENTRIES, ATTRIBUTES, AND VALUES	52
5.6. WORKING WITH DNS AND RDNS	58
5.7. WORKING WITH SEARCH FILTERS	59
5.8. CHECKING PASSWORDS	62
CHAPTER 6. WRITING PRE- AND POST-OPERATION PLUG-INS	64
6.1. HOW PRE- AND POST-OPERATION PLUG-INS WORK	64
6.2. TYPES OF PRE-OPERATION AND POST-OPERATION FUNCTIONS	65
6.3. USING PLUG-IN CONFIGURATION INFORMATION IN PREOP PLUG-INS	69
6.4. REGISTERING PRE- AND POST-OPERATION FUNCTIONS	69
CHAPTER 7. DEFINING FUNCTIONS FOR LDAP OPERATIONS	70
7.1. SPECIFYING START AND CLOSE FUNCTIONS	70
7.2. PROCESSING AN LDAP BIND OPERATION	70
7.3. PROCESSING AN LDAP UNBIND OPERATION	71
7.4. PROCESSING AN LDAP SEARCH OPERATION	72
7.5. PROCESSING AN LDAP COMPARE OPERATION	75

7.6. PROCESSING AN LDAP ADD OPERATION	76
7.7. PROCESSING AN LDAP MODIFY OPERATION	77
7.8. PROCESSING AN LDAP MODIFY RDN OPERATION	78
7.9. PROCESSING AN LDAP DELETE OPERATION	80
7.10. PROCESSING AN LDAP ABANDON OPERATION	80
CHAPTER 8. DEFINING FUNCTIONS FOR AUTHENTICATION	81
8.1. UNDERSTANDING AUTHENTICATION METHODS	81
8.2. HOW THE DIRECTORY SERVER IDENTIFIES CLIENTS	81
8.3. HOW THE AUTHENTICATION PROCESS WORKS	81
8.4. WRITING YOUR OWN AUTHENTICATION PLUG-IN	83
8.5. WRITING A PRE-OPERATION BIND PLUG-IN	83
CHAPTER 9. WRITING ENTRY STORE/FETCH PLUG-INS	95
9.1. HOW ENTRY STORE/FETCH PLUG-INS WORK	95
9.2. WRITING ENTRY STORE/FETCH FUNCTIONS	95
9.3. REGISTERING ENTRY STORE/FETCH FUNCTIONS	96
CHAPTER 10. WRITING EXTENDED OPERATION PLUG-INS	98
10.1. HOW EXTENDED OPERATION PLUG-INS WORK	98
10.2. WRITING EXTENDED OPERATION FUNCTIONS	98
10.3. REGISTERING EXTENDED OPERATION FUNCTIONS	99
10.4. SPECIFYING START AND CLOSE FUNCTIONS	102
CHAPTER 11. WRITING MATCHING RULE PLUG-INS	103
11.1. UNDERSTANDING MATCHING RULES	103
11.2. UNDERSTANDING MATCHING RULE PLUG-INS	104
11.3. INDEXING BASED ON MATCHING RULES	107
11.4. HANDLING EXTENSIBLE MATCH FILTERS	112
11.5. HANDLING SORTING BY MATCHING RULES	118
11.6. WRITING A DESTRUCTOR FUNCTION	119
11.7. WRITING AN INITIALIZATION FUNCTION	119
11.8. REGISTERING MATCHING RULE FUNCTIONS	120
11.9. SPECIFYING START AND CLOSE FUNCTIONS	121
CHAPTER 12. USING THE CUSTOM DISTRIBUTION LOGIC	122
12.1. ABOUT DISTRIBUTING FLAT NAMESPACES	122
12.2. CREATING A DISTRIBUTION FUNCTION	123
12.3. ADDING THE DISTRIBUTION FUNCTION TO YOUR DIRECTORY	124
12.4. USING THE DISTRIBUTION LOGIC EXAMPLES	127
12.5. CUSTOM DISTRIBUTION CHECKLIST	127
CHAPTER 13. USING DATA INTEROPERABILITY PLUG-INS	129
13.1. INSTALLING DIRECTORY SERVER	129
13.2. ENABLING THE DIOP FEATURE IN DIRECTORY SERVER	134
13.3. USING THE DIOP FEATURE	135
13.4. SAMPLE DIOP PLUG-IN	136
13.5. PLUG-IN API REFERENCE	139
PART III. DATA TYPE AND STRUCTURE REFERENCE	140
CHAPTER 14. DATA TYPE AND STRUCTURE REFERENCE	141
14.1. Berval	141
14.2. COMPUTED_ATTR_CONTEXT	141
14.3. LDAPCONTROL	142

14.4. LDAPMOD	143
14.5. MRFILTERMATCHFN	146
14.6. PLUGIN_REFERRAL_ENTRY_CALLBACK	147
14.7. PLUGIN_RESULT_CALLBACK	148
14.8. PLUGIN_SEARCH_ENTRY_CALLBACK	149
14.9. SEND_LDAP_REFERRAL_FN_PTR_T	150
14.10. SEND_LDAP_RESULT_FN_PTR_T	151
14.11. SEND_LDAP_SEARCH_ENTRY_FN_PTR_T	152
14.12. SLAPI_ATTR	153
14.13. SLAPI_BACKEND	154
14.14. SLAPI_BACKEND_STATE_CHANGE_FNPTR	156
14.15. SLAPI_COMPONENTID	157
14.16. SLAPI_COMPUTE_CALLBACK_T	157
14.17. SLAPI_COMPUTE_OUTPUT_T	158
14.18. SLAPI_CONNECTION	159
14.19. SLAPI_CONDVAR	160
14.20. SLAPI_COUNTER	160
14.21. SLAPI_DN	161
14.22. SLAPI_ENTRY	163
14.23. SLAPI_FILTER	166
14.24. SLAPI_MATCHINGRULEENTRY	167
14.25. SLAPI_MOD	169
14.26. SLAPI_MODS	170
14.27. SLAPI_MUTEX	172
14.28. SLAPI_OPERATION	173
14.29. SLAPI_PBLOCK	173
14.30. SLAPI_PLUGINDESC	175
14.31. SLAPI_RDN	176
14.32. SLAPI_TASK	178
14.33. SLAPI_UNIQUEID	181
14.34. SLAPI_VALUE	181
14.35. SLAPI_VALUESET	183
14.36. REPLICATION SESSION HOOKS CALLBACKS	184
14.37. SYNCHRONIZATION CALLBACKS AND DATA TYPES	189
PART IV. FUNCTION REFERENCE	199
CHAPTER 15. DISTRIBUTION ROUTINES	200
15.1. DISTRIBUTION_PLUGIN_ENTRY_POINT()	200
CHAPTER 16. FUNCTIONS FOR ACCESS CONTROL	202
16.1. SLAPI_ACCESS_ALLOWED()	202
16.2. SLAPI_ACL_CHECK_MODS()	204
16.3. SLAPI_ACL_VERIFY_ACI_SYNTAX()	206
CHAPTER 17. FUNCTIONS FOR INTERNAL OPERATIONS AND PLUG-IN CALLBACK	208
17.1. SLAPI_ADD_INTERNAL_PB()	208
17.2. SLAPI_DELETE_INTERNAL_PB()	209
17.3. SLAPI_FREE_SEARCH_RESULTS_INTERNAL()	210
17.4. SLAPI_MODIFY_INTERNAL_PB()	210
17.5. SLAPI_MODRDN_INTERNAL_PB()	211
17.6. SLAPI_SEARCH_INTERNAL_CALLBACK_PB()	211
17.7. SLAPI_SEARCH_INTERNAL_GET_ENTRY()	212
17.8. SLAPI_SEARCH_INTERNAL_PB()	213

CHAPTER 18. FUNCTIONS FOR SETTING INTERNAL OPERATION FLAGS	215
18.1. SLAPI_ADD_ENTRY_INTERNAL_SET_PB()	215
18.2. SLAPI_ADD_INTERNAL_SET_PB()	216
18.3. SLAPI_DELETE_INTERNAL_SET_PB()	217
18.4. SLAPI_MODIFY_INTERNAL_SET_PB()	218
18.5. SLAPI_RENAME_INTERNAL_SET_PB()	220
18.6. SLAPI_SEARCH_INTERNAL_SET_PB()	221
18.7. SLAPI_SEQ_INTERNAL_CALLBACK_PB()	223
18.8. SLAPI_SEQ_INTERNAL_SET_PB()	224
CHAPTER 19. FUNCTIONS FOR HANDLING ATTRIBUTES	226
19.1. SLAPI_ATTR_ADD_VALUE()	227
19.2. SLAPI_ATTR_BASETYPE()	228
19.3. SLAPI_ATTR_DUP()	228
19.4. SLAPI_ATTR_FIRST_VALUE()	229
19.5. SLAPI_ATTR_FLAG_IS_SET()	230
19.6. SLAPI_ATTR_FREE()	230
19.7. SLAPI_ATTR_GET_BERVALS_COPY()	231
19.8. SLAPI_ATTR_GET_FLAGS()	231
19.9. SLAPI_ATTR_GET_NUMVALUES()	232
19.10. SLAPI_ATTR_GET_OID_COPY()	233
19.11. SLAPI_ATTR_GET_TYPE()	233
19.12. SLAPI_ATTR_GET_VALUESET()	234
19.13. SLAPI_ATTR_INIT()	235
19.14. SLAPI_ATTR_NEW()	236
19.15. SLAPI_ATTR_NEXT_VALUE()	236
19.16. SLAPI_ATTR_SET_VALUESET()	237
19.17. SLAPI_ATTR_SYNTAX_NORMALIZE()	237
19.18. SLAPI_ATTR_TYPE2PLUGIN()	238
19.19. SLAPI_ATTR_TYPE_CMP()	238
19.20. SLAPI_ATTR_TYPES_EQUIVALENT()	239
19.21. SLAPI_ATTR_VALUE_CMP()	240
19.22. SLAPI_ATTR_VALUE_FIND()	241
19.23. SLAPI_VALUESSET_SET_FROM_SMOD()	242
CHAPTER 20. FUNCTIONS FOR MANAGING BACKEND OPERATIONS	243
20.1. SLAPI_BE_ADDSUFFIX()	244
20.2. SLAPI_BE_DELETE_ONEXIT()	245
20.3. SLAPI_BE_EXIST()	245
20.4. SLAPI_BE_FREE()	246
20.5. SLAPI_BE_GET_INSTANCE_INFO()	246
20.6. SLAPI_BE_GET_NAME()	246
20.7. SLAPI_BE_GET_READONLY()	247
20.8. SLAPI_BE_GETENTRYPOINT()	247
20.9. SLAPI_BE_GETSUFFIX()	248
20.10. SLAPI_BE_GETTYPE()	249
20.11. SLAPI_BE_IS_FLAG_SET()	249
20.12. SLAPI_BE_ISSUFFIX()	250
20.13. SLAPI_BE_LOGCHANGES()	250
20.14. SLAPI_BE_NEW()	251
20.15. SLAPI_BE_PRIVATE()	251
20.16. SLAPI_BE_SELECT()	252
20.17. SLAPI_BE_SELECT_BY_INSTANCE_NAME()	252

20.18. SLAPI_BE_SET_FLAG()	253
20.19. SLAPI_BE_SET_INSTANCE_INFO()	253
20.20. SLAPI_BE_SET_READONLY()	254
20.21. SLAPI_BE_SETENTRYPOINT()	254
20.22. SLAPI_GET_FIRST_BACKEND()	255
20.23. SLAPI_GET_FIRST_SUFFIX()	255
20.24. SLAPI_GET_NEXT_BACKEND()	256
20.25. SLAPI_GET_NEXT_SUFFIX()	257
20.26. SLAPI_IS_ROOT_SUFFIX()	258
20.27. SLAPI_REGISTER_BACKEND_STATE_CHANGE()	258
20.28. SLAPI_UNREGISTER_BACKEND_STATE_CHANGE()	259
CHAPTER 21. FUNCTIONS FOR DEALING WITH CONTROLS	261
21.1. SLAPI_ADD_CONTROL_EXT()	261
21.2. SLAPI_ADD_CONTROLS()	262
21.3. SLAPI_BUILD_CONTROL()	263
21.4. SLAPI_BUILD_CONTROL_FROM_BERVAL()	264
21.5. SLAPI_CONTROL_PRESENT()	265
21.6. SLAPI_DUP_CONTROL()	266
21.7. SLAPI_GET_SUPPORTED_CONTROLS_COPY()	266
21.8. SLAPI_REGISTER_SUPPORTED_CONTROL()	267
CHAPTER 22. FUNCTIONS FOR SYNTAX PLUG-INS	270
22.1. SLAPI_CALL_SYNTAX_ASSERTION2KEYS_AVA_SV()	270
22.2. SLAPI_CALL_SYNTAX_ASSERTION2KEYS_SUB_SV()	271
22.3. SLAPI_CALL_SYNTAX_VALUES2KEYS_SV()	272
CHAPTER 23. FUNCTIONS FOR MANAGING MEMORY	274
23.1. SLAPI_CH_ARRAY_ADD()	274
23.2. SLAPI_CH_ARRAY_FREE()	275
23.3. SLAPI_CH_BVDUP()	275
23.4. SLAPI_CH_BVECDUP()	276
23.5. SLAPI_CH_CALLOC()	277
23.6. SLAPI_CH_FREE()	277
23.7. SLAPI_CH_FREE_STRING()	278
23.8. SLAPI_CH_MALLOC()	278
23.9. SLAPI_CH_REALLOC()	279
23.10. SLAPI_CH_SMPRINTF()	280
23.11. SLAPI_CH_STRDUP()	281
CHAPTER 24. FUNCTIONS FOR MANAGING ENTRIES	282
24.1. SLAPI_ENTRY2STR()	284
24.2. SLAPI_ENTRY2STR_WITH_OPTIONS()	285
24.3. SLAPI_ENTRY_ADD_RDN_VALUES()	287
24.4. SLAPI_ENTRY_ADD_STRING()	287
24.5. SLAPI_ENTRY_ADD_VALUE()	288
24.6. SLAPI_ENTRY_ADD_VALUES_SV()	289
24.7. SLAPI_ENTRY_ADD_VALUESSET()	289
24.8. SLAPI_ENTRY_ALLOC()	290
24.9. SLAPI_ENTRY_APPLY_MODS()	291
24.10. SLAPI_ENTRY_ATTR_DELETE()	291
24.11. SLAPI_ENTRY_ATTR_FIND()	292
24.12. SLAPI_ENTRY_ATTR_GET_BOOL()	292
24.13. SLAPI_ENTRY_ATTR_GET_CHARPTR()	293

24.14. SLAPI_ENTRY_ATTR_GET_CHARRAY()	294
24.15. SLAPI_ENTRY_ATTR_GET_INT()	295
24.16. SLAPI_ENTRY_ATTR_GET_LONG()	295
24.17. SLAPI_ENTRY_ATTR_GET_UINT()	296
24.18. SLAPI_ENTRY_ATTR_GET_ULONG()	296
24.19. SLAPI_ENTRY_ATTR_HAS_SYNTAX_VALUE()	297
24.20. SLAPI_ENTRY_ATTR_MERGE_SV()	297
24.21. SLAPI_ENTRY_ATTR_REPLACE_SV()	298
24.22. SLAPI_ENTRY_ATTR_SET_CHARPTR()	299
24.23. SLAPI_ENTRY_ATTR_SET_INT()	299
24.24. SLAPI_ENTRY_ATTR_SET_LONG()	300
24.25. SLAPI_ENTRY_ATTR_SET_UINT()	300
24.26. SLAPI_ENTRY_ATTR_SET_ULONG()	301
24.27. SLAPI_ENTRY_DELETE_STRING()	301
24.28. SLAPI_ENTRY_DELETE_VALUES_SV()	302
24.29. SLAPI_ENTRY_DUP()	303
24.30. SLAPI_ENTRY_FIRST_ATTR()	303
24.31. SLAPI_ENTRY_FREE()	304
24.32. SLAPI_ENTRY_GET_DN()	304
24.33. SLAPI_ENTRY_GET_DN_CONST()	305
24.34. SLAPI_ENTRY_GET_NDN()	306
24.35. SLAPI_ENTRY_GET_SDN()	306
24.36. SLAPI_ENTRY_GET_SDN_CONST()	307
24.37. SLAPI_ENTRY_GET_UNIQUEID()	307
24.38. SLAPI_ENTRY_HAS_CHILDREN()	308
24.39. SLAPI_ENTRY_INIT()	308
24.40. SLAPI_ENTRY_MERGE_VALUES_SV()	309
24.41. SLAPI_ENTRY_NEXT_ATTR()	310
24.42. SLAPI_ENTRY_RDN_VALUES_PRESENT()	311
24.43. SLAPI_ENTRY_SCHEMA_CHECK()	311
24.44. SLAPI_ENTRY_SET_DN()	312
24.45. SLAPI_ENTRY_SET_SDN()	312
24.46. SLAPI_ENTRY_SET_UNIQUEID()	313
24.47. SLAPI_ENTRY_SIZE()	314
24.48. SLAPI_IS_ROOTDSE()	314
24.49. SLAPI_STR2ENTRY()	315
CHAPTER 25. FUNCTIONS RELATED TO ENTRY FLAGS	317
25.1. SLAPI_ENTRY_CLEAR_FLAG()	317
25.2. SLAPI_ENTRY_FLAG_IS_SET()	317
25.3. SLAPI_ENTRY_SET_FLAG()	318
CHAPTER 26. FUNCTIONS FOR DEALING WITH FILTERS	320
26.1. SLAPI_FILTER_APPLY()	321
26.2. SLAPI_FILTER_COMPARE()	322
26.3. SLAPI_FILTER_DUP()	322
26.4. SLAPI_FILTER_FREE()	323
26.5. SLAPI_FILTER_GET_ATTRIBUTE_TYPE()	323
26.6. SLAPI_FILTER_GET_AVA()	324
26.7. SLAPI_FILTER_GET_CHOICE()	325
26.8. SLAPI_FILTER_GET_SUBFILT()	327
26.9. SLAPI_FILTER_GET_TYPE()	328
26.10. SLAPI_FILTER_JOIN()	329

26.11. SLAPI_FILTER_JOIN_EX()	329
26.12. SLAPI_FILTER_LIST_FIRST()	330
26.13. SLAPI_FILTER_LIST_NEXT()	331
26.14. SLAPI_FILTER_TEST()	332
26.15. SLAPI_FILTER_TEST_EXT()	333
26.16. SLAPI_FILTER_TEST_SIMPLE()	334
26.17. SLAPI_FIND_MATCHING_PAREN()	335
26.18. SLAPI_STR2FILTER()	335
26.19. SLAPI_VATTR_FILTER_TEST()	335
CHAPTER 27. FUNCTIONS SPECIFIC TO EXTENDED OPERATION	337
27.1. SLAPI_GET_SUPPORTED_EXTENDED_OPS_COPY()	337
CHAPTER 28. FUNCTIONS SPECIFIC TO BIND METHODS	338
28.1. SLAPI_ADD_AUTH_RESPONSE_CONTROL()	338
28.2. SLAPI_GET_SUPPORTED_SASLMECHANISMS_COPY()	339
28.3. SLAPI_REGISTER_SUPPORTED_SASLMECHANISM()	339
CHAPTER 29. FUNCTIONS FOR THREAD-SAFE LDAP CONNECTIONS	340
29.1. SLAPI_LDAP_INIT()	340
29.2. SLAPI_LDAP_UNBIND()	342
CHAPTER 30. FUNCTIONS FOR LOGGING	343
30.1. SLAPI_LOG_ERROR()	343
30.2. SLAPI_IS_LOGLEVEL_SET()	345
CHAPTER 31. FUNCTIONS FOR COUNTERS	346
31.1. SLAPI_COUNTER_ADD()	346
31.2. SLAPI_COUNTER_DECREMENT()	347
31.3. SLAPI_COUNTER_DESTROY()	347
31.4. SLAPI_COUNTER_GET_VALUE()	348
31.5. SLAPI_COUNTER_INCREMENT()	348
31.6. SLAPI_COUNTER_INIT()	349
31.7. SLAPI_COUNTER_NEW()	349
31.8. SLAPI_COUNTER_SET_VALUE()	350
31.9. SLAPI_COUNTER_SUBTRACT()	350
CHAPTER 32. FUNCTIONS FOR HANDLING MATCHING RULES	352
32.1. SLAPI_BERVAL_CMP()	352
32.2. SLAPI_MATCHINGRULE_FREE()	353
32.3. SLAPI_MATCHINGRULE_GET()	353
32.4. SLAPI_MATCHINGRULE_IS_ORDERING()	354
32.5. SLAPI_MATCHINGRULE_NEW()	355
32.6. SLAPI_MATCHINGRULE_REGISTER()	355
32.7. SLAPI_MATCHINGRULE_SET()	356
32.8. SLAPI_MATCHINGRULE_UNREGISTER()	357
32.9. SLAPI_MR_FILTER_INDEX()	357
32.10. SLAPI_MR_INDEXER_CREATE()	358
CHAPTER 33. FUNCTIONS FOR LDAPMOD MANIPULATION	360
33.1. SLAPI_ENTRY2MODS()	363
33.2. SLAPI_MOD_ADD_VALUE()	363
33.3. SLAPI_MOD_DONE()	364
33.4. SLAPI_MOD_DUMP()	364
33.5. SLAPI_MOD_FREE()	365

33.6. SLAPI_MOD_GET_FIRST_VALUE()	365
33.7. SLAPI_MOD_GET_LDAPMOD_BYREF()	366
33.8. SLAPI_MOD_GET_LDAPMOD_PASSOUT()	366
33.9. SLAPI_MOD_GET_NEXT_VALUE()	367
33.10. SLAPI_MOD_GET_NUM_VALUES()	367
33.11. SLAPI_MOD_GET_OPERATION()	368
33.12. SLAPI_MOD_GET_TYPE()	368
33.13. SLAPI_MOD_INIT()	368
33.14. SLAPI_MOD_INIT_BYREF()	369
33.15. SLAPI_MOD_INIT_BYVAL()	370
33.16. SLAPI_MOD_INIT_PASSIN()	370
33.17. SLAPI_MOD_INIT_VALUESSET_BYVAL()	371
33.18. SLAPI_MOD_ISVALID()	372
33.19. SLAPI_MOD_NEW()	372
33.20. SLAPI_MOD_REMOVE_VALUE()	373
33.21. SLAPI_MOD_SET_OPERATION()	373
33.22. SLAPI_MOD_SET_TYPE()	373
33.23. SLAPI_MODS2ENTRY()	374
33.24. SLAPI_MODS_ADD()	375
33.25. SLAPI_MODS_ADD_LDAPMOD()	375
33.26. SLAPI_MODS_ADD_MOD_VALUES()	376
33.27. SLAPI_MODS_ADD_SMOD()	377
33.28. SLAPI_MODS_ADD_MODBVPS()	378
33.29. SLAPI_MODS_ADD_STRING()	378
33.30. SLAPI_MODS_DONE()	379
33.31. SLAPI_MODS_DUMP()	380
33.32. SLAPI_MODS_FREE()	380
33.33. SLAPI_MODS_GET_FIRST_MOD()	380
33.34. SLAPI_MODS_GET_FIRST_SMOD()	381
33.35. SLAPI_MODS_GET_LDAPMODS_BYREF()	382
33.36. SLAPI_MODS_GET_LDAPMODS_PASSOUT()	382
33.37. SLAPI_MODS_GET_NEXT_MOD()	383
33.38. SLAPI_MODS_GET_NEXT_SMOD()	383
33.39. SLAPI_MODS_GET_NUM_MODS()	384
33.40. SLAPI_MODS_INIT()	384
33.41. SLAPI_MODS_INIT_BYREF()	385
33.42. SLAPI_MODS_INIT_PASSIN()	385
33.43. SLAPI_MODS_INSERT_AFTER()	386
33.44. SLAPI_MODS_INSERT_AT()	386
33.45. SLAPI_MODS_INSERT_BEFORE()	387
33.46. SLAPI_MODS_INSERT_SMOD_AT()	388
33.47. SLAPI_MODS_INSERT_SMOD_BEFORE()	388
33.48. SLAPI_MODS_ITERATOR_BACKONE()	389
33.49. SLAPI_MODS_NEW()	389
33.50. SLAPI_MODS_REMOVE()	390
CHAPTER 34. FUNCTIONS FOR MONITORING OPERATIONS	391
34.1. SLAPI_OP_ABANDONED()	391
34.2. SLAPI_OP_GET_TYPE()	391
CHAPTER 35. FUNCTIONS FOR MANAGING PARAMETER BLOCK	393
35.1. SLAPI_PBLOCK_DESTROY()	393
35.2. SLAPI_PBLOCK_GET()	394

35.3. SLAPI_PBLOCK_INIT()	396
35.4. SLAPI_PBLOCK_NEW()	396
35.5. SLAPI_PBLOCK_SET()	397
CHAPTER 36. FUNCTIONS FOR HANDLING PASSWORDS	399
36.1. SLAPI_PW_FIND_SV()	399
36.2. SLAPI_IS_ENCODED()	400
36.3. SLAPI_ENCODE()	401
36.4. SLAPI_ADD_PWD_CONTROL()	401
36.5. SLAPI_PWPOLICY_MAKE_RESPONSE_CONTROL()	402
CHAPTER 37. FUNCTIONS FOR MANAGING RDNs	404
37.1. SLAPI_RDN_ADD()	405
37.2. SLAPI_RDN_COMPARE()	406
37.3. SLAPI_RDN_CONTAINS()	406
37.4. SLAPI_RDN_CONTAINS_ATTR()	407
37.5. SLAPI_RDN_DONE()	408
37.6. SLAPI_RDN_FREE()	408
37.7. SLAPI_RDN_GET_FIRST()	409
37.8. SLAPI_RDN_GET_INDEX()	410
37.9. SLAPI_RDN_GET_INDEX_ATTR()	411
37.10. SLAPI_RDN_GET_NEXT()	411
37.11. SLAPI_RDN_GET_NUM_COMPONENTS()	412
37.12. SLAPI_RDN_GET_RDN()	413
37.13. SLAPI_RDN_GET_NRDN()	413
37.14. SLAPI_RDN_INIT()	413
37.15. SLAPI_RDN_INIT_DN()	414
37.16. SLAPI_RDN_INIT_RDN()	414
37.17. SLAPI_RDN_INIT_SDN()	415
37.18. SLAPI_RDN_ISEMPY()	415
37.19. SLAPI_RDN_NEW()	416
37.20. SLAPI_RDN_NEW_DN()	416
37.21. SLAPI_RDN_NEW_RDN()	417
37.22. SLAPI_RDN_NEW_SDN()	418
37.23. SLAPI_RDN_REMOVE()	418
37.24. SLAPI_RDN_REMOVE_ATTR()	419
37.25. SLAPI_RDN_REMOVE_INDEX()	420
37.26. SLAPI_RDN_SET_DN()	420
37.27. SLAPI_RDN_SET_RDN()	421
37.28. SLAPI_RDN_SET_SDN()	421
37.29. SLAPI_RDN2TYPEVAL()	422
CHAPTER 38. FUNCTIONS FOR MANAGING ROLES	423
38.1. SLAPI_ROLE_CHECK()	423
38.2. SLAPI_REGISTER_ROLE_CHECK()	423
CHAPTER 39. FUNCTIONS FOR MANAGING DNS	425
39.1. SLAPI_DN_ISROOT()	426
39.2. SLAPI_DN_NORMALIZE_CASE()	427
39.3. SLAPI_DN_NORMALIZE_TO_END()	427
39.4. SLAPI_MODDN_GET_NEWDN()	428
39.5. SLAPI_SDN_ADD_RDN()	429
39.6. SLAPI_SDN_COMPARE()	429
39.7. SLAPI_SDN_COPY()	430

39.8. SLAPI_SDN_DONE()	431
39.9. SLAPI_SDN_DUP()	431
39.10. SLAPI_SDN_FREE()	431
39.11. SLAPI_SDN_GET_BACKEND_PARENT()	432
39.12. SLAPI_SDN_GET_DN()	433
39.13. SLAPI_SDN_GET_NDN()	433
39.14. SLAPI_SDN_GET_NDN_LEN()	434
39.15. SLAPI_SDN_GET_PARENT()	434
39.16. SLAPI_SDN_GET_RDN()	434
39.17. SLAPI_SDN_IS_RDN_COMPONENT()	435
39.18. SLAPI_SDN_ISEMPY()	436
39.19. SLAPI_SDN_ISGRANDPARENT()	436
39.20. SLAPI_SDN_ISPARENT()	437
39.21. SLAPI_SDN_ISSUFFIX()	438
39.22. SLAPI_SDN_NEW()	438
39.23. SLAPI_SDN_NEW_DN_BYREF()	439
39.24. SLAPI_SDN_NEW_DN_BYVAL()	439
39.25. SLAPI_SDN_NEW_DN_PASSIN()	440
39.26. SLAPI_SDN_NEW_NDN_BYREF()	441
39.27. SLAPI_SDN_NEW_NDN_BYVAL()	441
39.28. SLAPI_SDN_SCOPE_TEST()	442
39.29. SLAPI_SDN_SET_DN_BYREF()	443
39.30. SLAPI_SDN_SET_DN_BYVAL()	443
39.31. SLAPI_SDN_SET_DN_PASSIN()	444
39.32. SLAPI_SDN_SET_NDN_BYREF()	444
39.33. SLAPI_SDN_SET_NDN_BYVAL()	445
39.34. SLAPI_SDN_SET_PARENT()	446
39.35. SLAPI_SDN_SET_RDN()	446
CHAPTER 40. FUNCTIONS FOR SENDING ENTRIES AND RESULTS TO THE CLIENT	448
40.1. SLAPI_SEND_LDAP_REFERRAL()	448
40.2. SLAPI_SEND_LDAP_RESULT()	449
40.3. SLAPI_SEND_LDAP_SEARCH_ENTRY()	451
CHAPTER 41. FUNCTIONS RELATED TO UTF-8	453
41.1. SLAPI_HAS8THBIT()	454
41.2. SLAPI_UTF8CASECMP()	454
41.3. SLAPI_UTF8CASECMP()	455
41.4. SLAPI_UTF8NCASECMP()	456
41.5. SLAPI_UTF8NCASECMP()	457
41.6. SLAPI_UTF8ISLOWER()	458
41.7. SLAPI_UTF8ISLOWER()	458
41.8. SLAPI_UTF8ISUPPER()	459
41.9. SLAPI_UTF8ISUPPER()	459
41.10. SLAPI_UTF8STRTOLOWER()	460
41.11. SLAPI_UTF8STRTOLOWER()	460
41.12. SLAPI_UTF8STRTOUPPER()	461
41.13. SLAPI_UTF8STRTOUPPER()	461
41.14. SLAPI_UTF8TOLOWER()	462
41.15. SLAPI_UTF8TOLOWER()	463
41.16. SLAPI_UTF8TOUPPER()	463
41.17. SLAPI_UTF8TOUPPER()	464
CHAPTER 42. FUNCTIONS FOR HANDLING VALUES	465

42.1. SLAPI_VALUE_COMPARE()	466
42.2. SLAPI_VALUE_DUP()	467
42.3. SLAPI_VALUE_FREE()	467
42.4. SLAPI_VALUE_GET_BERVAL()	468
42.5. SLAPI_VALUE_GET_FLAGS()	468
42.6. SLAPI_VALUE_GET_INT()	469
42.7. SLAPI_VALUE_GET_LENGTH()	469
42.8. SLAPI_VALUE_GET_LONG()	470
42.9. SLAPI_VALUE_GET_STRING()	470
42.10. SLAPI_VALUE_GET_UINT()	471
42.11. SLAPI_VALUE_GET_ULONG()	472
42.12. SLAPI_VALUE_INIT()	472
42.13. SLAPI_VALUE_INIT_BERVAL()	473
42.14. SLAPI_VALUE_INIT_STRING()	473
42.15. SLAPI_VALUE_INIT_STRING_PASSIN()	474
42.16. SLAPI_VALUE_NEW()	474
42.17. SLAPI_VALUE_NEW_BERVAL()	475
42.18. SLAPI_VALUE_NEW_STRING()	476
42.19. SLAPI_VALUE_NEW_STRING_PASSIN()	476
42.20. SLAPI_VALUE_NEW_VALUE()	477
42.21. SLAPI_VALUE_SET()	478
42.22. SLAPI_VALUE_SET_BERVAL()	478
42.23. SLAPI_VALUE_SET_FLAGS()	479
42.24. SLAPI_VALUE_SET_INT()	479
42.25. SLAPI_VALUE_SET_STRING()	480
42.26. SLAPI_VALUE_SET_STRING_PASSIN()	481
42.27. SLAPI_VALUE_SET_VALUE()	482
42.28. SLAPI_VALUES_SET_FLAGS()	482
CHAPTER 43. FUNCTIONS FOR HANDLING VALUESETS	484
43.1. SLAPI_VALUESET_ADD_VALUE()	484
43.2. SLAPI_VALUESET_ADD_VALUE_EXT()	485
43.3. SLAPI_VALUESET_COUNT()	486
43.4. SLAPI_VALUESET_DONE()	486
43.5. SLAPI_VALUESET_FIND()	487
43.6. SLAPI_VALUESET_FIRST_VALUE()	487
43.7. SLAPI_VALUESET_FREE()	488
43.8. SLAPI_VALUESET_INIT()	488
43.9. SLAPI_VALUESET_NEW()	489
43.10. SLAPI_VALUESET_NEXT_VALUE()	490
43.11. SLAPI_VALUESET_SET_FROM_SMOD()	490
43.12. SLAPI_VALUESET_SET_VALUESET()	491
CHAPTER 44. FUNCTIONS SPECIFIC TO VIRTUAL ATTRIBUTE SERVICE	493
44.1. SLAPI_VATTR_LIST_ATTRS()	493
44.2. SLAPI_VATTR_ATTRS_FREE()	494
44.3. SLAPI_VATTR_SCHEMA_CHECK_TYPE()	495
44.4. SLAPI_VATTR_VALUE_COMPARE()	495
44.5. SLAPI_VATTR_VALUES_FREE()	496
44.6. SLAPI_VATTR_VALUES_GET()	497
44.7. SLAPI_VATTR_VALUES_GET_EX()	498
44.8. SLAPI_VATTR_VALUES_TYPE_THANG_GET()	499
CHAPTER 45. FUNCTIONS FOR MANAGING LOCKS AND SYNCHRONIZATION	501

45.1. SLAPI_DESTROY_CONDVAR()	501
45.2. SLAPI_DESTROY_MUTEX()	501
45.3. SLAPI_LOCK_MUTEX()	502
45.4. SLAPI_NEW_CONDVAR()	502
45.5. SLAPI_NEW_MUTEX()	503
45.6. SLAPI_NOTIFY_CONDVAR()	503
45.7. SLAPI_UNLOCK_MUTEX()	504
45.8. SLAPI_WAIT_CONDVAR()	504
CHAPTER 46. FUNCTIONS FOR MANAGING COMPUTED ATTRIBUTES	506
46.1. SLAPI_COMPUTE_ADD_EVALUATOR()	506
46.2. SLAPI_COMPUTE_ADD_SEARCH_REWRITER()	507
46.3. COMPUTE_REWRITE_SEARCH_FILTER()	507
CHAPTER 47. FUNCTIONS FOR MANIPULATING BITS	509
47.1. SLAPI_ISBITSET_INT()	509
47.2. SLAPI_ISBITSET_UCHAR()	510
47.3. SLAPI_SETBIT_INT()	510
47.4. SLAPI_SETBIT_UCHAR()	511
47.5. SLAPI_UNSETBIT_INT()	511
47.6. SLAPI_UNSETBIT_UCHAR()	512
CHAPTER 48. FUNCTIONS FOR REGISTERING AND UNREGISTERING OBJECT EXTENSIONS ...	513
48.1. SLAPI_GET_OBJECT_EXTENSION()	513
48.2. SLAPI_REGISTER_OBJECT_EXTENSION()	514
48.3. SLAPI_SET_OBJECT_EXTENSION()	515
48.4. SLAPI_UNREGISTER_OBJECT_EXTENSION()	516
CHAPTER 49. FUNCTIONS RELATED TO DATA INTEROPERABILITY	517
49.1. SLAPI_OP_RESERVED()	517
49.2. SLAPI_OPERATION_SET_FLAG()	518
49.3. SLAPI_OPERATION_CLEAR_FLAG()	518
49.4. SLAPI_OPERATION_IS_FLAG_SET()	519
CHAPTER 50. FUNCTIONS FOR REGISTERING ADDITIONAL PLUG-INS	521
50.1. SLAPI_REGISTER_PLUGIN()	521
CHAPTER 51. FUNCTIONS FOR SERVER TASKS	522
51.1. SLAPI_DESTROY_TASK()	523
51.2. SLAPI_PLUGIN_NEW_TASK()	523
51.3. SLAPI_TASK_BEGIN()	524
51.4. SLAPI_TASK_CANCEL()	524
51.5. SLAPI_TASK_DEC_REFCOUNT()	525
51.6. SLAPI_TASK_FINISH()	525
51.7. SLAPI_TASK_GET_DATA()	526
51.8. SLAPI_TASK_GET_REFCOUNT()	526
51.9. SLAPI_TASK_GET_STATE()	527
51.10. SLAPI_TASK_INC_PROGRESS()	527
51.11. SLAPI_TASK_INC_REFCOUNT()	528
51.12. SLAPI_TASK_LOG_NOTICE()	528
51.13. SLAPI_TASK_LOG_STATUS()	529
51.14. SLAPI_PLUGIN_TASK_REGISTER_HANDLER()	529
51.15. SLAPI_PLUGIN_TASK_UNREGISTER_HANDLER()	530
51.16. SLAPI_TASK_SET_DATA()	530

51.17. SLAPI_TASK_SET_CANCEL_FN()	531
51.18. SLAPI_TASK_SET_DESTRUCTOR_FN()	531
51.19. SLAPI_TASK_STATUS_CHANGED()	532
PART V. PARAMETER BLOCK REFERENCE	533
CHAPTER 52. PARAMETERS FOR REGISTERING PLUG-IN FUNCTIONS	534
52.1. PRE-OPERATION/DATA VALIDATION PLUG-INS	534
52.2. POST-OPERATION/DATA NOTIFICATION PLUG-INS	535
52.3. MATCHING RULE PLUG-INS	537
52.4. ENTRY PLUG-INS	537
CHAPTER 53. PARAMETERS ACCESSIBLE TO ALL PLUG-INS	539
53.1. INFORMATION ABOUT THE DATABASE	539
53.2. INFORMATION ABOUT THE CONNECTION	540
53.3. INFORMATION ABOUT THE OPERATION	543
53.4. INFORMATION ABOUT EXTENDED OPERATIONS	544
53.5. INFORMATION ABOUT THE TRANSACTION	544
53.6. INFORMATION ABOUT ACCESS CONTROL LISTS	544
53.7. NOTES IN THE ACCESS LOG	545
53.8. INFORMATION ABOUT THE PLUG-IN	546
53.9. INFORMATION ABOUT COMMAND-LINE ARGUMENTS	548
53.10. INFORMATION ABOUT ATTRIBUTES	549
53.11. INFORMATION ABOUT TARGETS	550
CHAPTER 54. PARAMETERS FOR THE BIND FUNCTION	552
CHAPTER 55. PARAMETERS FOR THE SEARCH FUNCTION	553
55.1. PARAMETERS PASSED TO THE SEARCH FUNCTION	553
55.2. PARAMETERS FOR EXECUTING THE SEARCH	554
55.3. PARAMETERS FOR THE SEARCH RESULTS	555
CHAPTER 56. PARAMETERS THAT CONVERT STRINGS TO ENTRIES	556
CHAPTER 57. PARAMETERS FOR THE ADD FUNCTION	557
CHAPTER 58. PARAMETERS FOR THE COMPARE FUNCTION	558
CHAPTER 59. PARAMETERS FOR THE DELETE FUNCTION	559
CHAPTER 60. PARAMETERS FOR THE MODIFY FUNCTION	560
CHAPTER 61. PARAMETERS FOR THE MODIFY RDN FUNCTION	561
CHAPTER 62. PARAMETERS FOR THE ABANDON FUNCTION	563
CHAPTER 63. PARAMETERS FOR THE MATCHING RULE FUNCTION	564
63.1. QUERY OPERATORS IN EXTENSIBLE MATCH FILTERS	565
CHAPTER 64. PARAMETERS FOR LDBM BACKEND PRE- AND POST-OPERATION FUNCTIONS	567
64.1. PRE-OPERATION PLUG-INS	567
64.2. POST-OPERATION PLUG-INS	567
CHAPTER 65. PARAMETERS FOR LDBM BACK END TRANSACTION PRE- AND POST-OPERATION FUNCTIONS	569
65.1. PRE-OPERATION PLUG-INS	569
65.2. POST-OPERATION PLUG-INS	570

CHAPTER 66. PARAMETERS FOR THE DATABASE	571
66.1. INFORMATION ABOUT THE DATABASE	571
66.2. INFORMATION ABOUT OPERATIONS	571
66.3. INFORMATION ABOUT BACKEND STATE CHANGE	572
CHAPTER 67. PARAMETERS FOR LDAP FUNCTIONS	573
67.1. PARAMETERS FOR LDAP OPERATIONS	573
67.2. PARAMETERS FOR LDAP CONTROL	573
67.3. PARAMETERS FOR GENERATING LDIF STRINGS	575
CHAPTER 68. PARAMETERS FOR ERROR LOGGING	576
CHAPTER 69. PARAMETERS FOR FILTERS	578
69.1. PARAMETERS FOR COMPARISON FILTERS	578
69.2. PARAMETERS FOR FILTER OPERATIONS	579
CHAPTER 70. PARAMETERS FOR PASSWORD STORAGE	580
70.1. PASSWORD STORAGE PLUG-INS	580
70.2. PARAMETERS FOR PASSWORD STORAGE	580
CHAPTER 71. PARAMETERS FOR RESOURCE LIMITS	581
71.1. PARAMETER FOR BINDER-BASED RESOURCE LIMITS	581
71.2. STATUS CODES FOR RESOURCE LIMITS	581
CHAPTER 72. PARAMETERS FOR THE VIRTUAL ATTRIBUTE SERVICE	582
APPENDIX A. REVISION HISTORY	584

PREFACE

This book describes how to write server plug-ins in order to customize and extend the capabilities of the Red Hat Directory Server (Directory Server).

1. REQUIRED CONCEPTS

This book is aimed at those who have at least the following general background:

- An understanding of the Internet and the World Wide Web (WWW).
- An understanding of the Lightweight Directory Access Protocol (LDAP) and the Directory Server. This book does not duplicate basic information on server administration or LDAP concepts. For such information, see the other manuals in the Red Hat Directory Server documentation set, available from <https://access.redhat.com/documentation/en/red-hat-directory-server/>.
- Programming experience in C or C++.

PART I. INTRODUCTION TO DIRECTORY SERVER PLUG-INS

CHAPTER 1. AN OVERVIEW OF DIRECTORY SERVER PLUG-INS

This chapter introduces you to Red Hat Directory Server plug-ins and discusses the different types of plug-ins that you can write.

If you have already written a plug-in for Directory Server, refer to [Section 1.4, “Using Directory Server Plug-in APIs”](#) for information on migrating your plug-in to the latest version of Directory Server.

1.1. ABOUT DIRECTORY SERVER PLUG-INS

If you want to extend the capabilities of the Directory Server, you can write your own Directory Server plug-in. A *server plug-in* is a shared object or library that contains custom functions that you write.

By writing your own plug-in functions, you can extend the functionality of the Directory Server. The following are some examples of what you can do with Directory Server plug-ins:

- You can design an action that the Directory Server performs before the server processes an LDAP action. For example, you can write a custom function to validate data before the server performs an LDAP operation on the data.
- You can design an action that the Directory Server performs after the server successfully completes an LDAP operation. For example, you can send mail to a client after an LDAP operation is successfully completed.
- You can define extended operations, as defined in the LDAP v3 protocol.
- You can provide alternative matching rules when comparing certain attribute values.

1.2. HOW DIRECTORY SERVER PLUG-INS WORK

You can configure Directory Server to load your plug-ins when Directory Server is started. After the plug-in has loaded, the Directory Server will resolve calls made to your plug-in functions as it processes the LDAP requests contained in your applications.

Internally, the Directory Server has hooks that allow you to register your own functions to be called when specific events occur. For example, the Directory Server has hooks to call a registered plug-in in the following situations:

- Before performing an LDAP operation (for example, before an entry is added to the directory).
- When adding, modifying, removing, renaming, or searching for entries in the database.
- After performing an LDAP operation (for example, after an entry is added to the directory).
- Before writing an entry to the database.
- After reading an entry from the database.

- When processing an extended operation.
- When indexing an attribute.
- When filtering search result candidates based on an attribute.

When you register your plug-in functions, you specify the function type and the plug-in type. Together, these specifications indicate when the function is called. Refer to [Section 1.3, “Types of Directory Server Plug-ins”](#).

1.2.1. Calling Directory Server Plug-in Functions

At specific LDAP events, the Directory Server calls all plug-in functions that are registered for that event. For example, before performing an LDAP add operation (an add event), the server calls all plug-in functions registered as pre-operation add functions. When the add operation is completed, the server will call all registered post-operation add functions.

In most plug-in function calls, the server passes a *parameter block* structure (also called a *pblock* or *Slapi_PBlock*) to the function. The parameter block contains data relevant to the operation. In most Directory Server plug-in functions you write, you access and modify the data in the parameter block.

For example, when the Directory Server receives an LDAP add request, the server does the following:

1. Parses the request, and retrieves the new DN and the entry to be added.
2. Places pointers to the DN and the entry in the parameter block.
3. Calls any registered pre-operation add functions, passing the parameter block to these functions.
4. Calls the registered database add function (which is responsible for adding the entry to the directory database), and passes to it the parameter block.
5. Calls any registered post-operation add functions, passing the parameter block to these functions.

If you are writing a function that is invoked before an LDAP operation is performed, you can prevent the operation from being performed. For example, you can write a function that validates data before a new entry is added to the directory. If the data is not valid, you can prevent the LDAP add operation from occurring and return an error message to the LDAP client.

In some situations, you can also set the data that is used by the server to perform an operation. For example, in a pre-operation add function, you can change an entry before it is added to the directory.

1.2.2. The Directory Server Architecture

Internally, the Directory Server consists of two major subsections, the *front-end* and the *backend*.

The front-end receives LDAP requests from clients and processes those requests. When processing requests, the front-end calls functions in the backend to read and write data. The front-end then sends the results back to the client.

The backend reads and writes data to the database containing the directory entries. The backend abstracts the database from the front-end. The data stored in a backend is identified by the suffixes that the backend supports. For example, a backend that supports the **dc=example,dc=com** suffix contains directory entries that end with that suffix.

The following diagram illustrates the Directory Server architecture.

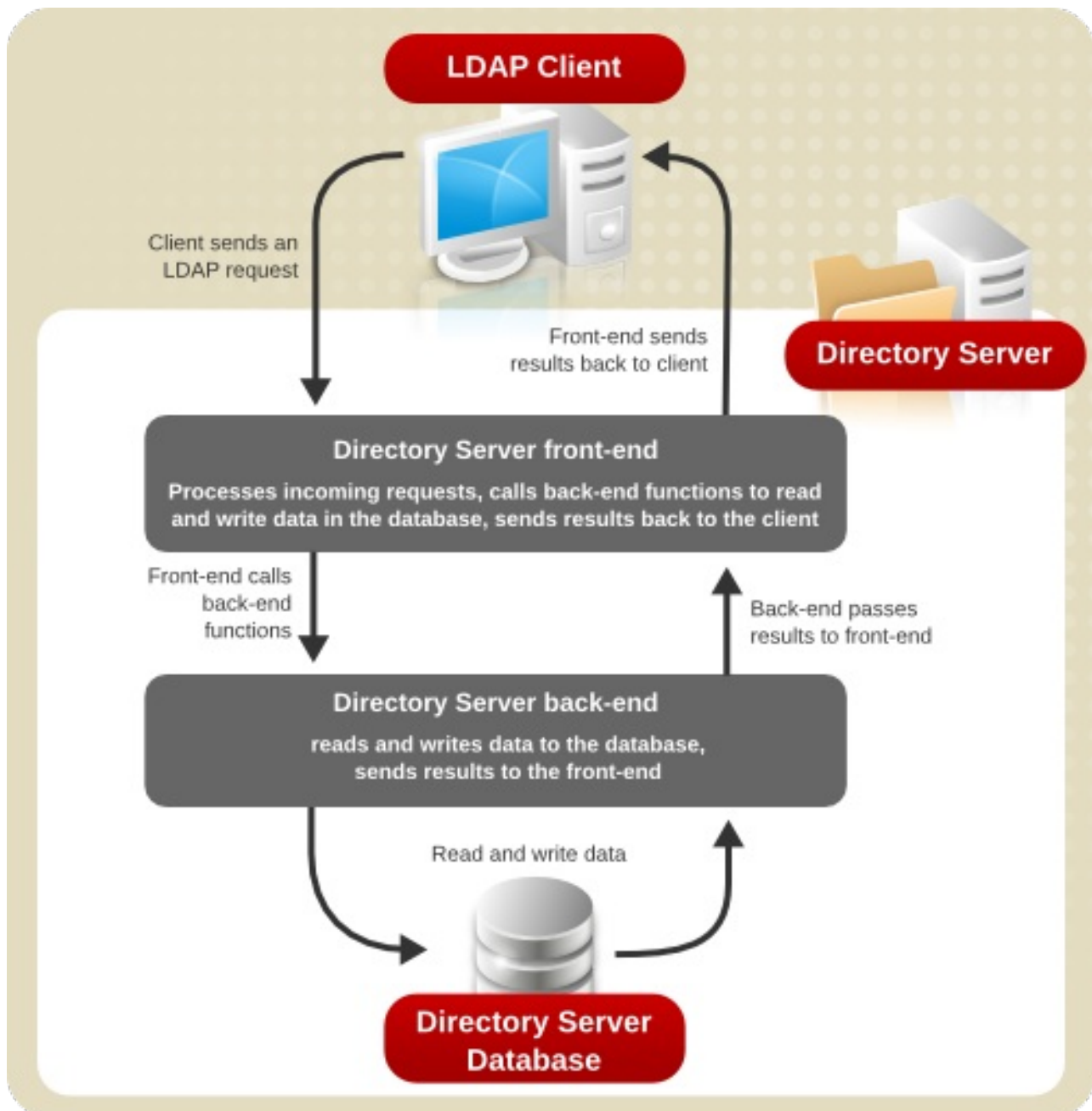


Figure 1.1. Directory Server Architecture

1.3. TYPES OF DIRECTORY SERVER PLUG-INS

Several types of plug-ins can be written for Directory Server:

- *Pre-operation/data validation*

The server calls a pre-operation/data validation plug-in function before performing an LDAP operation.

The main purpose of this type of plug-in is to validate data before the data is added to the directory or before it is used in an operation. For example, a bind pre-operation plug-in can be used to validate authentication or even to provide alternate authentication mechanisms, if passwords are stored in an external database.

- *Post-operation/data notification*

The server calls a post-operation/data notification plug-in function after performing an LDAP operation.

The main purpose of this type of plug-in is to invoke a function after a particular operation is executed. For example, you can write a plug-in that sends email to users if their entries are modified.

The post-operation plug-ins are called after an operation completes and returns the results for both success and failure. The returned result code can be pulled from the previous operation using the **SLAPI_RESULT_CODE** pblock parameter. For example:

```
int return_code;
if (slapi_pblock_get(pb, SLAPI_RESULT_CODE, &return_code) != 0) {
    // something went wrong
}
```

- *Entry storage and entry fetch*

The server calls an entry storage plug-in function immediately before writing data to the database backend. The server calls entry fetch plug-in functions after retrieving an entry from the database backend.

For example, you can create an entry storage plug-in that encrypts an entry before it is saved to the database and an entry fetch plug-in that decrypts an entry after it is read from the database.

- *Extended operation*

The server calls an extended operation plug-in function when the client requests an operation by OID. Extended operations are defined in LDAP v3 and are described in more detail in [Chapter 10, Writing Extended Operation Plug-ins](#).

- *Syntax*

The server calls a syntax plug-in function when getting a list of possible candidates for a search. The server also calls these functions when adding or deleting values from certain attribute indexes.

Syntax plug-in functions can define the comparison operations used in searches. For example, you could use a syntax plug-in function to define how the “equals” comparison works for case-insensitive strings.

- *Matching rule*

The server calls matching rule plug-in functions when the client sends a search request with an extensible matching search filter. You can also write matching rule plug-in functions that the server calls when indexing attributes for the backend database.

The following diagram illustrates how some of these different plug-in types fit into the Directory Server architecture.

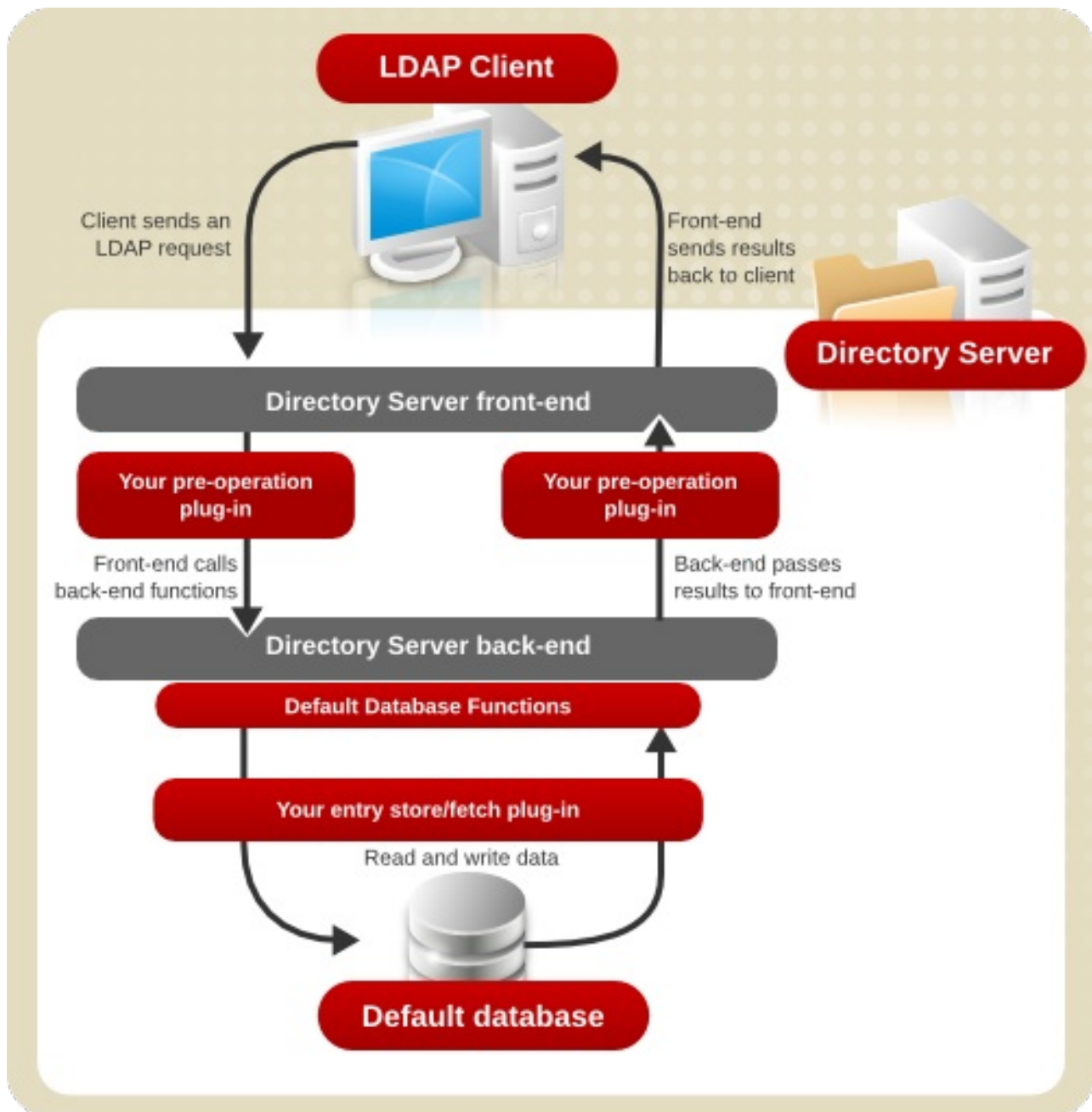


Figure 1.2. Architecture of the Directory Server and Server Plug-ins

1.4. USING DIRECTORY SERVER PLUG-IN APIS

When using Directory Server plug-in APIs, you should be aware of the following:

- Plug-in files and examples are installed separately from other Directory Server packages, available at the Fedora Directory Server CVS repos, <https://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/plugins> and <https://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/slapi/test-plugins>. These sample plug-in files can be installed in any directory.
- The main header file is `/usr/lib64/dirsrv/plugins/slapi-plugin.h`. On some systems, the **389-ds-base-devel** package installs this header file automatically in `/usr/include/dirsrv/slapi-plugin.h`.

- Plug-in directives are contained in the `/etc/dirsrv/slapd-instance/dse.ldif` file. These plug-in configuration entries can be added or modified using LDAP tools like **ldapmodify** or by stopping the server and manually editing the `dse.ldif` file. (Either way, the server must be restarted before the plug-in changes take effect.)

See chapter 2, "Core Server Configuration Reference," in the *Configuration, Command, and File Reference* for information on the syntax of the `dse.ldif` file.

- Database plug-ins are not supported in Directory Server. Be sure to use the pre-operation, post-operation, or extended operation API to register plug-in functions.
- To comply with IPv6, Red Hat Directory Server Plug-in Guide can use the **SLAPI_CONN_CLIENTEDADDR** and **SLAPI_CONNSERVERADDR** parameters to allow plug-ins to recognize the IP address of the LDAP client and server. These new parameters use the Netscape Portable Runtime (NSPR) **PRNetAddr** structure for storing the IP addresses. Because of this, the NSPR files are either shipped with Directory Server or are available from the operating system's **nspr-devel** package, with the header files installed in `/usr/include/nspr4`.

The NSPR API allows compliant applications to use system facilities such as threads, thread synchronization, I/O, interval timing, atomic operations, and several other low-level services independent of platform.

1.5. DEPRECATED FUNCTIONS AND REPLACEMENTS

Several functions have been deprecated in the Directory Server plug-in API. The deprecated functions are still supported for backward compatibility. They are, however, no longer documented and should not be used.

[Table 1.1, "Deprecated Functions and Suggested Replacements"](#) lists the deprecated functions and the functions to use in their place.

Table 1.1. Deprecated Functions and Suggested Replacements

Deprecated Function	Suggested Replacement Function
These functions deal with bervals . These functions are deprecated and their use is not recommended. For each deprecated function, a corresponding function is listed in slapi-plugin.h with an _sv extension that uses Slapi_Values instead of bervals .	
<code>slapi_entry_attr_merge()</code>	<code>slapi_entry_attr_merge_sv()</code>
<code>slapi_entry_add_values()</code>	<code>slapi_entry_add_values_sv()</code>
<code>slapi_entry_delete_values()</code>	<code>slapi_entry_delete_values_sv()</code>
<code>slapi_entry_attr_replace()</code>	<code>slapi_entry_attr_replace_sv()</code>
<code>slapi_attr_get_values()</code>	<code>slapi_attr_value_find()</code>
<code>slapi_attr_get_oid()</code>	<code>slapi_attr_get_oid_copy()</code>

Deprecated Function	Suggested Replacement Function
<code>slapi_pw_find()</code>	<code>slapi_pw_find_sv()</code>
<code>slapi_call_syntax_values2keys()</code>	<code>slapi_call_syntax_values2keys_sv()</code>
<code>slapi_call_syntax_assertion2keys_ava()</code>	<code>slapi_call_syntax_assertion2keys_ava_sv()</code>
<code>slapi_call_syntax_assertion2keys_sub()</code>	<code>slapi_call_syntax_assertion2keys_sub_sv()</code>
<code>slapi_entry_attr_hasvalue()</code>	<code>slapi_entry_attr_has_syntax_value()</code>
<p>The following internal-operation calls are deprecated. The new internal operation functions that are defined in slapi-plugin.h take a <code>Slapi_PBlock</code> for extensibility and support the new plug-in configuration capabilities.</p>	
<code>slapi_search_internal_callback()</code>	<code>slapi_search_internal_callback_pb()</code>
<code>slapi_search_internal()</code>	<code>slapi_search_internal_pb()</code>
<code>slapi_modify_internal()</code>	<code>slapi_modify_internal_pb()</code>
<code>slapi_add_internal()</code>	<code>slapi_add_internal_pb()</code>
<code>slapi_delete_internal()</code>	<code>slapi_delete_internal_pb()</code>
<code>slapi_modrdn_internal()</code>	<code>slapi_modrdn_internal_pb()</code>
<p>The following functions are not multi-thread safe; they return a pointer to unprotected data. Use the new functions in slapi-plugin.h that end in _copy() instead.</p>	
<code>slapi_get_supported_controls()</code>	<code>slapi_get_supported_controls_copy()</code>
<code>slapi_get_supported_extended_ops()</code>	<code>slapi_get_supported_extended_ops_copy()</code>
<code>slapi_get_supported_saslmechanism()</code>	<code>slapi_get_supported_saslmechanisms_copy()</code>
<p>All of these functions are designed to work with entry DNs, but they no longer work with the Slapi_DN structure. These DN functions have been deprecated. New DN management functions, slapi_sdn_*, have been added to slapi-plugin.h.</p>	
<code>slapi_dn_isparent()</code>	<code>slapi_sdn_isparent()</code>
<code>slapi_dn_issuffix()</code>	<code>slapi_sdn_issuffix()</code>
<code>slapi_dn_normalize()</code>	<code>slapi_sdn_get_ndn()</code>

Deprecated Function	Suggested Replacement Function
<code>slapi_dn_parent()</code>	<code>slapi_sdn_get_parent()</code>
<code>slapi_dn_plus_rdn()</code>	<code>slapi_sdn_add_rdn()</code>
<code>slapi_isbesuffix()</code>	<code>slapi_be_issuffix()</code>
<code>slapi_dn_beparent()</code>	<code>slapi_sdn_get_backend_parent()</code>
<code>slapi_dn_ignore_case()</code>	<div> <code>slapi_sdn_get_ndn()</code> (to normalize the case) </div> <div> <code>slapi_sdn_compare()</code> (to compare DNSs, regardless of case) </div>

CHAPTER 2. WRITING AND COMPILING PLUG-INS

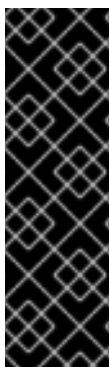
This chapter provides an introduction on how to write and compile Red Hat Directory Server. [Chapter 3, *Configuring Plug-ins*](#) describes how to load the plug-in into the Directory Server configuration after it has been successfully compiled.

If you have already written a plug-in for the Directory Server, see [Section 1.4, “Using Directory Server Plug-in APIs”](#) for information on migrating a custom plug-in to the latest version of Directory Server.

2.1. WRITING A PLUG-IN FUNCTION

To write a Directory Server plug-in, the plug-in code must:

- Include the API header file.
- Set the function parameters using the parameter block.
- Call the front-end.
- Specify the function return value.



IMPORTANT

Any custom plug-in files must be located in the default plug-ins directory, ***install_directory/ldapserver/ldap/servers/plugins*** on Red Hat Enterprise Linux 7. SELinux policies set rules on what directories the Directory Server processes are allowed to access, and any required files (such as plug-ins) must be in the default directories or the server cannot load them.

If the plug-in files are not in the default location, then the SELinux policies must be manually updated to include the alternate location.

For additional information on writing specific plug-in types, see the following sections:

- [Chapter 6, *Writing Pre- and Post-operation Plug-ins*](#)
- [Chapter 9, *Writing Entry Store/Fetch Plug-ins*](#)
- [Chapter 10, *Writing Extended Operation Plug-ins*](#)

2.1.1. Including the API Header File

The interface to the Directory Server plug-in API is located in the ***slapi-plugin.h*** header file. Include this header file in all custom plug-ins:

```
#include "slapi-plugin.h"
```

Then provide the path to this file with the compiler command line:

```
gcc ... -I /usr/lib64/dirsrv/plugins myplugin.c
```

On many platforms, the header file is provided by the **389-ds-base-devel** package. This package installs the file in ***/usr/include/dirsrv***, so the relative path can be used in the

include statement — `#include .dirsrv/slapi-plugin.h` — and there is no need to set the path in the `-I` since the compiler will find it in the default location. This is recommended.

2.1.2. Passing Data with Parameter Blocks

Plug-in functions often make use of the [Slapi_PBlock](#) parameter block (pblock) for passing information to and from the Directory Server. All functions use a pblock.

These functions take a single argument of type **Slapi_PBlock*** to the Directory Server. This argument contains the parameter values needed to complete the function request. A plug-in function typically has a prototype similar to the following:

```
int myFunction( Slapi_PBlock *pb [optional_arguments] );
```

In this prototype, *pb* is the parameter block that contains the parameters pertaining to the operation or function.

For example, the parameter block for an add operation will contain the target DN and the entry to be added; the parameter block for a bind operation will contain the DN of the user, the authentication method, and the user's credentials.

2.1.3. Working with Parameter Blocks

In the functions that you write, you set values in the parameter block that pertain to the operation you are performing. You can also get data from the parameter block that you can use within the plug-in functions. This process is described in [Section 2.1.3.1, “Getting Data from the Parameter Block”](#).

You can also set values in the parameter block during the processing of the plug-in functions. The Directory Server can then use the new data values to complete an operation which it might be processing. This process is described in [Section 2.1.3.2, “Setting Data in the Parameter Block”](#).

Some of the data that you can retrieve from the parameter block includes entries, attributes, search filters, and distinguished names (DNs). After you have retrieved a piece of data, you can manipulate it using the front-end API functions. For example, you can use front-end API functions to verify that an entry complies with the schema or you can split up a search filter into its individual components. This process is described in [Section 2.1.4, “Calling Front-end Functions”](#).

2.1.3.1. Getting Data from the Parameter Block

When the Directory Server calls the plug-in function, it passes any relevant data to the function in a parameter block defined as a data type found in [Slapi_PBlock](#). To access the data in a parameter block, call the [slapi_pblock_get\(\)](#) function.



NOTE

slapi_pblock_get() often returns a pointer to the actual data. When you call custom functions to modify the value retrieved by **slapi_pblock_get()**, you are modifying the actual data in the parameter block, not a copy of the data.

In [Example 2.1, “searchdn_preop_search\(\) Function”](#), the **searchdn_preop_search()**

function gets the DN of the base DN for the LDAP search operation. It then normalizes the DN, converts all characters to lowercase, and writes the converted DN to the error log. The function makes a copy of the DN so that the actual data in the pblock are not changed.



NOTE

The data in the pblock can be modified, but this is not common, and be careful doing that.

Since there is a copy of the data (using [slapi_ch_strdup\(\)](#)), free the memory at the end (using [slapi_ch_free_string\(\)](#)).

Example 2.1. searchdn_preop_search() Function

```
#include "slapi-plugin.h"
...
int searchdn_preop_search( Slapi_PBlock *pb )
{
    Slapi_DN *dn;
    char *mydn;
    /* Indicate the point when the plug-in starts executing */
    slapi_log_error( SLAPI_LOG_PLUGIN, "searchdn_preop_search" , "****
    PREOPERATION SEARCH PLUGIN ***\n" );

    /* Get the base DN of the search from the parameter block. */
    slapi_pblock_get( pb, SLAPI_SEARCH_TARGET, &dn);

    /* make a copy of the DN */
    mydn = slapi_ch_strdup(dn);

    /* Normalize the copy of the DN and convert it to lowercase */
    slapi_dn_normalize_case( mydn );

    /* Log the normalized DN */
    slapi_log_error( SLAPI_LOG_PLUGIN, "searchdn_preop_search" ,"Normalized
    DN: %s\n" , mydn );

    /* more processing . . . */

    /* free the copy */
    slapi_ch_free_string(&mydn);

    return( 0 );
}
```

SLAPI_SEARCH_TARGET identifies the parameter in the parameter block that contains the base DN of the search. The plug-in should arrange to release the resources for any data allocated. In this case, the plug-in should also provide a stop callback which would get the db from the pblock and free the resources associated with it.

For a complete listing of the parameter block IDs, see [Part V, “Parameter Block Reference”](#).

2.1.3.2. Setting Data in the Parameter Block

To modify the value of a parameter in the parameter block, call the [slapi_pblock_set\(\)](#) function. For example, you can call **slapi_pblock_set()** to change the value of the **SLAPI_PLUGIN_PRIVATE** parameter, which stores private data for the plug-in.

In the following example, the **ldif_back_init()** function sets the value of the **SLAPI_PLUGIN_PRIVATE** parameter to the context of the database.

```
#include "slapi-plugin.h";
...
int
ldif_back_init( Slapi_PBlock *pb )
{
    LDIF *db; /* context of the database */
    ...
    /* Allocate space for the database context, which contains information
    about the
    database and a pointer to the list of entries. */

    if ( slapi_pblock_set( pb, SLAPI_PLUGIN_PRIVATE, (void *) db ) == -1 )
    {
        slapi_log_error( SLAPI_LOG_PLUGIN, "ldif_back_init" ,
            "Unable to store database information\n" );
    }
    ...
}
```

This example uses the [slapi_log_error\(\)](#) function to notify the user if an error occurred.

In this code example, **SLAPI_PLUGIN_PRIVATE** identifies the parameter in the parameter block that contains private data for use in the database functions. For a complete listing of the parameter block IDs, see [Part V, “Parameter Block Reference”](#).

2.1.4. Calling Front-end Functions

The types of data that you can get from a parameter block include entries, attributes, distinguished names, and search filters. If you want to manipulate these data items, you can call the associated front-end API functions provided with the Directory Server. For example, using the front-end API functions, you can:

- Write messages to the error log.
- Get the attributes of an entry.
- Get or set the DN of an entry.
- Add or delete the values of an attribute.
- Determine the OID of an attribute.
- Determine the type of a search filter.

For more information on the front-end API, see [Chapter 5, Frontend API Functions](#) and [Part IV, “Function Reference”](#).

2.1.5. Plug-in Return Values

If the plug-in function is successful, it should return `0` to the front-end. If it is not successful, it should return a non-zero value, and you should call the front-end API function `slapi_log_error()` to log an error message to describe the problem. Refer to [Section 5.1, “Logging Messages”](#) for more information.

In some cases, you may need to send an LDAP result back to the client. For example, if you are writing a pre-operation bind function and an error occurs during the processing of the function, the function should return a non-zero value, log an error message, and send the appropriate LDAP result code back to the client. For information on the appropriate result code to return to the client, see the chapter that documents the type of plug-in you are writing.

2.2. WRITING PLUG-IN INITIALIZATION FUNCTIONS

Before the Directory Server can call the plug-in function, the function must be properly initialized. To do this, you must write an initialization function for the server plug-in. The initialization function should:

1. Specify the plug-in compatibility version.
2. Register each of the plug-in functions.
3. Return a value to the Directory Server.



NOTE

The initialization function should not do anything more than these three steps. For example, the init function should not attempt to perform an internal search or other internal operation, because the all of the subsystems are not up and running during the init phase.

To perform additional configuration or initialization, use a start function. This function is specified by using `slapi_pblock_set()` with the **SLAPI_PLUGIN_START_FN** parameter in the initialization function.

Your initialization function should have a prototype similar to the following:

```
int my_init_function( Slapi_PBlock pb );
```

In the initialization function, the Directory Server passes a single argument of the type `Slapi_PBlock*`.

2.2.1. Specifying Directory Server Compatibility

Specify the compatibility version of the plug-in so that the Directory Server can determine whether it supports the plug-in.

To specify the plug-in compatibility version, call the `slapi_pblock_set()` function, and set the **SLAPI_PLUGIN_VERSION_03** parameter to the version number associated with the plug-in. For example:

```
slapi_pblock_set(pb, SLAPI_PLUGIN_VERSION, SLAPI_PLUGIN_VERSION_03);
```

Plug-in version 3 (**SLAPI_PLUGIN_VERSION_03**) is compatible with versions 6.x and later of the Directory Server. Two other plug-in versions are available for backward compatibility with older version of Directory Server; otherwise, these versions are deprecated:

- **SLAPI_PLUGIN_VERSION_01** is compatible with versions 3.x and 4.x of the Directory Server.
- **SLAPI_PLUGIN_VERSION_02** is compatible with version 4.x of the Directory Server.

2.2.2. Specifying Information about the Plug-in

You specify information about the plug-in, such as a description, with the **Slapi_PluginDesc** structure. For details on this data structure, see [Section 14.30](#), “**Slapi_PluginDesc**”.

It is recommended that you include a plug-in description because the Red Hat Console displays this information as part of the server configuration information.

To specify plug-in information, call the [slapi_pblock_set\(\)](#) function, and set the **SLAPI_PLUGIN_DESCRIPTION** parameter to this structure. For example:

Specifying Plug-in Information

```
/* Specify information about the plug-in */
Slapi_PluginDesc mypdesc = { "test-plugin" , "example.com" "0.5" , "sample
pre-operation plugin" };
...
/* Set this information in the parameter block */
slapi_pblock_set( pb, SLAPI_PLUGIN_DESCRIPTION, (void*)&mypdesc );
```

This example code specifies the following plug-in information:

- The unique identifier for the server plug-in is **test-plugin**.
- The vendor of the server plug-in is **example.com**.
- The version of the server plug-in is **0.5**.

This version is intended to be used for tracking purposes only; it is not the same as the server compatibility version number specified by the **SLAPI_PLUGIN_VERSION** parameter (see [Section 2.2.1](#), “**Specifying Directory Server Compatibility**” for details on this parameter). As you make changes to the plug-in code, you can track the version distributed using the number contained in the **Slapi_PluginDesc** structure.

- The description of the server plug-in is contained in the data structure value sample pre-operation plug-in.

2.2.3. Registering Your Plug-in Functions

Whether the plug-in is registered through the initialization function depends on the type of function being registered.

For some plug-in types, you do not need to register the plug-in function from within the initialization function. For example, you register entry store and entry fetch plug-ins by specifying the function names in the **plugin** directive in the **dse.ldif** file.

For other plug-in types, such as pre-operation plug-ins and post-operation plug-ins, the Directory Server gets the pointer to the function from a parameter in the initialization function parameter block. In these cases, you use the `slapi_pblock_set()` function to specify the name of the plug-in function.

The full list parameters that you can use to register the plug-in functions are listed in [Chapter 52, Parameters for Registering Plug-in Functions](#).

For example, if you want to register `searchdn_preop_search()` as a pre-operation search function, include the following code in the initialization function:

```
slapi_pblock_set( pb, SLAPI_PLUGIN_PRE_SEARCH_FN, (void *)
searchdn_preop_search )
```

`SLAPI_PLUGIN_PRE_SEARCH_FN` is the parameter that specifies the pre-operation plug-in function for the LDAP search operation.



NOTE

If the plug-in functions are not registered, the Directory Server will not call them. All custom plug-in functions should be registered.

More than one plug-in function can be registered in the initialization function; it is not necessary to write an initialization function for each plug-in function that needs to be registered. However, define a different initialization function for each type of plug-in being registered. An **object** type plug-in can register any type of plug-in in its init function, so when writing a plug-in that must perform several different types of operations, an **object** plug-in is probably the best approach.

2.2.4. Returning a Value to the Directory Server

If the initialization function is successful, it should return `0`. If an error occurs, it should return `-1`, in which case the Directory Server will exit.

2.2.5. Example of an Initialization Function

The following is an example of an initialization function that registers the pre-operation plug-in function `searchdn_preop_search()`. After the initialization function returns, the server will call `searchdn_preop_search()` before each LDAP search operation is executed.

```
#include "slapi-plugin.h"
...
#ifdef _WIN32
__declspec(dllexport)
#endif

int
searchdn_preop_init( Slapi_PBlock *pb )
{
    /*Specify the version of the plug-in (set this to "03" )*/
    if ( slapi_pblock_set( pb, SLAPI_PLUGIN_VERSION, SLAPI_PLUGIN_VERSION_03
) != 0 ||

    /* Set up theserver to call searchdn_preop_search()
```

```
before each LDAP search operation. */

slapi_pblock_set( pb, SLAPI_PLUGIN_PRE_SEARCH_FN, (void
*)searchdn_preop_search ) !=0 ) {

    /* If a problem occurs, log an error message, return -1.*/
    slapi_log_error(SLAPI_LOG_PLUGIN, "searchdn_preop_init" ,
        "Error registeringfunction.\n" );
    return( -1 );
}

/*If the plug-in was successfully registered, log a
message and return 0. */
slapi_log_error( SLAPI_LOG_PLUGIN, "searchdn_preop_init" ,
    "Plug-in successfully registered.\n" );
return( 0 );
}
```

2.3. COMPILING A DIRECTORY SERVER PLUG-IN

Keep in mind the following tips when compiling the server plug-in:

- Compile and link the code as a shared object or library.

On Red Hat Enterprise Linux, use **-shared** with **gcc**:

```
gcc -shared
```

On Solaris, use this as the link command:

```
ld -G objects -o shared_object
```

Some compilers require special flags to be used when compiling code a shared object. On Linux, this is done with the **-fPIC** flag with **gcc** to generate position-independent code. On Solaris, this uses the **-KPIC** flag. Consult the documentation for the platform compiler and linker for details.

- Ensure that the directory with the **slapi-plugin.h** file is in the include path directive. For example:

```
gcc -I /usr/lib64/dirsrv/plugins
```

It is also possible to define the location of the header file in the plug-in file. Some platforms install the header file with the **389-ds-base-devel** package, which installs the header file in the **/usr/include/dirsrv**. In that case, it is not necessary to specify the full path in the plug-in file.

```
#include dirsrv/slapi-plugin.h
```

- All plug-in functions can be compiled in a single library. Although different types of plug-in functions can be included in the same library, separate initialization functions need to be written for each type of plug-in function. It is also possible to use an **object** type plug-in.

See [Chapter 3, *Configuring Plug-ins*](#) for details on how to specify each initialization function in a directive for a specific type of plug-in.

Example 2.2, “Makefile for Example Plug-ins” assumes that **389-ds-base-devel** package has been installed or that the header file, **slapi-plugin.h**, is in **/usr/include/dirsrv**.

Example 2.2. Makefile for Example Plug-ins

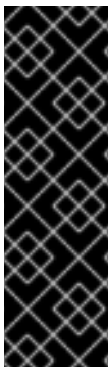
```
# Makefile for Directory Server plug-in examples
#
CC = gcc
LD = gcc
CFLAGS = -fPIC -I /usr/include/nspr4 -Wall
LDFLAGS = -shared -z defs -L/usr/lib64/dirsrv -lslapd
OBJS = testsaslbind.o testextendedop.o testpreop.o testpostop.o
testentry.o
all: libtest-plugin.so
libtest-plugin.so: $(OBJS)
$(LD) $(LDFLAGS) -o $@ $(OBJS)
.c.o:
$(CC) $(CFLAGS) -c $<
clean:
-rm -f $(OBJS) libtest-plugin.so
```

CHAPTER 3. CONFIGURING PLUG-INS

After compiling the server plug-in, configure the Red Hat Directory Server so that it correctly loads the plug-in. The following sections describe this process.

3.1. CREATING A PLUG-IN CONFIGURATION FILE

To add the plug-in to the Directory Server configuration, you need to create an LDIF representation of the plug-in entry, add the plug-in entry to the Directory Server configuration, and reload the server configuration. This section illustrates how to create the plug-in entry. [Section 3.3, “Loading the Plug-in Configuration File”](#) explains how to add the plug-in entry to the Directory Server configuration and reload the server configuration.



IMPORTANT

Any custom plug-in files must be located in the default plug-ins directory, **`install_directory/ldapserver/ldap/servers/plugins`** on Red Hat Enterprise Linux 7. SELinux policies set rules on what directories the Directory Server processes are allowed to access, and any required files (such as plug-ins) must be in the default directories or the server cannot load them.

If the plug-in files are not in the default location, then the SELinux policies must be manually updated to include the alternate location.

The plug-in configuration file must be an LDIF file written in ASCII format. The following listing shows the contents of an example plug-in configuration file.

Example 3.1. An Example Plug-in Configuration File

```
dn: cn=Example Plug-in,cn=plugins,cn=config
objectclass: top
objectclass: nsSlapdPlugin
objectclass: extensibleObject
cn: Example Plug-in
nsslapd-pluginpath: /servers/lib/test-plugin.so
nsslapd-plugininitfunc: searchdn_preop_init
nsslapd-plugintype: preoperation
nsslapd-pluginenabled: on
nsslapd-pluginid: Example Pre-operation Plug-in
nsslapd-pluginversion: 1.0
nsslapd-pluginvendor: Example Corporation
nsslapd-plugindescription: This plug-in does ...
nsslapd-pluginPrecedence: 1
```

This example plug-in configuration file defines a plug-in as follows:

- Line 1 sets the DN of the plug-in, which identifies the plug-in:

`dn: cn=Example Plug-in,cn=plugins,cn=config`

Here, the common name of the plug-in is set to **Example Plug-in**. The remainder of the DN entry (**`cn=plugins,cn=config`**) places the entry in the database tree that contains the configuration settings for plug-ins.

- Lines 2-4 declare the object classes of the plug-in.
- Line 5 sets the common name of the plug-in to **Example Plug-in**.
- Line 6 defines the absolute path to the library that implements the plug-in:

nsslapd-pluginpath: /servers/lib/test-plugin.so

- Line 7 identifies the initialization function that the server calls to register the plug-in. In this example, the initialization is set to **searchdn_preop_init**. For information on implementing initialization functions, refer to [Section 2.2, “Writing Plug-in Initialization Functions”](#).
- Line 8 specifies the type of plug-in. In this case, it is a pre-operation plug-in. For a complete list of the types of plug-in you can declare, refer to [Section 3.1.3, “Summary of Plug-in Directives”](#).
- Line 9 specifies whether the plug-in is active by default. The **nsslapd-pluginenabled** attribute can have a value of either **on** or **off**. The following line specifies that the plug-in is active by default:

nsslapd-pluginenabled: on

The Directory Server Console can also activate or deactivate the plug-in after it has been loaded.



NOTE

Whenever plug-in configuration is edited — as when a plug-in is activated or deactivated — then the server must be restarted for the plug-in changes to take effect.

- Line 10 uses the **nsslapd-pluginid** attribute to set the name of the plug-in. The name that you specify here will show up in the Directory Server Console. In this example, the plug-in identification is set to **Example Pre-operation Plug-in**.
- Line 11 sets the version number of the plug-in. This version number is also displayed in the Directory Server Console and is used to track the version of the distributed plug-in. It does not indicate the Directory Server compatibility; this is defined by the plug-in version number described in [Section 2.2.1, “Specifying Directory Server Compatibility”](#).
- Line 12 identifies the vendor or author of the plug-in. In the following line, the vendor is set to *Example Corporation*:

nsslapd-pluginvendor: Example Corporation

- Line 13 sets the description of the plug-in. This is the description visible through the Directory Server Console.
- Line 14 sets the plug-in precedence. This defines the priority that the plug-in has in the execution order of plug-ins for an operation.

3.1.1. Setting Plug-in Dependencies

A plug-in can be dependent on one or more different plug-ins. Any specified plug-in dependencies must start correctly before the associated plug-in will start.

There are two attributes in the plug-in configuration file that specify the dependencies of the plug-in:

- ***nsslapd-plugin-depends-on-named***
- ***nsslapd-plugin-depends-on-type***

Each of these attributes can take multiple values, meaning that the plug-in depends on one or more other plug-ins.

3.1.1.1. Specific Plug-in Dependencies

If you specify the ***nsslapd-plugin-depends-on-named*** attribute in the plug-in configuration file, set its value to the names of one or more plug-ins. For example, in the plug-in configuration file, you could specify the following:

```
nsslapd-plugin-depends-on-named: my_pluginA  
nsslapd-plugin-depends-on-named: vendor_pluginB
```

In this example, the plug-in depends on two specifically named plug-ins: **my_pluginA** and **vendor_pluginB**. This configuration line indicates that before the plug-in can be loaded, the two specifically named plug-ins must be loaded. If either of these two plug-ins fails to load, the Directory Server will exit with a -1 error code.

3.1.1.2. Plug-in Type Dependencies

If you specify the ***nsslapd-plugin-depends-on-type*** attribute in the plug-in configuration file, set its value to one or more plug-in types. For example, in the plug-in configuration file, you could specify the following:

```
nsslapd-plugin-depends-on-type: syntax
```

This configuration line indicates that the plug-in depends on any plug-in of the type syntax. If there is a configured plug-in of type syntax, it must be successfully loaded before the plug-in can be loaded; otherwise, the Directory Server will exit with a -1 error code.

If you specify a plug-in type dependency, the Directory Server will search for any and all plug-ins of the types specified. If none are found, processing will continue without errors. However, the Directory Server must load all plug-ins of the types specified before it can load the plug-in. For a complete list of the supported plug-in types, refer to the [Section 3.1.3, “Summary of Plug-in Directives”](#).

3.1.2. Specifying the Order of Plug-ins

Generally, plug-ins are not called in a specific order. As in, it is not possible to define that Preoperation Plug-in A is *always* called before Preoperation Plug-in B. Ideally, plug-ins should be written so that they function correctly irrespective of the order in which they are called.

It can be convenient, however, to set one plug-in to complete its job before the next plug-in is executed. This can allow more complex interactions between plug-ins and more specific functionality for plug-ins.

To set a general plug-in order (within a plug-in type), define a *precedence* for a plug-in. The precedence sets a value (between 1 and 99) on the plug-in priority, and high-priority plug-ins are called and completed first, going down the line in precedence. The default, if no precedence is set, is 50.

Plug-in precedence sets the execution order *for plug-ins of the same type*. A precedence of 5 on a preoperation plug-in only matters when other preoperation plug-ins are called of an operation. It doesn't affect the execution order of other types of plug-ins called in the operation.

The plug-in precedence is set in the ***nsslapd-pluginPrecedence*** attribute.

nsslapd-pluginPrecedence: 12

Although the plug-in precedence can be set on any plug-in, it is really only relevant for preoperation and postoperation plug-ins. Other plug-in types, like extended operation or matching rule plug-ins, implement callbacks to support specific data types or operations, rather than being called with all other plug-ins of that time for normal LDAP operations.



NOTE

Plug-in *precedence* is not the same as *plug-independence*. Dependencies determine the plug-in startup and shutdown order. Precedence determines the execution order for plug-ins in LDAP operations. The plug-in dependencies do not affect the plug-in precedence, and vice versa.

The server loads plug-ins of the same precedence in alphabetical order, determined by standard ASCII ordering of the **cn** value of the plug-in entry. (The **cn** appears in the plug-in configuration entry under **cn=plugins, cn=config** in the **dse.ldif** file.) Therefore, plug-in names can be used as an extra control measure for the order that plug-ins are loaded and executed.

3.1.3. Summary of Plug-in Directives

The following table summarizes the different types of plug-ins that you can specify in the plug-in configuration file.

Table 3.1. Directives for Specifying Different Plug-in Types

Directive	Description	Example of use
entryfetch	Declares an entry fetch plug-in, which is called by the server after retrieving an entry from the default backend database.	If you encrypt data with an entry store plug-in function before saving the data to the database, you can define an entry_fetch function that decrypts data after reading it from the database.

Directive	Description	Example of use
entrystore	Declares an entry store plug-in, which is called by the server before saving an entry to the default backend database. (If you are writing your own database plug-in, you do not need to use this plug-in.)	You can define an entry_store function to encrypt data before saving the data to the database.
extendedop	Declares an extended operation plug-in, which is called by the server when receiving a request for an extended operation from a client.	Three common server operations use extended operation plug-ins: <ul style="list-style-type: none"> • StartTLS (start_tls_extop.c) • Password Change (passwd_extop.c) • Replication (plugins/replication/)
matchingRule	Declares a matching rule plug-in, which is called by the server when receiving a search request with an extensible matching search filter from a client. This type of plug-in is also called by the server when indexing attributes for the backend database.	
postoperation	Declares a post-operation/data notification plug-in, which is called by the server after performing an LDAP operation.	You can define a data_notification function to send notification to the administrator if certain data has changed.
preoperation	Declares a pre-operation/data validation plug-in, which is called by the server before performing an LDAP operation.	You can define a data_validation function to check new entries before they are added to the directory.

Directive	Description	Example of use
syntax	Declares a syntax plug-in, which is called by the server when getting a list of possible candidates for a search, when determining how to compare values in searches, and when adding or deleting values from certain attribute indexes.	You can define a function that specifies how the “equals” comparison works for case-insensitive strings.
object	Declares an object plug-in. Object plug-ins can install SLAPI_PLUGIN_START_FN , SLAPI_PLUGIN_CLOSE_FN , and SLAPI_PLUGIN_POSTSTART_FN functions. They can also use the slapi_register_plugin() call to register any other kind of plug-in. Object plug-ins are typically used to simplify configuration of a group of related plug-ins (one entry under cn=config instead of many).	You can use this plug-in type when the plug-in does not fit in any of the other types listed in this table. For example, if the plug-in does performs more than one operation, then you should use this directive. This type of plug-in will typically register the types of operations it wants to handle using the internal API.
pwdstoragescheme	This defines a new hashing scheme for passwords.	This kind of plug-in can be used to add support for hashing or encrypting passwords with a method that is not otherwise supported in Directory Server.
bepreoperation	The back end pre-operation plug-in is called after the pre-operation plug-ins and before the operation is applied to the back end.	
bepostoperation	The back end post-operation plug-in is called after the back end was applied to the operation.	
betxnpreoperation	The back end transaction pre-operation plug-in is called after the bepreoperation plug-ins. If a plug-in returns a failure, the entire operation is aborted.	

Directive	Description	Example of use
betxnpostoperation	The back end transaction post-operation plug-in is called after the bepostoperation plug-ins. If a plug-in returns a failure, the entire operation is aborted.	

3.2. PLUG-IN LOGGING FEATURES

For debugging, you can enable access and audit logging for operations a plug-ins executes. For details, see the **nsslapd-logAccess** and **nsslapd-logAudit** parameter in [the corresponding section in the Red Hat Directory Server Configuration, Command, and File Reference](#).

3.3. LOADING THE PLUG-IN CONFIGURATION FILE

After you have written the plug-in configuration file, you need to load it into the `/etc/dirsrv/slapd-instance/dse.ldif` file. You can do this either by using an LDAP utility, such as **ldapmodify**, or by editing the file directly. If you choose to edit the file directly, be sure to shut down the Directory Server first.

The following is an example of an LDAP command that loads the plug-in defined in the configuration file

```
example-plugin.ldif: ldapmodify -h my_host -p 389 -a -D cn= Directory
Manager -W -f example-plugin.ldif
```

After the plug-in configuration has been loaded, restart the Directory Server in order to make calls to the plug-in. The Directory Server Console can restart the server from the **Tasks** tab, or run the following from the command line:

```
# systemctl restart dirsrv.target
```

3.4. PASSING EXTRA ARGUMENTS TO PLUG-INS

The standard method for configuring plug-ins is to provide configuration parameters as attributes and values in the plug-in entry in the **dse.ldif** file. All plug-ins use the **extensibleObject** object class, so any custom attribute can be added to that object class in the schema. A better alternative is to define a custom auxiliary object class which contains the custom plug-in configuration attributes.

Use the plug-in start function (registered in the initialization function with **slapi_pblock_set()** using **SLAPI_PLUGIN_START_FN**) to use the custom configuration. One of the **pblock** parameters passed to the start function is the plug-in entry, as the **SLAPI_ADD_TARGET** parameter. The **slapi_entry_attr_*** family of functions can be used to get and use these values.

Plug-ins can be dynamically configured at run-time by registering DSE callbacks for the plug-in entry with **slapi_config_register_callback_plugin()**.

Example 3.2. Extended Operation Plug-in

```
dn: cn=Test Extended Op,cn=plugins,cn=config
objectClass: top
objectClass: nsSlapdPlugin
objectClass: extensibleObject
cn: Test ExtendedOp
nsslapd-pluginPath: libtest-plugin.so
nsslapd-pluginInitfunc: testexop_init
nsslapd-pluginType: extendedop
nsslapd-pluginEnabled: on
nsslapd-plugin-depends-on-type: database
nsslapd-pluginId: test-extendedop
nsslapd-pluginarg0: 1.2.3.4
```

Example 3.3. MemberOf Plug-in

```
dn: cn=MemberOf Plugin,cn=plugins,cn=config
objectClass: top
objectClass: nsSlapdPlugin
objectClass: extensibleObject
cn: MemberOf Plugin
nsslapd-pluginPath: libmemberof-plugin
nsslapd-pluginInitfunc: memberof_postop_init
nsslapd-pluginType: postoperation
nsslapd-pluginEnabled: off
nsslapd-plugin-depends-on-type: database
memberofgroupattr: member
memberofattr: memberOf
nsslapd-pluginId: memberof
nsslapd-pluginVersion: 1.1.4
nsslapd-pluginVendor: Fedora Project
nsslapd-pluginDescription: memberof plugin
```

This method allows for a much more descriptive configuration, which is easier to maintain. The attributes beginning with **pam** are specific to the plug-in and used for its configuration. The plug-in entry is provided to the plug-in start function as a **Slapi_Entry *** in the **SLAPI_ADD_TARGET pblock** parameter. Use the **slapi_entry_attr_*** family of functions to get the configuration values from that entry.

For additional information, code samples are available from the 389 Directory Server repositories at <http://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/plugins> and <http://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/slapd/test-plugins>. The plug-in samples are in the **/usr/lib64/dirsrv/plugins** directory.

3.5. SETTING THE LOG LEVEL OF THE SERVER

If the functions call the [slapi_log_error\(\)](#) function to write messages to the error log, you need to make sure that the Directory Server is configured to log messages with the severity level that you have specified. The available severity levels are fully documented in [Part IV, “Function Reference”](#).

For example, suppose you call this function in the plug-in:

```
slapi_log_error( SLAPI_LOG_PLUGIN, "searchdn_preop_init", "Plug-in  
successfully registered.\n" );
```

You need to make sure that the Directory Server is configured to log messages with the severity level **SLAPI_LOG_PLUGIN**. Error logging is controlled with the **nsslapd-errorlog-level** attribute in **cn=config**.

CHAPTER 4. AN EXAMPLE PLUG-IN

This chapter provides an example of a pre-operation Red Hat Directory Server plug-in that you can compile and run. It provides both the source code and a makefile that you can use to build the plug-in.

Even though you may not understand all of the functionality contained in the example, all of the necessary concepts are explained in detail in the chapters that follow.

The example shows how to create a pre-operation plug-in for the LDAP search operation. That is, the Directory Server will process the registered plug-in functions before it processes each LDAP search operation. The example contains two primary functions:

- The **test_preop_search()** function logs information about the search, including the base DN of the search, the search scope, and the type of filter used.
- The **test_preop_init()** function is the initialization function that registers **test_preop_search()** as a **SLAPI_PLUGIN_PRE_SEARCH_FN** pre-operation plug-in function for LDAP search operations.

These functions illustrate the data in the parameter block that is available to your function. You can get and manipulate the parameter block data by calling various front-end API functions.

4.1. WRITING THE PLUG-IN EXAMPLE

[Example 4.1, “Sample Pre-Operation Search and Initialization Functions”](#) includes the sample pre-operation search function and the sample initialization function.

Example 4.1. Sample Pre-Operation Search and Initialization Functions

```
#include <stdio.h>
#include <string.h>
#include "slapi-plugin.h"

/* function prototypes */
int test_preop_init( Slapi_PBlock *pb );
int test_preop_search( Slapi_PBlock *pb );

/* Description of the plug-in */
Slapi_PluginDesc srchpdesc = { "test-search", "example.com",
"0.5", "sample pre-operation search plugin" };

/*
 * Initialization function
 *
 * This function registers your plug-in function as a
 * pre-operation search function in the Directory Server.
 * You need to specify this initialization function in the
 * server configuration file so that the server calls
 * this initialization function on startup.
 */

#ifdef _WIN32
__declspec(dllexport)
```

```

#endif

int test_preop_init( Slapi_PBlock *pb )
{
    /* Specify the version of the plug-in ( "03" in this release ) */
    if (slapi_pblock_set( pb, SLAPI_PLUGIN_VERSION,
        SLAPI_PLUGIN_VERSION_03 ) != 0 ||

        /* Specify the description of the plug-in */
        slapi_pblock_set( pb,SLAPI_PLUGIN_DESCRIPTION, (void *)&srchpdesc ) !=
        0 ||

        /*
         * Set test_preop_search() as the function to call before
         * executing LDAP search operations.
         */
        slapi_pblock_set( pb, SLAPI_PLUGIN_PRE_SEARCH_FN, (void *)
        test_preop_search ) !=0 )
    {
        /* Log an error message and return -1 if a problem occurred*/
        slapi_log_error( SLAPI_LOG_PLUGIN, "test_preop_init" ,"Error
        registering the plug-in.\n" );
        return( -1 );
    }

    /* If successful, log a message and return 0 */
    slapi_log_error( SLAPI_LOG_PLUGIN, "test_preop_init" ,"Plug-in
    successfully registered.\n" );
    return( 0 );
}

/*
 * Pre-operation plug-in function for LDAP search operations
 * This function is called by the server before processing an LDAP
 * search operation. The function gets data about the search request
 * from the parameter block and prints the data to the error log.
 */
int test_preop_search( Slapi_PBlock *pb )
{
    char *base, *filter_str, *attr_type, *substr_init, *substr_final;
    char **substr_any;
    int scope, deref, filter_type, i;
    Slapi_Filter *filter;
    struct berval *bval;

    /* Log a message to indicate when the plug-in function starts */
    slapi_log_error( SLAPI_LOG_PLUGIN, "test_preop_search" ,****
    PREOPERATION SEARCH PLUGIN ***\n" );

    /* Get and log the base DN of the search criteria */
    if ( slapi_pblock_get( pb, SLAPI_SEARCH_TARGET, &base ) == 0 ) {
        slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_TARGET"
        ,"%s\n" , base );
    } else {
        slapi_log_error( SLAPI_LOG_FATAL, "test_preop_search", "Error:
        could not get search base\n." );
    }
}

```



```

    }

    /* Get and log the search scope */
    if ( slapi_pblock_get( pb, SLAPI_SEARCH_SCOPE, &scope ) == 0 ) {
        switch( scope ) {
            case LDAP_SCOPE_BASE:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_SCOPE"
, "LDAP_SCOPE_BASE\n" );
                break;

            case LDAP_SCOPE_ONELEVEL:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_SCOPE"
, "LDAP_SCOPE_ONELEVEL\n" );
                break;

            case LDAP_SCOPE_SUBTREE:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_SCOPE"
, "LDAP_SCOPE_SUBTREE\n" );
                break;

            default:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_SCOPE"
, "unknown value specified: %d\n" , scope );
                break;
        }
    } else {
        slapi_log_error( SLAPI_LOG_FATAL, "test_preop_search", "Error: could
not get search scope\n." );
    }

    /* Get and log the alias dereferencing setting */
    if ( slapi_pblock_get( pb, SLAPI_SEARCH_DEREF, &deref ) == 0 ) {
        switch( deref ) {
            case LDAP_DEREF_NEVER:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_DEREF"
, "LDAP_DEREF_NEVER\n" );
                break;

            case LDAP_DEREF_SEARCHING:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_DEREF"
, "LDAP_DEREF_SEARCHING\n" );
                break;

            case LDAP_DEREF_FINDING:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_DEREF"
, "LDAP_DEREF_FINDING\n" );
                break;

            case LDAP_DEREF_ALWAYS:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_DEREF"
, "LDAP_DEREF_ALWAYS\n" );
                break;

            default:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_DEREF" , "unknown
value specified: %d\n" , deref );

```

```
break;
    }
} else {
    slapi_log_error( SLAPI_LOG_FATAL, "test_preop_search", "Error:
could not get search alias deref\n" );
}

/* Get and log the search filter information */
if (slapi_pblock_get(pb,SLAPI_SEARCH_FILTER, &filter)==0 ) {

    /* Get and log the filter type */
    filter_type = slapi_filter_get_choice( filter );
    switch( filter_type ) {
case LDAP_FILTER_AND:
    case LDAP_FILTER_OR:
    case LDAP_FILTER_NOT:
        slapi_log_error( SLAPI_LOG_PLUGIN,
"SLAPI_SEARCH_FILTER" ,
        "Complex search filter. See value of
SLAPI_SEARCH_STRFILTER.\n" );
        break;

    case LDAP_FILTER_EQUALITY:
        slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_FILTER"
, "LDAP_FILTER_EQUALITY\n" );
        break;

    case LDAP_FILTER_GE:
        slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_FILTER"
, "LDAP_FILTER_GE\n" );
        break;

    case LDAP_FILTER_LE:
        slapi_log_error(SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_FILTER"
, "LDAP_FILTER_LE\n" );
        break;

    case LDAP_FILTER_APPROX:
        slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_FILTER"
, "LDAP_FILTER_APPROX\n" );
        break;

    case LDAP_FILTER_SUBSTRINGS:
        slapi_log_error( SLAPI_LOG_PLUGIN,
"SLAPI_SEARCH_FILTER" , "LDAP_FILTER_SUBSTRINGS\n" );

        /*
        * For substring filters, get and log the attribute
type and
        * the substrings in the filter
        */
        slapi_filter_get_subfilt( filter, &attr_type,
&substr_init, &substr_any, &substr_final );
        if ( attr_type != NULL )
            slapi_log_error( SLAPI_LOG_PLUGIN, "\tAttribute
type" , "%s\n" ,attr_type );
```

```

        if ( substr_init != NULL )
            slapi_log_error( SLAPI_LOG_PLUGIN, "\tInitial
substring" , "%s\n" , substr_init );

        if ( substr_any != NULL ) {
            for ( i = 0; substr_any[i] != NULL; i++ ) {
                slapi_log_error( SLAPI_LOG_PLUGIN,
"\tSubstring" , "# %d: %s\n" , i, substr_any[i] );
            }
        }

        if ( substr_final != NULL )
            slapi_log_error( SLAPI_LOG_PLUGIN, "\tFinal
substring" , "%s\n" , substr_final );
        break;

        case LDAP_FILTER_PRESENT:
            slapi_log_error( SLAPI_LOG_PLUGIN,
"SLAPI_SEARCH_FILTER" , "LDAP_FILTER_PRESENT\n" );

            /* For presence filters, get and log the attribute type */
            slapi_filter_get_attribute_type( filter, &attr_type );

            if ( attr_type != NULL )
                slapi_log_error( SLAPI_LOG_PLUGIN, "\tSearch for presence of attr"
, "%s\n" , attr_type );
            break;

            case LDAP_FILTER_EXTENDED:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_FILTER"
, "LDAP_FILTER_EXTENDED\n" );
                break;

            default:
                slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_FILTER" ,
"Unknown filter type: "
                    "slapi_filter_get_choice returned %d\n",
filter_type );
                break;
        }
    } else {
        slapi_log_error( SLAPI_LOG_FATAL, "test_preop_search", "Error:
could not get search filter\n" );
    }

    /* For comparison filters, get and log the attribute type */
    if ( filter_type == LDAP_FILTER_EQUALITY || LDAP_FILTER_GE ||
        LDAP_FILTER_LE || LDAP_FILTER_APPROX )
    {
        slapi_filter_get_ava( filter, &attr_type, &bval );

        if ( ( attr_type != NULL ) && ( bval->bv_val != NULL ) ) {
            slapi_log_error( SLAPI_LOG_PLUGIN, "\tAttribute type"
, "%s\n" , attr_type );
            slapi_log_error( SLAPI_LOG_PLUGIN, "\tAttribute value"

```

```

, "%s\n" , bval->bv_val );
    }

    /* Get and log the string representation of the filter */
    if ( slapi_pblock_get(pb, SLAPI_SEARCH_STRFILTER, &filter_str) == 0
)
        slapi_log_error( SLAPI_LOG_PLUGIN, "SLAPI_SEARCH_STRFILTER"
, "%s\n" , filter_str );

    slapi_log_error( SLAPI_LOG_PLUGIN, "test_preop_search" , "**** DONE
***\n\n" );
    return( 0 );
}

```

4.2. COMPILING THE PLUG-IN EXAMPLE

The makefile in [Example 4.2, “Makefile”](#) can compile the example.

This example assumes that the source code is stored in **srchxmpl.c** and that the plug-in being compiled is **srchxmpl.so**. Additionally, the **389-ds-base-devel** package must have been installed or the header file, **slapi-plugin.h**, is in **/usr/include/dirsrv** specified in the include path. Otherwise, specify **-I /path/to/slapi-plugin.h** in the **CFLAGS** directive.

Example 4.2. Makefile

```

# Makefile for Directory Server plug-in examples
#
CC = gcc
LD = gcc
CFLAGS = -fPIC -I /usr/include/nspr4 -Wall
LDFLAGS = -shared -z defs -L/usr/lib64/dirsrv -lslapd
OBSJS = srchxmpl.o
all: srchxmpl.so
srchxmpl.so: $(OBSJS)
$(LD) $(LDFLAGS) -o $@ $(OBSJS)
.c.o:
$(CC) $(CFLAGS) -c $<
clean:
-rm -f $(OBSJS) srchxmpl.so

```

4.3. REGISTERING THE PLUG-IN

To register this example plug-in, you need to:

1. Create an LDIF configuration file in an ASCII text editor. See [Section 3.1, “Creating a Plug-in Configuration File”](#).
2. Load the plug-in configuration file. See [Section 3.3, “Loading the Plug-in Configuration File”](#).
3. Shut down the Directory Server.

4. Restart the Directory Server.

When you restart the Directory Server, it will read the entries in the **dse.ldif** file, which contains the entry for your new plug-in. It is a good idea to check the plug-ins list in the Directory Server Console to ensure that your plug-in loaded.

4.4. RUNNING THE PLUG-IN

After compiling the plug-in and registering it with the Directory Server, you can make calls that are processed by the plug-in functions.

The first step is to restart the Directory Server and check the error log to see that the plug-in is properly registered. You should see the following line in the error log:

Error log message here!

Verify that the plug-ins log level is selected. Error logging is controlled with the **nsslapd-errorlog-level** attribute in **cn=config**. For the plug-ins log level, set this value to **65536** or set an **OR** for the current value to **65536**. (The sample source code logs messages to the error log with the plug-ins severity level.)

If the plug-in is properly registered, you can then perform a few searches against the directory. The pre-operation search plug-in function should write data about each search to the error log.

PART II. WRITING FUNCTIONS AND PLUG-INS

CHAPTER 5. FRONTEND API FUNCTIONS

The Red Hat Directory Server (Directory Server) provides some general-purpose, frontend API functions that allow you to work with the entries in the Directory Server. This chapter explains how to use the frontend API functions to accomplish various tasks; you can call these functions in your plug-in to interact with the client (for example, to send results or result codes), log messages, and work with entries, attributes, and filters. While all of the functions described here must be used in conjunction with other API functions, understanding how these functions work will help you understand how to program other plug-in API functions.

The frontend functions are declared in the `slapi-plugin.h` header file.

5.1. LOGGING MESSAGES

To write an error message to the error log, call the `slapi_log_error()` function. For example, the following function call writes a message in the error log:

```
slapi_log_error( SLAPI_LOG_PLUGIN, "searchdn_preop_search", "***
PREOPERATION SEARCH PLUGIN ***\n");
```

This call will create the following message in the error log:

```
[01/Oct/2015:02:24:18.577543334 -0700] searchdn_preop_search  ***
PREOPERATION SEARCH PLUGIN ***
```

Make sure that the Directory Server is configured to log messages that have the severity that you specify (for example, **SLAPI_LOG_PLUGIN**). For more information, see [Section 3.5, “Setting the Log Level of the Server”](#). The `slapi_log_error()` function allows additional parameters to be added to the function and to format the output, similar to the standard `printf()` function. This provides a way to add custom parameters and data to the log output.

Some debugging output is expensive to calculate, so check if a certain log level is set before doing additional processing. The `slapi_is_loglevel_set()` function can determine if the log level is set. For example:

```
if (slapi_loglevel_is_set(SLAPI_LOG_PLUGIN)) {
    do some expensive processing for debugging purposes....
    slapi_log_error(SLAPI_LOG_PLUGIN, .my plugin., ... some data...);
}
```

5.2. ADDING NOTES TO ACCESS LOG ENTRIES

When the backend database processes a search operation, it attempts to use indexes to narrow down the list of candidates matching the search criteria. If the back-end is unable to use indexes, it appends the following string to the access log entry for the search:
notes="U"

This note indicates that an unindexed search was performed.

If you are writing your own backend database search function, you can append this note to access log entries by setting the **SLAPI_OPERATION_NOTES** parameter to the flag **SLAPI_OP_NOTE_UNINDEXED**. This parameter identifies any notes that need to be appended

to access log entries. Currently, **SLAPI_OP_NOTE_UNINDEXED** is the only value that you can set for this parameter. In future releases, additional flags will be defined. You will be able to use bitwise **OR** combinations of flags to specify different combinations of notes.

The server appends these notes and writes out the access log entries whenever sending a result or search entry back to the client.

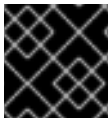
5.3. SENDING DATA TO THE CLIENT

Sometimes you might need to communicate various information directly to the client. This could occur in the following situations:

- If you need to send a result code to the client (for example, to report an error or a successful result to an LDAP operation), call the [slapi_send_ldap_result\(\)](#) function.
- If you are fulfilling a search request and need to send matching entries to the client, call the [slapi_send_ldap_search_entry\(\)](#) function for each entry.
- To refer the LDAP request to a different LDAP server, call the [slapi_send_ldap_referral\(\)](#) function.

For example, the following statement sends an LDAP_SUCCESS status code back to the client:

```
slapi_send_ldap_result( pb, LDAP_SUCCESS, NULL, "The operation was  
processed successfully.\n", 0, NULL );
```



IMPORTANT

Ensure that you send only one result per operation to the client.

5.4. DETERMINING IF AN OPERATION WAS ABANDONED

At any point in time, the client can choose to abandon an LDAP operation. When writing database functions, remember that you should periodically check to see if the operation has been abandoned.

To determine if an operation has been abandoned, call [slapi_op_abandoned\(\)](#). For example:

```
if ( slapi_op_abandoned( pb ) ) {  
    slapi_log_error( SLAPI_LOG_PLUGIN, "my_function", "The operation was  
abandoned.\n" );  
    return 1;  
}
```


5.5. WORKING WITH ENTRIES, ATTRIBUTES, AND VALUES

In certain situations, you will need to pass directory entries between the frontend and the client. For example, if you create a custom **add** function, the frontend passes an entry to your function in the parameter block. When you perform a search operation, you return each matching search entry to the client.

When working with entries, use the [Section 14.22, “Slapi_Entry”](#) data type to get attribute

value pairs. The frontend routines listed in [Table 5.1, “Frontend Functions for Manipulating Entries and Attributes”](#) are designed to help you manipulate entries passed in parameter blocks. These functions are described in more detail in the following sections.

Table 5.1. Frontend Functions for Manipulating Entries and Attributes

Frontend Function	Description
slapi_entry_alloc()	Allocate memory for a new entry.
slapi_entry_init()	Initialize the entry created with slapi_entry_alloc() . <div>  <div> IMPORTANT <p>This must be done before any other operation can occur on the entry. Otherwise, severe memory errors may occur.</p> </div> </div>
slapi_entry_dup()	Copy an entry.
slapi_entry_free()	Free an unused entry from memory.
slapi_entry2str() slapi_entry2str_with_options() slapi_str2entry()	Convert an entry to an LDIF string representation and vice versa.
slapi_entry_get_dn() slapi_entry_set_dn()	Get or set the DN for an entry.
slapi_entry_schema_check()	Verify that an entry complies with the schema.
slapi_entry_first_attr() slapi_entry_next_attr()	Get the attributes of an entry.
slapi_entry_attr_find()	Find the values for a specified attribute.
slapi_entry_attr_merge_sv()	Merge an attribute and its values into an entry.
slapi_entry_add_values_sv()	Add values to an attribute in an entry.

5.5.1. Creating a New Entry

In some situations, you might need to create a new entry. You can create a new entry in two ways:

- By allocating memory for a new entry.

To allocate memory for a new entry, call the `slapi_entry_alloc()` function. This function returns a pointer to a new entry of the opaque datatype `Section 14.22, “Slapi_Entry”`. Then, call the function `slapi_entry_init()` to initialize the entry for use with the other slapi functions. Once you create and initialize a new entry, you should call other frontend routines to set the DN and attributes of the entry.



NOTE

Failing to call `slapi_entry_init()` after `slapi_entry_alloc()` may cause severe memory problems.

- By copying an existing entry.

To make a copy of an existing entry, call the `slapi_entry_dup()` routine. This function returns a pointer to a new entry of the data type **Slapi_Entry** that contains the copied data.

When you are finished using the entry, you should free it from memory by calling the `slapi_entry_free()` function.

5.5.2. Converting Between Entries and Strings

Entries can be stored in LDIF files. When stored in these files, entries are converted into a string representation. The following format is the LDIF string representation for a directory entry:

```
dn: [:]dn\n
[<attr>:[:]<value>\n]
[<attr>:[:]<value>\n]
[<spacecontinuedvalue>\n]*
...
```

To continue the specification of a value on additional lines (that is, if the value wraps to another line), use a single space (the ASCII 32 character) at the beginning of subsequent lines. For example:

```
dn: cn=Jane Doe
   inski, ou=Accounting, dc=example, dc=com
```

Refer to the *Red Hat Directory Server Administration Guide* for details on DN syntax.

If a double-colon is used after a data type, it indicates that the value after the double-colon is encoded as a base-64 string. Data is sometimes encoded as a base-64 string. For example, it might be encoded this way if the value contains a non-printing character or newline.

You can use the following functions to convert between the LDIF string representation of an entry and its **Slapi_Entry** data type:

- To convert an entry from the **Slapi_Entry** data type to its LDIF string representation, call the [slapi_entry2str\(\)](#) and [slapi_entry2str_with_options\(\)](#) functions.

This function returns the LDIF string representation of the entry, or NULL if an error occurs. When the string is no longer required, free it from memory by calling the [slapi_ch_free_string\(\)](#)* function.

- To convert an LDIF string representation back to an entry of the data type **Slapi_Entry**, call the [slapi_str2entry\(\)](#) function.

This function returns an entry of the data type **Slapi_Entry***. If an error occurred during the conversion process, the function returns NULL instead.

When you are finished working with the entry, call the [slapi_entry_free\(\)](#) function.



NOTE

Calling the [slapi_str2entry\(\)](#) function modifies the value of the string argument passed into the function (not a copy). To use the string representation of the entry still, make a copy of the string using [slapi_ch_strdup\(\)](#) before calling this function. Then, call [slapi_ch_free_string\(\)](#) to free the copied string after use.

5.5.3. Miscellaneous Operations

5.5.3.1. Getting and Setting the DN of an Entry

You can call the following two frontend routines to get and set the DN for an entry:

- To get the DN for an entry, call the [slapi_entry_get_dn\(\)](#) function.
- To set the DN for an entry, call the [slapi_entry_set_dn\(\)](#) function.

5.5.3.2. Verifying Compliance with the Schema

Before you add or modify an entry in the database, you may want to verify that the new or changed entry still complies with the database schema.

To verify that an entry complies with the schema, call the [slapi_entry_schema_check\(\)](#) function.

5.5.3.3. Getting the Attributes and Values of an Entry

You can use either of the following methods to obtain the attributes and values of an entry:

- Iterate through the attributes of the entry, testing each one to determine if it is the required attribute.
- Use the [slapi_entry_attr_find\(\)](#) function to determine if an entry has a specific attribute.

After you have found the required attribute, use the [slapi_attr_value_find\(\)](#) function to return the value of that attribute, or use [slapi_attr_value_first\(\)](#) and [slapi_attr_value_next\(\)](#) to iterate through all of the values of a multi-valued attribute.

5.5.3.3.1. Iterating through the Attributes in an Entry

To iterate through the attributes associated with an entry, call the `slapi_entry_first_attr()` function to get the first attribute of the entry. This function returns a pointer to the first attribute in the entry. You can use the pointer to the attribute to determine if it is the attribute that you want.

To retrieve the subsequent parameters in the entry, call `slapi_entry_next_attr()`, and pass it the pointer to the current parameter in the **cookie** parameter of the function. The `slapi_entry_next_attr()` function returns a pointer to the current parameter in the same fashion as the `slapi_entry_first_attr()` function.

After finding the required attribute, it is possible to retrieve its value using `slapi_attr_value_find()` or using `slapi_attr_value_first()` and `slapi_attr_value_next()` to iterate through all of the values of a multi-valued attribute.

5.5.3.3.2. Finding a Specific Attribute in an Entry

To determine if an entry contains a specific attribute, call the `slapi_entry_attr_find()` function. This function returns 0 if the entry contains the attribute, -1 if it does not.

5.5.3.4. Adding and Removing Values

You can call frontend routines to add or remove attributes and values in an entry. The frontend provides the following functionality:

- To add new values to an entry, call the `slapi_entry_add_values_sv()` function.
- To remove values from an entry, call the `slapi_entry_delete_values_sv()` function.
- In certain situations, you may want to add an attribute and its values to an entry while not replacing any attribute values that already exist. To do this, call the `slapi_entry_attr_merge_sv()` function.

There are many convenience functions provided to get and set values of attributes in an entry, beginning with `slapi_entry_attr_get*` and `slapi_entry_attr_set*`, respectively. These functions allow you to get and set specific values in a **Slapi_Entry**. Because they are convenience functions, there are some limitations about their use.

Example 5.1. Get Functions

```
char **slapi_entry_attr_get_chararray(const Slapi_Entry* e, const char
*type);
char *slapi_entry_attr_get_charptr(const Slapi_Entry* e, const char
*type);
int slapi_entry_attr_get_int(const Slapi_Entry* e, const char *type);
unsigned int slapi_entry_attr_get_uint(const Slapi_Entry* e, const char
*type);
long slapi_entry_attr_get_long( const Slapi_Entry* e, const char *type);
unsigned long slapi_entry_attr_get_ulong( const Slapi_Entry* e, const
char *type);
long long slapi_entry_attr_get_longlong( const Slapi_Entry* e, const
char *type);
unsigned long long slapi_entry_attr_get_ulonglong( const Slapi_Entry* e,
```

```
const char *type);
PRBool slapi_entry_attr_get_bool( const Slapi_Entry* e, const char
*type);
```

The *type* parameter is the name of the attribute.

If the attribute does not exist in the entry, the get function return a 0 or NULL value. Some attributes may actually exist and have a 0 value, so to verify that an attribute exists before getting the value, use [slapi_entry_attr_find\(\)](#) first.

These functions return only the first value of an attribute. For multi-valued attributes, use a function to get the **Slapi_Attr***, then iterate through the values using the **slapi_attr_first_value()** and **slapi_attr_next_value()** functions.

Values are usually stored as strings. The functions that return integral values (such as **get_int** and **get_uint**) use **atoi(3)** to get the *int* values, **atol(3)** to get the *long* values, and **strtoll(3)** to get the longlong values. Check the platform documentation for these functions for information about how these functions deal with non-integral values. They usually return a 0 if the value could not be converted to the specified type. To continue these functions, ensure the value is of the correct type.

[slapi_entry_attr_get_charray\(\)](#) returns either NULL or allocated memory that must be freed with [slapi_ch_array_free\(\)](#). The last element of the array is **(char *)NULL**, so you can iterate through the values looking for this as the terminator.

[slapi_entry_attr_get_charptr\(\)](#) returns either NULL or allocated memory that must be freed with [slapi_ch_free_string\(\)](#).

[slapi_entry_attr_get_bool\(\)](#) returns a PR_TRUE if the value is present and is equivalent to **true**, **yes**, or a non-zero numeric value. It returns PR_FALSE if the value is absent, empty, or any of the values **false**, **no**, or a numeric 0 value. String comparisons are case-insensitive, so **TRUE** is the same as **True** or **true**.

Example 5.2. Set Functions

```
void slapi_entry_attr_set_charptr(Slapi_Entry* e, const char *type,
const char *value);
void slapi_entry_attr_set_int( Slapi_Entry* e, const char *type, int l);
void slapi_entry_attr_set_uint( Slapi_Entry* e, const char *type,
unsigned int l);
void slapi_entry_attr_set_long(Slapi_Entry* e, const char *type, long
l);
void slapi_entry_attr_set_ulong(Slapi_Entry* e, const char *type,
unsigned long l);
```

The *type* parameter is the name of the attribute. If the attribute does not exist in the entry, the set function creates it and sets the value. If the attribute exists in the entry, the set function will replace all values with the new specified value. To add values without replacing, use [slapi_entry_add_values_sv\(\)](#).

[slapi_entry_attr_set_charptr\(\)](#) makes a copy of the specified value, so this function is ok to use with a string constant or other non-writable string value. Passing a value of NULL deletes the specified attribute from the entry, just like calling **slapi_entry_attr_delete(Slapi_Entry *, const char *type)**.

There is no **slapi_entry_attr_set_charray()** nor **slapi_entry_attr_set_bool()**

function. For the former function, use a **Slapi_Attr*** or **Slapi_Value*** API function to add the multiple values. For the latter, use `slapi_entry_attr_set_charptr()` or `slapi_entry_attr_set_int()` to set the value to the true or false value.

5.6. WORKING WITH DNS AND RDNS

In certain situations, the frontend passes DN's to the backend through the parameter block. For example, when calling the **add** function, the parameter block includes a parameter that specifies the DN of the new entry to be added.

If you need to manipulate DN's within parameter blocks, you can call the following frontend routines:

Table 5.2. Frontend Functions for Manipulating DN's

Function	Description
<code>slapi_dn_isroot()</code>	Determines if a DN is the root DN (the DN of the privileged superuser).
<code>slapi_sdn_get_parent()</code> <code>slapi_sdn_get_backend_parent()</code>	Gets a copy of the parent DN.
<code>slapi_sdn_issuffix()</code>	Checks if a Slapi_DN structure is the child of another suffix.
<code>slapi_be_issuffix()</code>	Determines if a DN is a suffix served by one of the server's backends.
<code>slapi_sdn_get_ndn()</code>	Normalizes a DN.
<code>slapi_dn_normalize_case()</code>	Normalizes a DN and converts all characters to lowercase.

5.6.1. Determining If a DN Is the Root DN

To determine if a DN is the root DN, call `slapi_dn_isroot()`. This function returns 1 if the specified DN is the root DN of the local database. It returns 0 if the DN is not the root DN.

5.6.2. Working with DN Suffixes

A suffix of a DN identifies a subtree in the directory tree where the DN is located. For example, consider the following DN:

```
cn=Babs Jensen,ou=Product Development,l=US, dc=example,dc=com
```

In this case, one of the suffixes is:

```
l=US, dc=example,dc=com
```

This suffix indicates that the *Babs Jensen* entry is located in the *Example Corporation* subtree in the directory tree.

To determine if a value is a suffix for a DN, call `slapi_sdn_issuffix()`. To determine if a DN is one of the suffixes served by the backend, call the `slapi_be_issuffix()` function.

For more information on suffixes and backend configuration, see the *Administration Guide*.

5.6.3. Getting the Parent DN of a DN

To get a copy of the parent DN for a DN, call either the `slapi_sdn_get_parent()` or the `slapi_sdn_get_backend_parent()` function.

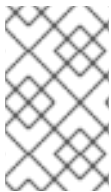
These functions return the parent DN of **dn**. If **dn** is a suffix served by the backend, `slapi_sdn_get_backend_parent()` returns NULL.

When you are finished working with the parent DN, you should free it from memory by calling `slapi_ch_free_string()`.

5.6.4. Normalizing a DN

You can use the following frontend functions to normalize and convert the case of a DN:

- Use `slapi_sdn_get_ndn()` to normalize a DN.
- Use `slapi_dn_normalize_case()` to both normalize the DN and convert all characters in the DN to lowercase.



NOTE

These functions operate on the actual DN specified in the argument, not a copy of the DN. If you want to modify a copy of the DN, call `slapi_ch_strdup()` to make a copy of the DN.

To compare DNs (for example, to search the database for a particular DN), use the `slapi_sdn_compare()` function instead of normalizing and comparing the DNs using string functions.

5.7. WORKING WITH SEARCH FILTERS

When a client requests an LDAP search operation, the frontend passes the search filter to the backend as part of the parameter block. The filter is passed through the **SLAPI_SEARCH_FILTER** parameter. A string representation of the filter is also available in the **SLAPI_SEARCH_STRFILTER** parameter.

To manipulate search filters, call the following frontend routines:

Table 5.3. Frontend Functions for Manipulating Filters

Function	Description
<code>slapi_filter_test()</code>	Determine if an entry matches a filter's criteria.

Function	Description
<code>slapi_filter_get_choice()</code>	Get the filter type.
<code>slapi_filter_get_ava()</code>	Get the attribute type and value used for comparison in an attribute-value assertion filter. (Only applicable to the following searches: LDAP_FILTER_EQUALITY , LDAP_FILTER_GE , LDAP_FILTER_LE , and LDAP_FILTER_APPROX .)
<code>slapi_filter_get_type()</code>	Get the type of attribute that the filter is searching for. (Only applicable to LDAP_FILTER_PRESENT searches.)
<code>slapi_filter_get_subfilt()</code>	Get the substring pattern used for the filter. (Only applicable to LDAP_FILTER_SUBSTRING searches.)
<code>slapi_str2filter()</code>	Convert a string representation of a filter to a filter of the data type Slapi_Filter .
<code>slapi_filter_join()</code>	Construct a new LDAP_FILTER_AND , LDAP_FILTER_OR , or LDAP_FILTER_NOT filter from other filters.
<code>slapi_filter_list_first()</code> <code>slapi_filter_list_next()</code>	Get the components of a filter. (Only applicable to LDAP_FILTER_AND , LDAP_FILTER_OR , and LDAP_FILTER_NOT searches.)
<code>slapi_filter_free()</code>	Free a filter from memory.

5.7.1. Determining If an Entry Matches a Filter

After retrieving a filter from the **SLAPI_SEARCH_FILTER** parameter of the parameter block, you can call the `slapi_filter_test()` function to determine if entries in your database match the filter.

5.7.2. Getting the Filter Type

To determine the type of filter that you are using, call the `slapi_filter_get_choice()` function. This function returns the filter type, which can be any of the following values:

Table 5.4. Types of Filters

Filter Type	Description
LDAP_FILTER_AND	Find entries that match all filters that are specified in this complex filter.

Filter Type	Description
<i>LDAP_FILTER_OR</i>	Find entries that match any filter specified in this complex filter.
<i>LDAP_FILTER_NOT</i>	Find entries that do not match the specified filter.
<i>LDAP_FILTER_EQUALITY</i>	Find entries that contain a value equal to the specified attribute value.
<i>LDAP_FILTER_SUBSTRINGS</i>	Find entries that contain a value that matches the specified substrings.
<i>LDAP_FILTER_GE</i>	Find entries that contain a value greater than or equal to the specified attribute value.
<i>LDAP_FILTER_LE</i>	Find entries that contain a value less than or equal to the specified attribute value.
<i>LDAP_FILTER_PRESENT</i>	Find entries that contain the specified attribute.
<i>LDAP_FILTER_APPROX</i>	Find entries that contain a value approximately matching the specified attribute value.

5.7.3. Getting the Search Criteria

You can use the following functions to retrieve the search criteria specified by a search filter:

Table 5.5. Functions used to Retrieve the Search Criteria Specified by Search Filters

To retrieve the search criteria for this filter type...	Use this function...
<i>LDAP_FILTER_EQUALITY</i> <i>LDAP_FILTER_GE</i> <i>LDAP_FILTER_LE</i> <i>LDAP_FILTER_APPROX</i>	slapi_filter_get_ava()
<i>LDAP_FILTER_PRESENT</i>	slapi_filter_get_type()
<i>LDAP_FILTER_SUBSTRINGS</i>	slapi_filter_get_subfilt()

To retrieve the search criteria for this filter type...	Use this function...
<code>LDAP_FILTER_AND</code> <code>LDAP_FILTER_OR</code> <code>LDAP_FILTER_NOT</code>	slapi_filter_list_first() and slapi_filter_list_next() Both of these functions will return either a filter component of the complex filter or a NULL value, according to the following: <ul style="list-style-type: none"> • If <code>slapi_list_first()</code> returns a NULL, the complex filter is not of the type <code>LDAP_FILTER_AND</code>, <code>LDAP_FILTER_OR</code>, or <code>LDAP_FILTER_NOT</code>. • If <code>slapi_list_next()</code> returns a NULL, the component returned by the call is the last component in the complex filter.

**NOTE**

You do not need to free the values returned by the [slapi_filter_get_ava\(\)](#), [slapi_filter_get_type\(\)](#), and [slapi_filter_get_subfilt\(\)](#) functions.

5.7.4. Converting a String to a Filter

A search filter can be represented by either the data type [Section 14.23, “Slapi_Filter”](#) or as a string. In a parameter block for a search operation, **`SLAPI_SEARCH_FILTER`** is a filter of the data type **`Slapi_Filter`** and **`SLAPI_SEARCH_STRFILTER`** is the string representation of that filter. In general, it is easier to specify a filter as a string than it is to construct a filter from the type **`Slapi_Filter`**.

To convert the string representation of a filter into a filter of the data type **`Slapi_Filter`**, call the [slapi_str2filter\(\)](#) function.

When you have finished working with the filter, you should free it from memory by calling the [slapi_filter_free\(\)](#) function.

5.7.5. Creating Complex Filters by Combining Filters

AND, OR and NOT can combine different filters to create a complex filter. The [slapi_filter_join\(\)](#) function can create these types of filters.

The **`slapi_filter_join()`** function returns the complex filter that you created. When you have finished using the complex filter, you should free it from memory by calling [slapi_filter_free\(\)](#).

Filters of the type **`LDAP_FILTER_NOT`** can have only one component. If the filter type (**`ftype`**) is **`LDAP_FILTER_NOT`**, you must pass a NULL value for the second filter when calling [slapi_filter_join\(\)](#).

5.8. CHECKING PASSWORDS

By default, Directory Server uses the ***userPassword*** attribute to store the credentials for an entry. The server encodes the password using the scheme specified in the ***nsslapd-rootpwstoragescheme*** attribute for the Directory Manager or ***passwordStorageScheme*** attribute for other users. These attributes are defined in the ***cn=config*** entry contained in the ***dse.ldif*** file. The scheme can be any of the following:

- **CLEAR** — No encryption is used, and can be defined using the ***clear-password-storage-scheme*** plug-in.
- **CRYPT** — Uses the Unix crypt algorithm, and can be defined using the ***crypt-password-storage-scheme*** plug-in.
- **SHA, SHA256, SHA384, SHA512** — Uses the Secure Hashing Algorithm, and can be defined using the ***sha-password-storage-scheme*** plug-in. **SHA** is **SHA-1**, which is 140 bits. For the others, the number indicates the number of bits used by the hash.
- **SSHA, SSHA256, SSHA384, SSHA512** — Uses the Salted Secure Hashing Algorithm, and can be defined using the ***ssha-password-storage-scheme*** plug-in. **SSHA** is **SSHA-1**, which is 140 bits, including the salt. For the others, the number indicates the number of bits used by the hash, including the salt.

To determine if a given password is one of the values of the ***userPassword*** attribute, call the [`slapi_pw_find_sv\(\)`](#) function. This function determines which password scheme was used to store the password and uses the appropriate comparison function to compare a given value against the encrypted values of the ***userPassword*** attribute.

CHAPTER 6. WRITING PRE- AND POST-OPERATION PLUG-INS

This chapter explains how to write functions that the Red Hat Directory Server (Directory Server) calls before and after executing an LDAP operation. These functions are called *pre-operation* and *post-operation* plug-in functions.

6.1. HOW PRE- AND POST-OPERATION PLUG-INS WORK

The Directory Server can perform the following LDAP operations: **bind**, **unbind**, **search**, **modify**, **add**, **delete**, **modifyRDN**, **compare**, and **abandon**.



NOTE

The Directory Server can also perform extended operations as defined in the LDAPv3 protocol. For information on implementing plug-in functions to execute extended operations, refer to [Chapter 10, Writing Extended Operation Plug-ins](#).

You can configure the Directory Server to call your custom plug-in functions before and after executing any of these LDAP operations.

For example, you can write a pre-operation function that validates an entry before the server performs an LDAP add operation. An example of a post-operation plug-in function would be one that sends a notification to a user after their entry has been modified by an LDAP modify operation.

The Directory Server can call custom plug-in functions before and after performing operations, such as:

- Sending an LDAP entry to the client.
- Sending an LDAP result code to the client.
- Sending an LDAP referral to the client.

[Figure 6.1, “Calling Pre-operation and Post-operation Plug-in Functions”](#) illustrates how the Directory Server front-end calls pre-operation and post-operation functions before and after executing an LDAP operation.

When processing a request, the Directory Server will call all registered pre-operation functions before it calls the backend to service the request. All pre-operation functions must return before the front-end calls the associated backend function.

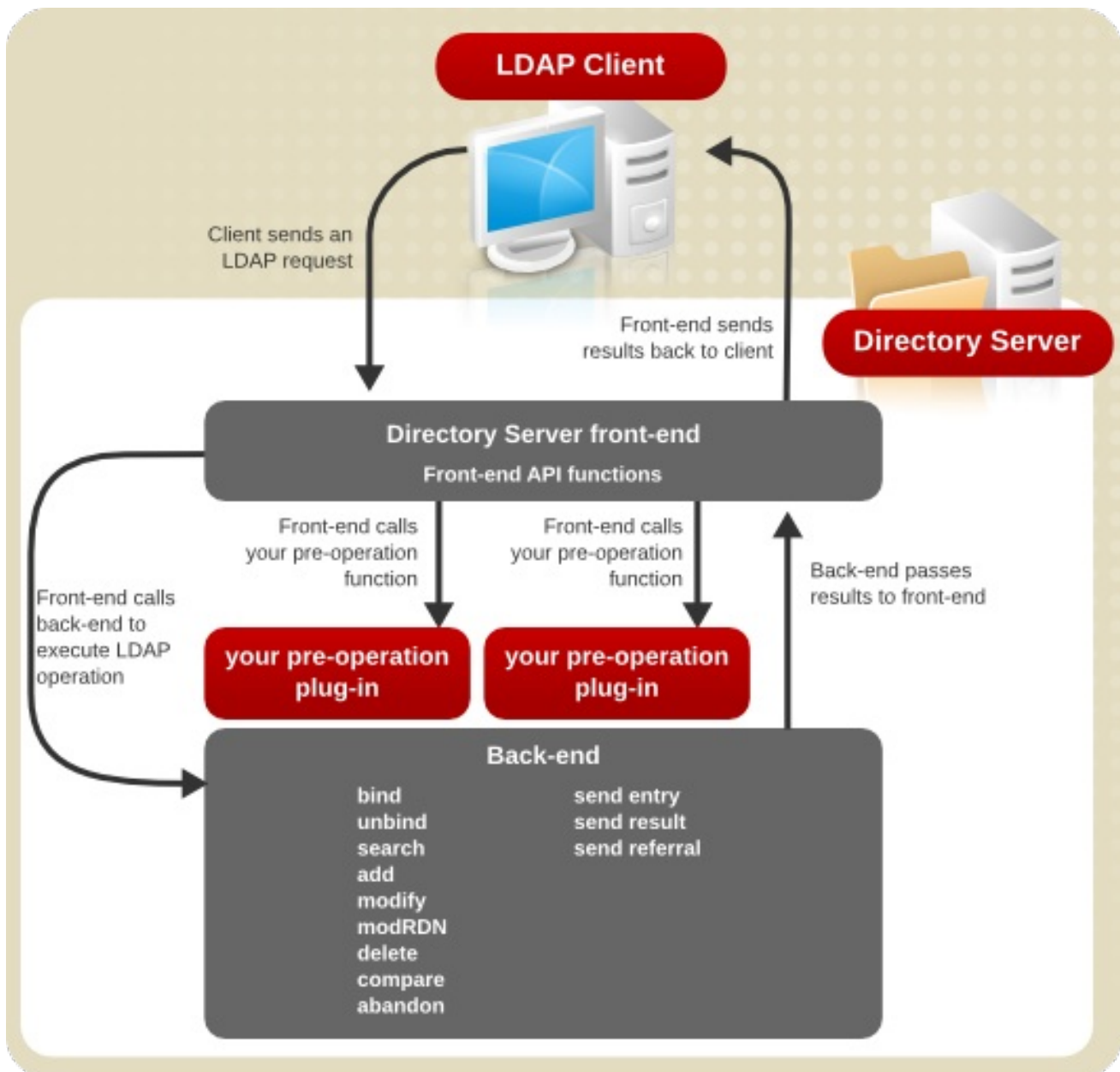


Figure 6.1. Calling Pre-operation and Post-operation Plug-in Functions

6.2. TYPES OF PRE-OPERATION AND POST-OPERATION FUNCTIONS

Pre-operation and post-operation functions are specified in a parameter block that you can set on server startup. Each function corresponds to an ID in the parameter block. In your initialization function, you can call the `slapi_pblock_set()` function to specify the name of your function that corresponds to the pre-operation or post-operation function. For more information on the parameter block, refer to [Section 2.1.3.1, "Getting Data from the Parameter Block"](#).

6.2.1. Types of Pre-operation Functions

The following table lists the Directory Server pre-operation functions and their purpose.

Table 6.1. Functions Called Before the Directory Server Executes an Operation

ID in Parameter Block	Description	Further Information
<i>SLAPI_PLUGIN_PRE_BIND_FN</i>	Specifies the function called before the Directory Server executes an LDAP bind operation.	Section 7.2, “Processing an LDAP Bind Operation” .
<i>SLAPI_PLUGIN_PRE_UNBIND_FN</i>	Specifies the function called before the Directory Server executes an LDAP unbind operation.	Section 7.3, “Processing an LDAP Unbind Operation” .
<i>SLAPI_PLUGIN_PRE_SEARCH_FN</i>	Specifies the function called before the Directory Server executes an LDAP search operation.	Section 7.4, “Processing an LDAP Search Operation” .
<i>SLAPI_PLUGIN_PRE_COMPARE_FN</i>	Specifies the function called before the Directory Server executes an LDAP compare operation.	Section 7.5, “Processing an LDAP Compare Operation” .
<i>SLAPI_PLUGIN_PRE_ADD_FN</i>	Specifies the function called before the Directory Server executes an LDAP add operation.	Section 7.6, “Processing an LDAP Add Operation” .
<i>SLAPI_PLUGIN_PRE_MODIFY_FN</i>	Specifies the function called before the Directory Server executes an LDAP modify operation.	Section 7.7, “Processing an LDAP Modify Operation” .
<i>SLAPI_PLUGIN_PRE_MODRDN_FN</i>	Specifies the function called before the Directory Server executes an LDAP modifyRDN operation.	Section 7.8, “Processing an LDAP Modify RDN Operation” .
<i>SLAPI_PLUGIN_PRE_DELETE_FN</i>	Specifies the function called before the Directory Server executes an LDAP delete operation.	Section 7.9, “Processing an LDAP Delete Operation” .
<i>SLAPI_PLUGIN_PRE_ABANDON_FN</i>	Specifies the function called before the Directory Server executes an LDAP abandon operation.	Section 7.10, “Processing an LDAP Abandon Operation” .

ID in Parameter Block	Description	Further Information
<i>SLAPI_PLUGIN_PRE_ENTRY_FN</i>	Specifies the function called before the Directory Server sends an entry to the client (for example, when you call slapi_send_ldap_search_entry() , the pre-operation entry function is called before the entry is sent to the client).	
<i>SLAPI_PLUGIN_PRE_REFERRAL_FN</i>	Specifies the function called before the Directory Server sends a referral to the client (for example, when you call slapi_send_ldap_referral() , the pre-operation referral function is called before the referral is sent to the client).	
<i>SLAPI_PLUGIN_PRE_RESULT_FN</i>	Specifies the function called before the Directory Server sends a result code to the client (for example, when you call slapi_send_ldap_result() , the pre-operation result function is called before the result code is sent to the client).	

6.2.2. Types of Post-operation Functions

The following table lists the Directory Server post-operation functions and their purpose.

Table 6.2. Functions Called After the Directory Server Executes an Operation

ID in Parameter Block	Description	Further Information
<i>SLAPI_PLUGIN_POST_BIND_FN</i>	Specifies the function called after the Directory Server executes an LDAP bind operation.	Section 7.2, “Processing an LDAP Bind Operation” .
<i>SLAPI_PLUGIN_POST_UNBIND_FN</i>	Specifies the function called after the Directory Server executes an LDAP unbind operation.	Section 7.3, “Processing an LDAP Unbind Operation” .
<i>SLAPI_PLUGIN_POST_SEARCH_FN</i>	Specifies the function called after the Directory Server executes an LDAP search operation.	Section 7.4, “Processing an LDAP Search Operation” .

ID in Parameter Block	Description	Further Information
<i>SLAPI_PLUGIN_POST_COMPARE_FN</i>	Specifies the function called after the Directory Server executes an LDAP compare operation.	Section 7.5, “Processing an LDAP Compare Operation” .
<i>SLAPI_PLUGIN_POST_ADD_FN</i>	Specifies the function called after the Directory Server executes an LDAP add operation.	Section 7.6, “Processing an LDAP Add Operation” .
<i>SLAPI_PLUGIN_POST_MODIFY_FN</i>	Specifies the function called after the Directory Server executes an LDAP modify operation.	Section 7.7, “Processing an LDAP Modify Operation” .
<i>SLAPI_PLUGIN_POST_MODIFY_RDN_FN</i>	Specifies the function called after the Directory Server executes an LDAP modify RDN operation.	Section 7.8, “Processing an LDAP Modify RDN Operation” .
<i>SLAPI_PLUGIN_POST_DELETE_FN</i>	Specifies the function called after the Directory Server executes an LDAP delete operation.	Section 7.9, “Processing an LDAP Delete Operation” .
<i>SLAPI_PLUGIN_POST_ABANDON_FN</i>	Specifies the function called after the Directory Server executes an LDAP abandon operation.	Section 7.10, “Processing an LDAP Abandon Operation” .
<i>SLAPI_PLUGIN_POST_ENTRY_FN</i>	Specifies the function called after the Directory Server sends an entry to the client (for example, when you call slapi_send_ldap_search_entry() , the post-operation entry function is called after the entry is sent to the client).	

ID in Parameter Block	Description	Further Information
<i>SLAPI_PLUGIN_POST_REFERRAL_FN</i>	Specifies the function called after the Directory Server sends a referral to the client (for example, when you call slapi_send_ldap_referral() , the post-operation referral function is called after the referral is sent to the client).	
<i>SLAPI_PLUGIN_POST_RESULT_FN</i>	Specifies the function called after the Directory Server sends a result code to the client (for example, when you call slapi_send_ldap_result() , the post-operation result function is called after the result code is sent to the client).	

6.3. USING PLUG-IN CONFIGURATION INFORMATION IN PREOP PLUG-INS

A plug-in may need to access its configuration data to use as part of the initialization function. For example, a plug-in may specify a specific port to use to connect to the Directory Server for some operation, and that port number is set in the plug-in's configuration entry.

However, since the plug-in is not initialized yet, there is no **SLAPI_TARGET_DN** set yet. The plug-in can retrieve its own configuration using the **SLAPI_PLUGIN_CONFIG_ENTRY** pblock parameter.

```
int
my_plugin_init(Slapi_PBlock *pb)
{
    Slapi_Entry *my_config_entry = NULL;
    slapi_pblock_get(pb, SLAPI_PLUGIN_CONFIG_ENTRY, &my_config_entry);
}
```

The plug-in must not free the **Slapi_Entry*** returned with **SLAPI_PLUGIN_CONFIG_ENTRY**, and any changes made to this entry may be lost or ignored. The **Slapi_Entry*** structure should be read-only and used only to get plug-in configuration parameters as attributes of the entry.

Use the internal **slapi_modify_internal*** SLAPI functions to modify the returned entry.

6.4. REGISTERING PRE- AND POST-OPERATION FUNCTIONS

To register your pre-operation and post-operation plug-in functions, you need to write an initialization function and then configure the server to load your plug-in. For details, follow the procedures outlined in [Section 2.2, “Writing Plug-in Initialization Functions”](#), and [Chapter 3, Configuring Plug-ins](#).

CHAPTER 7. DEFINING FUNCTIONS FOR LDAP OPERATIONS

This chapter explains how to write pre-operation and post-operation functions for specific LDAP operations. In general, the functions outlined here use a parameter block to pass information between the plug-in and the Red Hat Directory Server. Because of this, these plug-in functions will pass a single argument, a parameter block defined by the data type [Slapi_PBlock](#). Refer to [Section 2.1.2, “Passing Data with Parameter Blocks”](#) for more information.

7.1. SPECIFYING START AND CLOSE FUNCTIONS

For each pre-operation and post-operation plug-in, you can specify a function to be called after the server starts and before the server is shut down.

For example, the plug-in may need some context information to pass to all of the plug-in callback functions. Then, create the context in the start function and store it in the **SLAPI_PLUGIN_PRIVATE** slot to use throughout the plug-in. Any resources (**free()**) and any memory allocated in the start function must be released in the close function.

Another example is if the plug-in needs to communicate with an external database. The connection is usually opened in the start function, and the connection must be closed in the close function.

In the plug-in init function, use the following parameters with [slapi_pblock_set\(\)](#) to specify these functions:

SLAPI_PLUGIN_START_FN	Specifies the function called after the Directory Server starts.
SLAPI_PLUGIN_CLOSE_FN	Specifies the function called before the Directory Server shuts down.

7.2. PROCESSING AN LDAP BIND OPERATION

When the Directory Server receives an LDAP bind request from a client, the frontend determines the DN the client is attempting to bind and the authentication method being used. The frontend also gets the credentials used for authentication and, if SASL is used for authentication, the SASL mechanism used.

7.2.1. Defining Functions for the Bind Operation

In the parameter block, the following parameters specify plug-in functions that are called in the process of executing a bind operation:

- The **SLAPI_PLUGIN_PRE_BIND_FN** parameter specifies the pre-operation bind function.
- The **SLAPI_PLUGIN_POST_BIND_FN** parameter specifies the post-operation bind function.

To register the plug-in functions, call [slapi_pblock_set\(\)](#) to set these parameters in your initialization function. Refer to [Section 2.2.3, “Registering Your Plug-in Functions”](#).

Your pre-operation and post-operation bind functions should return **0** if successful. If the pre-operation function returns a non-zero value, the post-operation bind function is never called.

For information on defining a function that handles authentication, refer to [Chapter 8, *Defining Functions for Authentication*](#).

7.2.2. Getting and Setting Parameters for the Bind Operation

The frontend makes this information available to pre-operation and post-operation plug-in functions in the form of parameters in a parameter block.

Table 7.1. Parameters for the Bind Operation

Parameter ID	Data Type	Description
<i>SLAPI_BIND_TARGET</i>	char *	DN of the entry as which to bind.
<i>SLAPI_BIND_METHOD</i>	int	Authentication method used; for example, <i>LDAP_AUTH_SIMPLE</i> or <i>LDAP_AUTH_SASL</i> .
<i>SLAPI_BIND_CREDENTIALS</i>	struct berval *	Credentials from the bind request.
<i>SLAPI_BIND_RET_SASLCREDENTIALS</i>	struct berval *	The credentials that you want sent to the client. Set this before calling slapi_send_ldap_result() .
<i>SLAPI_BIND_SASLMECHANISM</i>	char *	SASL mechanism used; for example, <i>LDAP_SASL_EXTERNAL</i> .

If the ***SLAPI_BIND_SASLMECHANISM*** parameter is empty, simple authentication was used, and simple credentials were provided.

7.3. PROCESSING AN LDAP UNBIND OPERATION

When the Directory Server receives an LDAP unbind request from a client, the frontend calls the `nunbind` function for each backend. No operation-specific parameters are placed in the parameter block that is passed to the unbind function.

In the parameter block, the following parameters specify plug-in functions that are called in the process of executing an **unbind** operation:

- The ***SLAPI_PLUGIN_PRE_UNBIND_FN*** parameter specifies the pre-operation **unbind** function.
- The ***SLAPI_PLUGIN_POST_UNBIND_FN*** parameter specifies the post-operation **unbind** function.

Call the `slapi_pblock_set()` function to set these parameters to the names of your functions.

Your plug-in functions should return **0** if successful. If the pre-operation function returns a non-zero value, the post-operation **unbind** function is never called.

7.4. PROCESSING AN LDAP SEARCH OPERATION

The server processes an LDAP search operation in two stages:

1. First, the server gets a list of candidate entries, using an index (if applicable).

For example, for a search filter that finds entries where **mail=a***, the server checks the index for the **mail** attribute (if the index exists), finds the keys that start with **a**, and generates a list of matching entries.

If no applicable index exists, all entries are considered to be candidates.

To get the list of candidates, the server calls the backend search function. For details, refer to [Section 7.4.1, “Getting the List of Candidates”](#).

2. Next, the server iterates through each candidate in the list and determines if the candidate matches the search criteria.

If an entry matches the criteria, the server sends the entry to the client.

To check each candidate, the server calls the backend **next_candidate** function for each candidate in the list. For details, refer to [Section 7.4.2, “Iterating through Candidates”](#).

The rest of this section explains these stages in more detail.

7.4.1. Getting the List of Candidates

When the Directory Server receives an LDAP search request, the frontend gets information about the search (such as the scope and base DN). The frontend normalizes the base DN by calling the `slapi_sdn_get_ndn()` function and determines if the base DN identifies a *DSA-specific entry (DSE)*. If so, the frontend handles the search request directly and does not pass it to the backend search function.

If the base DN is not a DSE, the frontend finds the backend that services the suffix specified in the base DN. The frontend then passes the search criteria to the **search** function for that backend.

The frontend makes this information available to pre-operation and post-operation plug-in functions in the form of parameters in a parameter block.

Table 7.2. Table of Information Available during an LDAP Search Operation

Parameter ID	Data Type	Description
<i>SLAPI_SEARCH_TARGET</i>	char *	DN of the base entry in the search operation (the starting point of the search).

Parameter ID	Data Type	Description
<i>SLAPI_ORIGINAL_TARGET_DN</i>	char *	The original DN sent by the client (this DN is normalized by <i>SLAPI_SEARCH_TARGET</i>); read-only parameter.
<i>SLAPI_SEARCH_SCOPE</i>	int	The scope of the search. The scope can be one of the following values: <ul style="list-style-type: none"> • <i>LDAP_SCOPE_BASE</i> • <i>LDAP_SCOPE_ONELEVEL</i> • <i>LDAP_SCOPE_SUBTREE</i>
<i>SLAPI_SEARCH_DEREF</i>	int	Method for handling aliases in a search. This method can be one of the following values: <ul style="list-style-type: none"> • <i>LDAP_DEREF_NEVER</i> • <i>LDAP_DEREF_SEARCHING</i> • <i>LDAP_DEREF_FINDING</i> • <i>LDAP_DEREF_ALWAYS</i>
<i>SLAPI_SEARCH_SIZELIMIT</i>	int	Maximum number of entries to return in the search results.
<i>SLAPI_SEARCH_TIMELIMIT</i>	int	Maximum amount of time (in seconds) allowed for the search operation.
<i>SLAPI_SEARCH_FILTER</i>	Slapi_Filter *	Slapi_Filter struct (an opaque data structure) representing the filter to be used in the search.
<i>SLAPI_SEARCH_STRFILTER</i>	char *	String representation of the filter to be used in the search.
<i>SLAPI_SEARCH_ATTRS</i>	char **	Array of attribute types to be returned in the search results.

Parameter ID	Data Type	Description
<i>SLAPI_SEARCH_ATTRONLY</i>	int	Specifies whether the search results return attribute types only or attribute types and values. (0 means return both attributes and values; 1 means return attribute types only).

Your search function should return **0** if successful. Call the `slapi_pblock_set()` function to assign the set of search results to the ***SLAPI_SEARCH_RESULT_SET*** parameter in the parameter block.

The frontend then uses this function in conjunction with the **next entry** function (refer to [Section 7.4.2, “Iterating through Candidates”](#)) to iterate through the result set. The frontend sends each result back to the client and continues updates the ***SLAPI_NENTRIES*** parameter with the current number of entries sent back to the client.

If a result is actually a referral, the frontend sends the referral back to the client and updates the ***SLAPI_SEARCH_REFERRALS*** parameter with the list of referral URLs.

Finally, after sending the last entry to the client, the frontend sends an LDAP result message specifying the number of entries found.

7.4.2. Iterating through Candidates

In addition to the parameters specified in [Section 7.4, “Processing an LDAP Search Operation”](#), the **next entry** function has access to the following parameters (which are set by the frontend and the backend during the course of executing a search operation):

Table 7.3. Table of Information Available to the next entry Function

Parameter ID	Data Type	Description
<i>SLAPI_SEARCH_RESULT_SET</i>	void *	Set of search results. These data are specific to the backend processing the search. Any plug-in functions should not touch this value.
<i>SLAPI_SEARCH_RESULT_ENTRY</i>	Slapi_Entry *	Entry returned from iterating through the results set. This next entry function actually sets this parameter.
<i>SLAPI_SEARCH_RESULT_ENTRY_EXT</i>	void *	Reserved for future use. The context identifying the last result sent in the results set. This next entry function actually sets this parameter.

Parameter ID	Data Type	Description
<i>SLAPI_NENTRIES</i>	int	Number of search results found.
<i>SLAPI_SEARCH_REFERRALS</i>	struct berval **	NULL-terminated array of the URLs to other LDAP servers to which the current server is referring the client.

The **next entry** function should get the next result specified in the set of results in the ***SLAPI_SEARCH_RESULT_SET*** parameter. The function should set this next entry as the value of the ***SLAPI_SEARCH_RESULT_ENTRY*** parameter in the parameter block, and then **next entry** function should return **0** if successful.

The **next entry** function should set the ***SLAPI_SEARCH_RESULT_ENTRY*** parameter to NULL and return **-1** if one of the following situations occurs:

- The operation is abandoned (you can check this by calling the [slapi_op_abandoned\(\)](#) function).
- The time limit has been exceeded.
- The maximum number of entries has been exceeded.

If no more entries exist in the set of results, the **next entry** function should set the ***SLAPI_SEARCH_RESULT_ENTRY*** parameter to NULL and return **0**.

7.5. PROCESSING AN LDAP COMPARE OPERATION

When the Directory Server receives an LDAP compare request from a client, the frontend gets the DN of the entry being compared and the attribute and value being used in the comparison.

The frontend makes this information available to pre-operation and post-operation plug-in functions in the form of parameters in a parameter block.

Table 7.4. Table of Information Returned during an LDAP Compare Operation

Parameter ID	Data Type	Description
<i>SLAPI_COMPARE_TARGET</i>	char *	DN of the entry to be compared.
<i>SLAPI_COMPARE_TYPE</i>	char *	Attribute type to use in the comparison.
<i>SLAPI_COMPARE_VALUE</i>	struct berval *	Attribute value to use in the comparison.

The compare function should call `slapi_send_ldap_result()` to send **LDAP_COMPARE_TRUE** if the specified value is equal to the value of the entry's attribute or **LDAP_COMPARE_FALSE** if the values are not equal.

If successful, the compare function should return **0**. If an error occurs (for example, if the specified attribute doesn't exist), the compare function should call `slapi_send_ldap_result()` to send an LDAP error code and should return **1**.

7.6. PROCESSING AN LDAP ADD OPERATION

When the Directory Server receives an LDAP add request from a client, the frontend normalizes the DN of the new entry. The frontend makes this information available to pre-operation and post-operation plug-in functions in the form of parameters in a parameter block.

Table 7.5. Table of Information Processed during an LDAP Add Operation

Parameter ID	Data Type	Description
SLAPI_ADD_TARGET	char *	DN of the entry to be added.
SLAPI_ADD_ENTRY	Slapi_Entry *	The entry to be added (specified as the opaque Slapi_Entry data type).

The add function should check if the operation has been abandoned, and, if it has, the function should return **-1**.



NOTE

It is not necessary to call `slapi_send_ldap_result()` to send an LDAP error code to the client. According to the LDAP protocol, the client does not expect a server response after an operation is abandoned.

These optional checks are not required. The plug-in can pass the operation to the regular frontend and backend processing which handle these cases.

- If the entry already exists in the database, the function should call `slapi_send_ldap_result()` to send an LDAP error code **LDAP_ALREADY_EXISTS** and should return **-1**.
- If the parent entry, or the closest matching entry, is a referral entry (that is, an entry with the object class **ref**) and no **manageDSAIT** control is included with the request, the function should call `slapi_send_ldap_referral()` to send a referral and return **-1**.

To determine if a **manageDSAIT** control is present, call `slapi_pblock_get()` to get the value of the **SLAPI_MANAGEDSAIT** parameter. If the value is **1**, the control is included in the request. If the value is **0**, the control is not included in the request.

- If the parent entry does not exist, the function should call `slapi_send_ldap_result()` to send an LDAP error code **LDAP_NO_SUCH_OBJECT** and return **-1**.

- If the entry is not schema-compliant (call [slapi_entry_schema_check\(\)](#) to determine this), the function should call **slapi_send_ldap_result()** to send the LDAP error code `LDAP_OBJECT_CLASS_VIOLATION` and should return **-1**.
- If the requestor does not have permission to add the entry (call [slapi_access_allowed\(\)](#) to determine this), the function should call **slapi_send_ldap_result()** to send the LDAP error code `LDAP_INSUFFICIENT_ACCESS` and should return **-1**.

You should also verify that the ACI syntax for the entry is correct; call [slapi_acl_check_mods\(\)](#) to determine this.

If the **add** function is successful, the function should call **slapi_send_ldap_result()** to send an **LDAP_SUCCESS** code back to the client and should return **0**.

7.7. PROCESSING AN LDAP MODIFY OPERATION

When the Directory Server receives an LDAP modify request from a client, the frontend gets the DN of the entry to be modified and the modifications to be made. The frontend makes this information available to pre-operation and post-operation plug-in functions in the form of parameters in a parameter block.

Table 7.6. Table of Information Processed during an LDAP Modify Operation

Parameter ID	Data Type	Description
SLAPI_MODIFY_TARGET	char *	DN of the entry to be modified.
SLAPI_MODIFY_MODS	LDAPMod **	A NULL-terminated array of LDAPMod structures, which represent the modifications to be performed on the entry.

The **modify** function should check the following:

- If the operation has been abandoned, the function should return **-1**.



NOTE

You do not need to call [slapi_send_ldap_result\(\)](#) to send an LDAP error code to the client. According to the LDAP protocol, the client does not expect a server response after an operation is abandoned.

- If the entry is a referral entry (that is, an entry with the object class **ref**) and no **manageDSAIT** control is included with the request, the function should call [slapi_send_ldap_referral\(\)](#) to send a referral and return **-1**.

To determine if a **manageDSAIT** control is present, call [slapi_pblock_get\(\)](#) to get the value of the **SLAPI_MANAGEDSAIT** parameter. If the value is **1**, the control is included in the request. If the value is **0**, the control is not included in the request.

- If the entry does not exist, check the following:

- If the closest matching entry is a referral entry, and if no *manageDSAIT* control is included in the request, the function should call `slapi_send_ldap_referral()` to send a referral and return **-1**.
- Otherwise, the function should call `slapi_send_ldap_result()` to send an LDAP error code `LDAP_NO_SUCH_OBJECT` and return **-1**.
- If the entry is not schema-compliant (call `slapi_entry_schema_check()` to determine this), the function should call `slapi_send_ldap_result()` to send the LDAP error code `LDAP_OBJECT_CLASS_VIOLATION` and should return **-1**.
- If the RDN of the entry contains attribute values that are not part of the entry (for example, if the RDN is **uid=bjensen**, but the entry has no **uid** value or has a different **uid** value), the function should call `slapi_send_ldap_result()` to send the LDAP error code `LDAP_NOT_ALLOWED_ON_RDN` and should return **-1**.
- If the requester does not have permission to modify the entry (call `slapi_access_allowed()` to determine this), the function should call `slapi_send_ldap_result()` to send the LDAP error code `LDAP_INSUFFICIENT_ACCESS` and should return **-1**.

You should also verify that the ACL syntax for the entry is correct; call `slapi_acl_check_mods()` to determine this.

If the **modify** function is successful, the function should call `slapi_send_ldap_result()` to send an `LDAP_SUCCESS` code back to the client and should return **0**.

7.8. PROCESSING AN LDAP MODIFY RDN OPERATION

When the Directory Server receives an LDAP **modifyRDN** request from a client, the frontend gets the original DN of the entry, the new RDN, and, if the entry is moving to a different location in the directory tree, the DN of the new parent of the entry.

The frontend makes this information available to pre-operation and post-operation plug-in functions in the form of parameters in a parameter block.

Table 7.7. Table of Information Processed during an LDAP ModifyRDN Operation

Parameter ID	Data Type	Description
<i>SLAPI_MODRDN_TARGET</i>	char *	DN of the entry that you want to rename.
<i>SLAPI_MODRDN_NEWRDN</i>	char *	New RDN to assign to the entry.
<i>SLAPI_MODRDN_DELOLDRDN</i>	int	Specifies whether to delete the old RDN. <ul style="list-style-type: none"> • 0 - Do not delete the old RDN. • 1 - Delete the old RDN

Parameter ID	Data Type	Description
<i>SLAPI_MODRDN_NEWSUPERIOR</i>	char *	DN of the new parent of the entry, if the entry is being moved to a new location in the directory tree.

The modify RDN function should check the following:

- If the operation has been abandoned, the function should return **-1**.



NOTE

You do not need to call `slapi_send_ldap_result()` to send an LDAP error code to the client. According to the LDAP protocol, the client does not expect a server response after an operation is abandoned.

- If the entry is a referral entry (that is, an entry with the object class **ref**) and no ***manageDSAIT*** control is included with the request, the function should call `slapi_send_ldap_referral()` to send a referral and return **-1**.

To determine if a ***manageDSAIT*** control is present, call `slapi_pblock_get()` to get the value of the ***SLAPI_MANAGEDSAIT*** parameter. If the value is **1**, the control is included in the request. If the value is **0**, the control is not included in the request.

- If the entry does not exist, check the following:
 - If the closest matching entry is a referral entry, and if no ***manageDSAIT*** control is included in the request, the function should call `slapi_send_ldap_referral()` to send a referral and return **-1**.
 - Otherwise, the function should call `slapi_send_ldap_result()` to send an LDAP error code `LDAP_NO_SUCH_OBJECT` and return **-1**.
- If the entry is not schema-compliant (call `slapi_entry_schema_check()` to determine this), the function should call `slapi_send_ldap_result()` to send the LDAP error code `LDAP_OBJECT_CLASS_VIOLATION` and should return **-1**.
- If the RDN of the entry contains attribute values that are not part of the entry (for example, if the RDN is **uid=bjensen**, but the entry has no **uid** value or has a different **uid** value), the function should call `slapi_send_ldap_result()` to send the LDAP error code `LDAP_NOT_ALLOWED_ON_RDN` and should return **-1**.
- If the requester does not have permission to modify the entry (call `slapi_access_allowed()` to determine this), the function should call `slapi_send_ldap_result()` to send the LDAP error code `LDAP_INSUFFICIENT_ACCESS` and should return **-1**.

You should also verify that the ACI syntax for the entry is correct; call `slapi_acl_check_mods()` to determine this.

If the **modifyRDN** function is successful, the function should call `slapi_send_ldap_result()` to send an `LDAP_SUCCESS` code back to the client and should return **0**.

7.9. PROCESSING AN LDAP DELETE OPERATION

When the Directory Server receives an LDAP delete request from a client, the frontend gets the DN of the entry to be removed from the directory. The frontend makes this information available to pre-operation and post-operation plug-in functions in the form of parameters in a parameter block.

Table 7.8. Table of Information Processed during an LDAP Delete Operation

Parameter ID	Data Type	Description
<i>SLAPI_DELETE_TARGET</i>	char *	DN of the entry to delete.

If the delete function is successful, it should return **0**.

7.10. PROCESSING AN LDAP ABANDON OPERATION

When the Directory Server receives an LDAP abandon request from a client, the frontend gets the message ID of the operation that should be abandoned. The frontend makes this information available to pre-operation and post-operation plug-in functions in the form of parameters in a parameter block.

Table 7.9. Table of Information Processed during an LDAP Abandon Operation

Parameter ID	Data Type	Description
<i>SLAPI_ABANDON_MSGID</i>	unsigned long	Message ID of the operation to abandon.

CHAPTER 8. DEFINING FUNCTIONS FOR AUTHENTICATION

This chapter explains how to write a plug-in function to bypass or replace the standard function for authentication with your own function.

8.1. UNDERSTANDING AUTHENTICATION METHODS

Authentication methods for LDAP is described in RFC 4513, available at <http://www.ietf.org/rfc/rfc4513.txt>.

Two methods that you can use to authenticate clients are simple authentication and SASL authentication:

- Simple authentication is described in RFC 4513, available at <http://www.ietf.org/rfc/rfc4513.txt>.

Simple authentication provides minimal facilities for authentication. In the simple authentication method, clients send a DN and password to the server for authentication. The server compares the password sent by the client against the password stored in the client's directory entry.

- Simple Authentication and Security Layer (SASL) is described in RFC 4422, which you can find at <http://www.ietf.org/rfc/rfc4422.txt>.

SASL provides the means to use mechanisms other than simple authentication and TLS to authenticate to the Directory Server.

8.2. HOW THE DIRECTORY SERVER IDENTIFIES CLIENTS

The server keeps track of the identity of the LDAP client through the **SLAPI_CONN_DN** and **SLAPI_CONN_AUTHTYPE** parameters.

During an LDAP **bind** operation, the server authenticates the user and puts the DN and authenticated method in the **SLAPI_CONN_DN** and **SLAPI_CONN_AUTHTYPE** parameters.

When an authenticated client requests the server to perform an LDAP operation, the server checks the DN in the **SLAPI_CONN_DN** parameter to determine if the client has the appropriate access rights.

8.3. HOW THE AUTHENTICATION PROCESS WORKS

When the Directory Server receives an LDAP **bind** request from a client, it processes the request as follows:

Procedure 8.1. How an Authentication Request is Processed

1. The server parses the LDAP **bind** request and retrieves the following information:
 - The DN as which the client is attempting to authenticate.
 - The method of authentication used.
 - Any credentials (such as a password) included in the request.

If the method of authentication is **LDAP_AUTH_SASL** (SASL authentication), the server also retrieves the name of the SASL mechanism used from the LDAP **bind** request.

2. The server normalizes the DN retrieved from the request. (Refer to the [slapi_sdn_get_ndn\(\)](#) function for more information on normalized DN's.)
3. The server retrieves any LDAPv3 controls included with the LDAP **bind** request.
4. If the method of authentication is **LDAP_AUTH_SASL** (SASL authentication), the server determines whether the SASL mechanism (specified in the request) is supported.

If the SASL mechanism is not supported by the server, the server sends an **LDAP_AUTH_METHOD_NOT_SUPPORTED** result code back to the client and ends the processing of the **bind** request.

5. If the method of authentication is **LDAP_AUTH_SIMPLE** (simple authentication), the server checks if the DN is an empty string or if there are no credentials.

If the DN is an empty string, if the DN is not specified, or if no credentials are specified, the server assumes that the client is binding anonymously and sends an **LDAP_SUCCESS** result code back to the client.

The DN and authentication method for the connection, which are used to determine access rights for all operations performed through the connection, are left as **NULL** and **SLAPD_AUTH_NONE**, respectively.

6. If the DN specified in the request is not served by this Directory Server (for example, if the DN is **uid=moxcross,dc=example,dc=com**, and the directory root of the server is **dc=example,dc=com**), the server sends one of the following two results back to the client and ends the processing of the **bind** request:
 - If the server is configured with a default referral (that is, an LDAP URL which identifies an LDAP server that handles referrals), the server sends an **LDAP_REFERRAL** result code back to the client, or **LDAP_PARTIAL_RESULTS** if the client only supports the LDAPv2 protocol.
 - If the server is not configured with a default referral, the server sends an **LDAP_NO_SUCH_OBJECT** result code back to the client.
7. The server puts the information from the **bind** request into the parameter block:
 - **SLAPI_BIND_TARGET** is set to the DN as which the client is authenticating.
 - **SLAPI_BIND_METHOD** is set to the authentication method (for example, **LDAP_AUTH_SIMPLE** or **LDAP_AUTH_SASL**).
 - **SLAPI_BIND_CREDENTIALS** is set to the credentials (for example, the password) included in the request.
 - **SLAPI_BIND_SASLMECHANISM** (if the authentication method is **LDAP_AUTH_SASL**) is set to the name of the SASL mechanism that the client is using for authentication.
8. If the DN is the root DN or the update DN (the DN of the master entity responsible for replicating the directory), the server authenticates the client.
 - If the credentials are correct, the server sets the **SLAPI_CONN_DN** parameter to the DN and the **SLAPI_CONN_AUTHTYPE** parameter to **LDAP_AUTH_SIMPLE**. The

server sends an LDAP_SUCCESS result code back to the client and ends the processing of the **bind** request.

- If the credentials are incorrect, the server sends an LDAP_INVALID_CREDENTIALS result code back to the client and ends the processing of the **bind** request.
9. At this point, the server calls any pre-operation **bind** plug-in functions. If the function returns a non-zero value, the server ends the processing of the **bind** request.

If you are writing your own plug-in function to handle authentication, you should return a non-zero value so that the server does not attempt to continue processing the **bind** request.

10. The server calls the backend **bind** function. The **bind** function returns one of the following values:
- If the function returns a non-zero value, the server ends the processing of the **bind** request. The **bind** function is responsible for sending the appropriate result code back to the client before returning a non-zero value.
 - If the function returns 0, the server continues processing the **bind** request. The server sends the LDAP_SUCCESS result code back to the client. (The **bind** function does not do this.)
11. If the backend **bind** function succeeds, the server sets the **SLAPI_CONN_DN** parameter to the DN, and the **SLAPI_CONN_AUTHTYPE** parameter to the authentication method.
12. The server sends an LDAP_SUCCESS result code back to the client and ends the processing of the **bind** request.

If the client's password is due to expire, the server includes a *password expiring* control (with the OID 2.16.840.1.113730.3.4.5) as part of the result sent to the client. If the client is logging in for the first time and needs to change the password, the server includes a *password expired* control (with the OID 2.16.840.1.113730.3.4.4) as part of the result sent to the client.

8.4. WRITING YOUR OWN AUTHENTICATION PLUG-IN

The situation may arise where you want to write and implement your own authentication function; that is, replace the standard means of authentication with your own function. You can write a pre-operation **bind** plug-in function (a function that the server calls before processing an LDAP **bind** request) that performs the authentication and bypasses the default **bind** functionality. This is described in the following section.

8.5. WRITING A PRE-OPERATION BIND PLUG-IN

You can define your own pre-operation **bind** plug-in function to authenticate LDAP clients. The server will call your function during the authentication process. See [Procedure 8.1, “How an Authentication Request is Processed”](#) for more information on the authentication process. Your function should return a non-zero value to bypass the default backend **bind** function and the post-operation **bind** functions.

This means that the final steps of the authentication process are skipped. Your pre-operation plug-in function is responsible for sending the result code to the client and for setting the DN and authentication method for the connection.

Figure 8.1, “Using a Pre-Operation **bind** Plug-in Function to Handle Authentication” summarizes the process of using a pre-operation **bind** plug-in function to authenticate LDAP clients to the Directory Server.

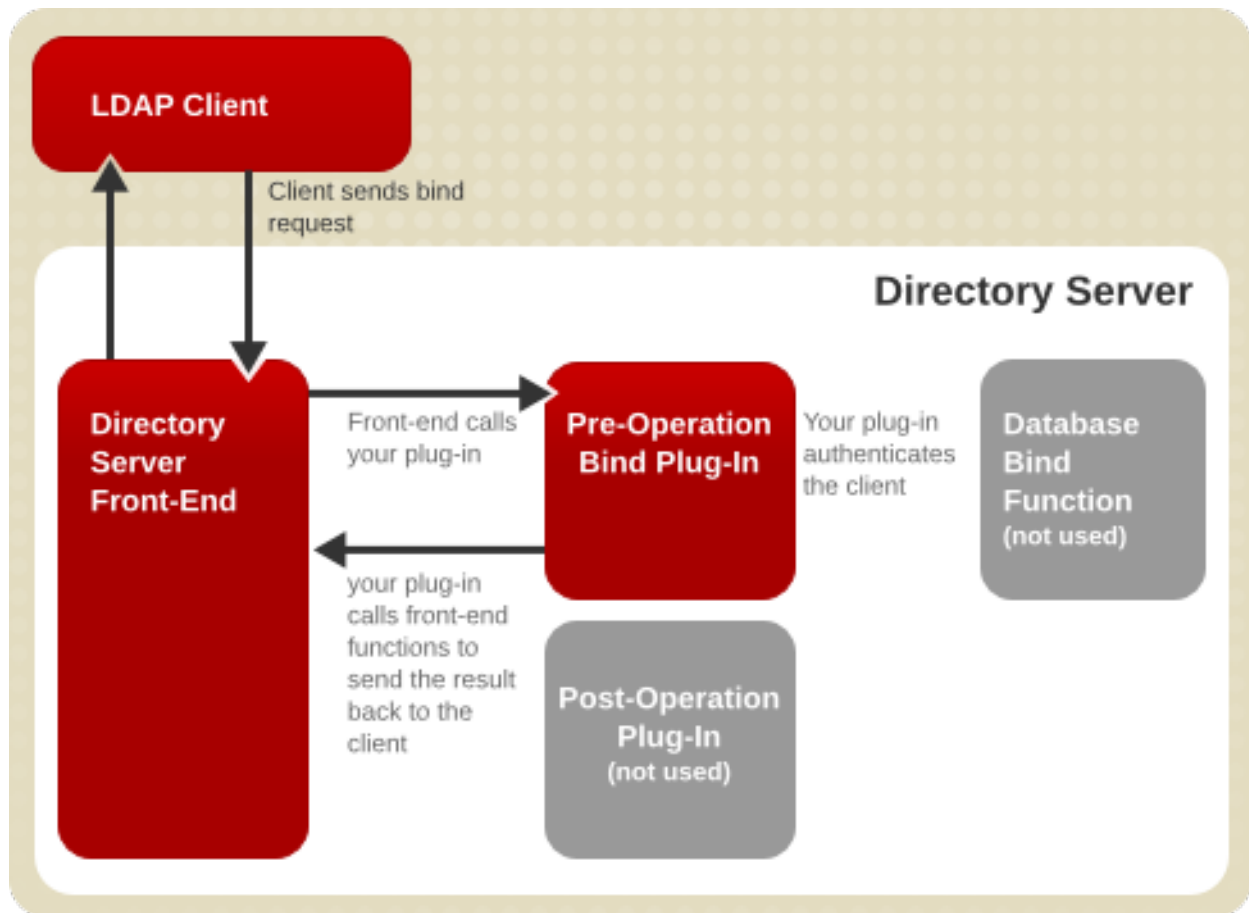


Figure 8.1. Using a Pre-Operation bind Plug-in Function to Handle Authentication

Figure 8.2, “How Your Pre-Operation Bind Plug-in Function Can Authenticate LDAP Clients” illustrates the steps that your pre-operation **bind** plug-in function must take to authenticate LDAP clients to the Directory Server.

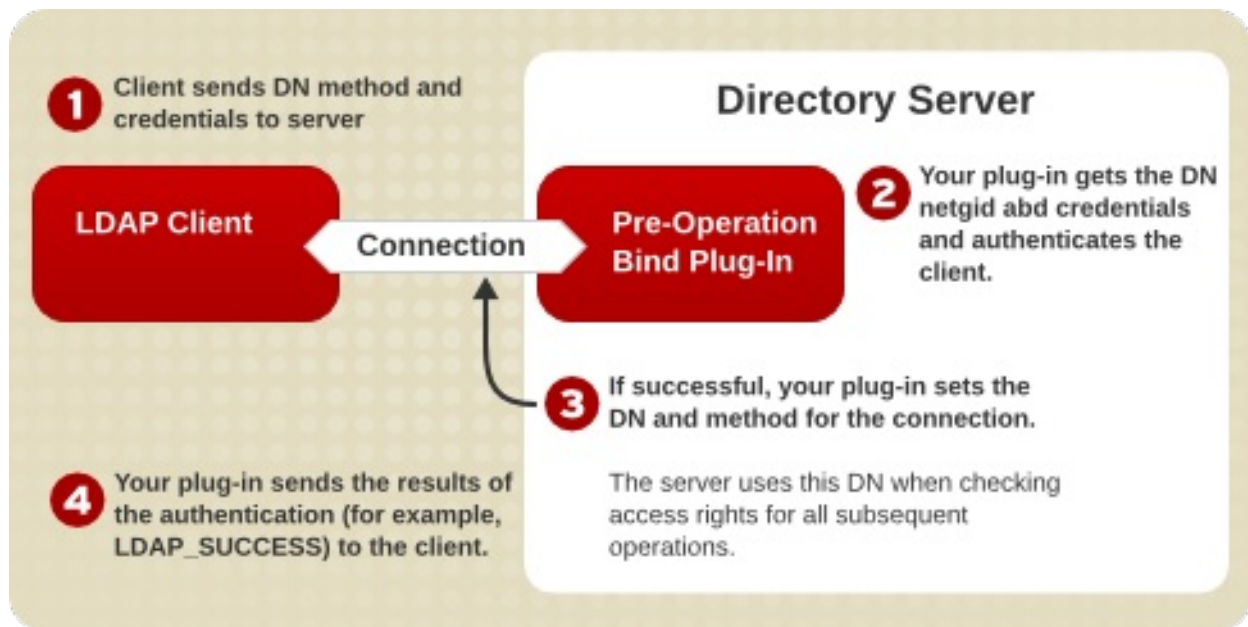


Figure 8.2. How Your Pre-Operation Bind Plug-in Function Can Authenticate LDAP Clients

8.5.1. Defining the Authentication Function



NOTE

Check out the sample **testbind.c** source file as an example of a pre-operation plug-in function that handles authentication. This file is in the `/usr/lib64/dirsrv/plugins/test-plugins/` directory.

Sample plug-in files are installed separately from other Directory Server packages, available at the 389 Directory Server repos, <http://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/plugins> and <http://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/slapd/test-plugins>. These sample plug-in files can be installed in any directory.

8.5.1.1. Getting and Checking the Bind Parameters

Call the `slapi_pblock_get()` function to get the values of the following parameters:

- **SLAPI_BIND_TARGET** - A string value specifying the DN as which the client is attempting to authenticate.
- **SLAPI_BIND_METHOD** - An integer value specifying the authentication method, such as **LDAP_AUTH_SIMPLE** or **LDAP_AUTH_SASL**.
- **SLAPI_BIND_CREDENTIALS** - A `berval` structure containing the credentials sent by the client.

If you plan to support authentication through SASL mechanisms, you should also get the value of the **SLAPI_BIND_SASLMECHANISM** parameter (a string value specifying the name of the SASL mechanism to use for authentication).

To handle the bind operation entirely within the plug-in, with no further frontend or backend processing, do the following:

- Determine if the client is requesting to **bind** as an anonymous user.

If the **SLAPI_BIND_METHOD** parameter is **LDAP_AUTH_SIMPLE** and the **SLAPI_BIND_CREDENTIALS** parameter is empty or NULL, the client is attempting to **bind** anonymously. Alternatively, disallow an anonymous bind and return the LDAP result code **LDAP_UNWILLING_TO_PERFORM**.

Call `slapi_send_ldap_result()` to send the LDAP result code **LDAP_SUCCESS** back to the client.

- If the **SLAPI_BIND_METHOD** parameter specifies a method that you do not recognize or support, call `slapi_send_ldap_result()` to send an **LDAP_STRONG_AUTH_NOT_SUPPORTED** result code back to the client.

In both cases, return a non-zero value as the plug-in function return value to prevent the server from calling the default backend function for authentication.

8.5.1.2. Getting the Entry and Checking the Credentials

Get the entry for the DN specified by the **SLAPI_BIND_TARGET** parameter, and compare the credentials in the **SLAPI_BIND_CREDENTIALS** parameter against the known credentials for that entry. In order to get the entry, you must perform an internal search. There are several functions that can be used, listed in order of increasing power and complexity:

- `slapi_search_internal_get_entry()` is useful to retrieve a single entry given a DN and a list of attributes.
- `slapi_search_internal_pb()` returns an array of matching entries.
- `slapi_search_internal_callback_pb()` returns each matching entry in a user-supplied callback.

Then, it is possible to use the attribute and value functions listed in [Table 5.1, “Frontend Functions for Manipulating Entries and Attributes”](#) to get the values.

Directory Server uses the **userPassword** attribute to store the credentials for an entry. The server encodes the password using the scheme specified in the **nsslapd-rootpwstoragescheme** attribute for the Directory Manager or **passwordStorageScheme** attribute for other users. These attributes are defined in the **cn=config** entry contained in the **dse.ldif** file. The scheme can be any of the following:

- **CLEAR** — No encryption is used, and can be defined using the **clear-password-storage-scheme** plug-in.
- **CRYPT** — Uses the Unix crypt algorithm, and can be defined using the **crypt-password-storage-scheme** plug-in.
- **SHA, SHA256, SHA384, SHA512** — Uses the Secure Hashing Algorithm, and can be defined using the **sha-password-storage-scheme** plug-in. **SHA** is **SHA-1**, which is 140 bits. For the others, the number indicates the number of bits used by the hash.
- **SSHA, SSHA256, SSHA384, SSHA512** — Uses the Salted Secure Hashing Algorithm, and can be defined using the **ssha-password-storage-scheme** plug-in. **SSHA** is **SSHA-1**, which is 140 bits, including the salt. For the others, the number indicates the number of bits used by the hash, including the salt.

To compare the client's credentials against the value of the **userPassword** attribute, you

can call the `slapi_pw_find_sv()` function. This function determines which password scheme was used to store the password and uses the appropriate comparison function to compare a given value against the encrypted value of the ***userPassword*** attribute.

8.5.1.3. What to Do If Authentication Fails

If authentication fails, send one of the following result codes back to the client:

- If no entry matches the DN specified by the client, send an `LDAP_NO_SUCH_OBJECT` result code back to the client.

When calling the `slapi_send_ldap_result()` function to send the result code back to the client, specify the closest matching DN as the ***matched*** argument.

- If the client fails to provide the necessary credentials, or if credentials cannot be found in the entry, send an `LDAP_INAPPROPRIATE_AUTH` result code back to the client.
- If the credentials specified by the client do not match the credentials found in the entry, send an `LDAP_INVALID_CREDENTIALS` result code back to the client.
- If a general error occurs, send an `LDAP_OPERATIONS_ERROR` result code back to the client.

Your function should also return a non-zero value.

You do not need to set any values for the ***SLAPI_CONN_DN*** parameter and the ***SLAPI_CONN_AUTHTYPE*** parameter. By default, these parameters are set to `NULL` and ***LDAP_AUTH_NONE***, which indicate that the client has bound anonymously.

8.5.1.4. What to Do If Authentication Succeeds

If the authentication is successful, your authentication function should:

- Call `slapi_pblock_set()` to set the values of the ***SLAPI_CONN_DN*** parameter and the ***SLAPI_CONN_AUTHTYPE*** parameter to the DN and authentication method.

This sets the DN and authentication method for the connection to the client. The server uses this DN and method in subsequent operations when checking access rights.

You can set ***SLAPI_CONN_AUTHTYPE*** to one of the following values:

- ***SLAPD_AUTH_NONE*** represents no authentication. (The client is binding anonymously.)
- ***SLAPD_AUTH_SIMPLE*** represents the simple authentication method.
- ***SLAPD_AUTH_SSL*** represents authentication through TLS.
- ***SLAPD_AUTH_SASL*** represents SASL authentication.

These values differ from the values in the ***SLAPI_BIND_METHOD*** parameter. The values listed above are string values defined in the ***slapi-plugin.h*** header file, whereas the values of the ***SLAPI_BIND_METHOD*** parameter (such as ***LDAP_AUTH_SIMPLE*** and ***LDAP_AUTH_SASL***) are integer values defined in the ***ldap.h*** header file.

- If required, specify the credentials that you want sent back to the client.

If the value of the **SLAPI_BIND_METHOD** parameter is **LDAP_AUTH_SASL** and you want to return a set of credentials to the client, call `slapi_pblock_set()` to set the **SLAPI_BIND_RET_SASLCREDS** parameter to the credentials.

- Send the result of the authentication process back to the client.

Call `slapi_send_ldap_result()` to send an **LDAP_SUCCESS** return code to the client.

Make sure that your function returns a non-zero value to bypass the default backend **bind** function and any post-operation plug-in functions.

8.5.2. Registering the SASL Mechanism

If you are using SASL as the authentication method, you need to register the SASL mechanisms that you plan to use.

In your initialization function (see [Section 2.2, “Writing Plug-in Initialization Functions”](#)), call the `slapi_register_supported_saslmechanism()` function and specify the name of the SASL mechanism. For example:

```
slapi_register_supported_saslmechanism( "babsmechanism" );
```

If you do not register your SASL mechanism, the Directory Server will send an **LDAP_AUTH_METHOD_NOT_SUPPORTED** result code back to the client and will not call your pre-operation **bind** function.

NOTE

Check out the sample **testsaslbind.c** source file as an example of a pre-operation plug-in function for SASL authentication. This file is in the `/usr/lib64/dirsrv/plugins/test-plugins/` directory.

Sample plug-in files are installed separately from other Directory Server packages, available at the 389 Directory Server repos, <http://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/plugins> and <http://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/slapd/test-plugins>. These sample plug-in files can be installed in any directory.

8.5.3. Example of a Pre-Operation Bind Plug-in

The following sections document an example of a pre-operation **bind** plug-in that handles authentication.



NOTE

Check out the sample **testbind.c** source file as an example of a pre-operation plug-in function that handles authentication. This file is in the `/usr/lib64/dirsrv/plugins/test-plugins/` directory.

Sample plug-in files are installed separately from other Directory Server packages, available at the 389 Directory Server repos, <http://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/plugins> and <http://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/slapd/test-plugins>. These sample plug-in files can be installed in any directory.

8.5.3.1. Example of a Pre-Operation Bind Function

The following is an example of a pre-operation **bind** function that authenticates clients and bypasses the default backend **bind** function. In this example, the function compares the client's credentials against the value of the **userpassword** attribute for the entry.

```
#include <stdio.h>
#include <string.h>
#include "dirsrv/slapi-plugin.h"

/* Pre-operation plug-in function */

int
test_bind(Slapi_PBlock *pb )
{
    int method, rc = LDAP_SUCCESS;
    struct berval *credentials;
    Slapi_Entry *e = NULL;
    Slapi_Attr *attr = NULL;
    Slapi_ValueSet *vs = NULL;
    Slapi_Value *sv_creds = NULL;
    Slapi_Value **sva = NULL;
    Slapi_DN *sdn = NULL;
    const char *dn = NULL;
    /* we only care about these attributes */
    char *attrlist[] = { "userPassword", NULL };
    void *my_plugin_id = NULL;

    /* Log a message to the server error log. */
    slapi_log_error( SLAPI_LOG_PLUGIN, "test_bind", "Pre-operation bind
function called.\n" );

    /* Gets parameters available when processing an LDAP bind operation.
*/

    if ( slapi_pblock_get( pb, SLAPI_BIND_TARGET_SDN, &sdn ) != 0 ||
        slapi_pblock_get( pb, SLAPI_BIND_METHOD, &method ) != 0 ||
        slapi_pblock_get( pb, SLAPI_BIND_CREDENTIALS, &credentials ) != 0 ||
        slapi_pblock_get( pb, SLAPI_PLUGIN_IDENTITY, &my_plugin_id ) != 0 )
    {
        slapi_log_error( SLAPI_LOG_PLUGIN, "test_bind", "Could not get
parameters for bind operation\n" );
        slapi_send_ldap_result( pb, LDAP_OPERATIONS_ERROR, NULL, NULL, 0, NULL );
    }
}
```

```

return( 1 );
}

sv_creds = slapi_value_new_berval(credentials); /* wrap in
Slapi_Value* */
dn = slapi_sdn_get_dn(sdn);

/* Check the authentication method */
switch( method ) {
    case LDAP_AUTH_SIMPLE:
        /* First, get the entry specified by the DN. */
        rc = slapi_search_internal_get_entry(sdn, attrlist, &e,
my_plugin_id);
        if ((LDAP_SUCCESS == rc) && (NULL != e)) {
            Slapi_Value *val = NULL;
            int num_vals = 0;
            int hint = 0;
            int i = 0;

            /* see if the entry has the userpassword attribute */
            if ( slapi_entry_attr_find( e, "userpassword" , &attr ) != 0 ) {
                slapi_log_error( SLAPI_LOG_PLUGIN, "test_bind" , "Entry has no
userpassword attribute\n" );
                rc = LDAP_INAPPROPRIATE_AUTH;
                break;
            }
            slapi_attr_get_valueset( attr, &vs ); /* must free vs */
            slapi_attr_get_numvalues(attr, &num_vals);
            sva = (Slapi_Value **) slapi_ch_calloc( (num_vals + 1),
sizeof(Slapi_Value *));

            /* Loop through all of our values s and create a value array */
            hint = slapi_valueset_first_value(vs, &val);
            while (val)
            {
                sva[i] = val;
                i++;
                hint = slapi_valueset_next_value(vs, hint, &val);
            }

            /* Next, check the credentials against the userpassword attribute of
that entry. */
            if ( slapi_pw_find_sv( sva, sv_creds ) != 0 ) {
                slapi_log_error( SLAPI_LOG_PLUGIN, "test_bind" ,
                "Credentials are not correct for the entry\n" );
                rc = LDAP_INVALID_CREDENTIALS;
                break;
            }

            /* Set the DN and the authentication method for the connection. */
            if ( slapi_pblock_set( pb, SLAPI_CONN_DN, slapi_ch_strdup( dn ) ) != 0
||
                slapi_pblock_set( pb, SLAPI_CONN_AUTHMETHOD, SLAPD_AUTH_SIMPLE) != 0
)
            {
                slapi_log_error( SLAPI_LOG_PLUGIN, "testbind_init" ,

```

```

        "Failed to set DN and auth method for connection\n" );
    rc = LDAP_OPERATIONS_ERROR;
    break;
}

/* Send a success result code back to the client. */
slapi_log_error( SLAPI_LOG_PLUGIN, "test_bind" , "Authenticated: %s\n",
dn );
rc = LDAP_SUCCESS;
    } else { /* error code or no entry */
        slapi_log_error( SLAPI_LOG_PLUGIN, "test_bind",
            "Could not find entry for %s: Error: %s\n",
            dn, (rc == LDAP_SUCCESS) ? "unknown" : ldap_err2string(rc) );
/* if the entry was null, there was probably an internal error */
if (LDAP_SUCCESS == rc) {
    rc = LDAP_OPERATIONS_ERROR;
}

    }
    break;

    /*
* If NONE is specified, the client is requesting to bind anonymously.
* Normally, this case should be handled by the server's front-end
* before it calls this plug-in function. Just in case this does
* get through to the plug-in function, you can handle this by
* sending a successful result code back to the client and returning 1,
* or if you do not want to support anon, return
LDAP_UNWILLING_TO_PERFORM
*/
case LDAP_AUTH_NONE:
    slapi_log_error( SLAPI_LOG_PLUGIN, "test_bind" , "Authenticating
anonymously\n" );
    rc = LDAP_SUCCESS; /* or return LDAP_UNWILLING_TO_PERFORM if anon not
supported */
    break;

    /* This plug-in does not support any other method of authentication
*/
case LDAP_AUTH_SASL:
default:
    slapi_log_error( SLAPI_LOG_PLUGIN, "test_bind" ,
        "Unsupported authentication method requested: %d\n" , method );
    rc = LDAP_AUTH_METHOD_NOT_SUPPORTED;
    break;
}

/* clean up - ok to pass NULL to these */
slapi_entry_free(e);
slapi_valueset_free(vs);
slapi_ch_free((void *)&sva);
slapi_value_free(&sv_creds);
slapi_sdn_free(&sdn);

/* actually return the result to the client */
slapi_send_ldap_result( pb, rc, NULL, NULL, 0, NULL );

```

```

    return( rc );
}

```

8.5.3.2. Example of an Initialization Function

To initialize your plug-in, write an initialization function to:

- Call `slapi_pblock_set()` to set the **SLAPI_PLUGIN_PRE_BIND_FN** parameter to the name of your pre-operation **bind** function. (For details, see [Section 2.2.3, "Registering Your Plug-in Functions"](#).)
- If you are using SASL as the authentication method, call the [Chapter 22, Functions for Syntax Plug-ins](#) function to register your SASL mechanism with the Directory Server.

The following is an example of an initialization function that registers the pre-operation **bind** function.

```

#include <stdio.h>
#include <string.h>
#include "dirsrv/slapi-plugin.h"

Slapi_PluginDesc bindpdesc = { "test-bind" , "Red Hat" , "0.5" ,"sample
bind pre-operation plugin" };

/* our plug-in identity . set in init function */
static Slapi_ComponentId *my_plugin_identity;

/* Initialization function */
#ifdef _WIN32
__declspec(dllexport)
#endif

int
testbind_init( Slapi_PBlock *pb )
{
    /*
     * Get our plug-in identity . we will need this to perform
     * any internal operations (search, modify, etc.)
     */
    slapi_pblock_get (pb, SLAPI_PLUGIN_IDENTITY, &my_plugin_identity);

    /*
     * Register the pre-operation bind function and specify
     * the server plug-in version.
     */
    if ( slapi_pblock_set( pb,
        SLAPI_PLUGIN_VERSION,SLAPI_PLUGIN_VERSION_03 ) != 0 ||
        slapi_pblock_set( pb, SLAPI_PLUGIN_DESCRIPTION,(void *)&bindpdesc
    ) != 0 ||
        slapi_pblock_set( pb, SLAPI_PLUGIN_PRE_BIND_FN,(void *) test_bind ) != 0
    )
    {
        slapi_log_error( SLAPI_LOG_PLUGIN, "testbind_init" , "Failed to
set version and function\n" );
    }
}

```



```

    return( -1 );
  }
  return( 0 );
}

```

8.5.3.3. Registering the Plug-in

To register the plug-in, add the following to the end of the `/etc/dirsrv/slapd-instance/dse.ldif` file:

```

dn: cn=Test Bind,cn=plugins,cn=config
objectClass: top
objectClass: nsSlapdPlugin
objectClass: extensibleObject
cn: Test Bind
nsslapd-pluginPath:/path/to/test-plugin.so
nsslapd-pluginInitfunc: testbind_init
nsslapd-pluginType: preoperation
nsslapd-pluginEnabled: on
nsslapd-plugin-depends-on-type: database
nsslapd-pluginId: test-bind

```

Check out the sample `testbind.c` source file as an example of a pre-operation plug-in function that handles authentication. This file is in the `/usr/lib64/dirsrv/plugins/test-plugins/` directory.



NOTE

Sample plug-in files are installed separately from other Directory Server packages, available at the 389 Directory Server repos, <http://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/plugins> and <http://git.fedorahosted.org/cgit/389/ds.git/tree/ldap/servers/slapd/test-plugins>. These sample plug-in files can be installed in any directory.

There are also examples in the source code itself. Look in the `/usr/lib64/dirsrv/plugins` directory for plug-ins that implement `SLAPI_PLUGIN_PRE_BIND_FN`.

The example code given is very basic. There are many other things which a bind plug-in could do. For example:

- Log the authentication attempt to the access log for auditing.
- Check for password expiration and use `slapi_add_pwd_control()` to send that information back to the client.
- See if the client has requested additional password policy information in a couple of different ways:

```

slapi_pblock_get (pb, SLAPI_REQCONTROLS, ...)
slapi_pblock_get (pb, SLAPI_PWPOLICY, ...)

```

Then send the requested information back to the client using `slapi_pwpolicy_make_response_control()`.

- Manage other aspects of password policy.

Finally, take a look at the server bind code in **bind.c** to see what sort of processing it does.

CHAPTER 9. WRITING ENTRY STORE/FETCH PLUG-INS

This chapter describes how to write entry store and entry fetch plug-ins. You can use these types of plug-ins to invoke functions before and after data is read from the default database.

9.1. HOW ENTRY STORE/FETCH PLUG-INS WORK

Entry store plug-in functions are called before data is written to the database. Entry fetch plug-in functions are called after data is read from the default database. This processing is illustrated in [Figure 9.1, “How the Server Calls Entry Store and Entry Fetch Plug-in Functions”](#).

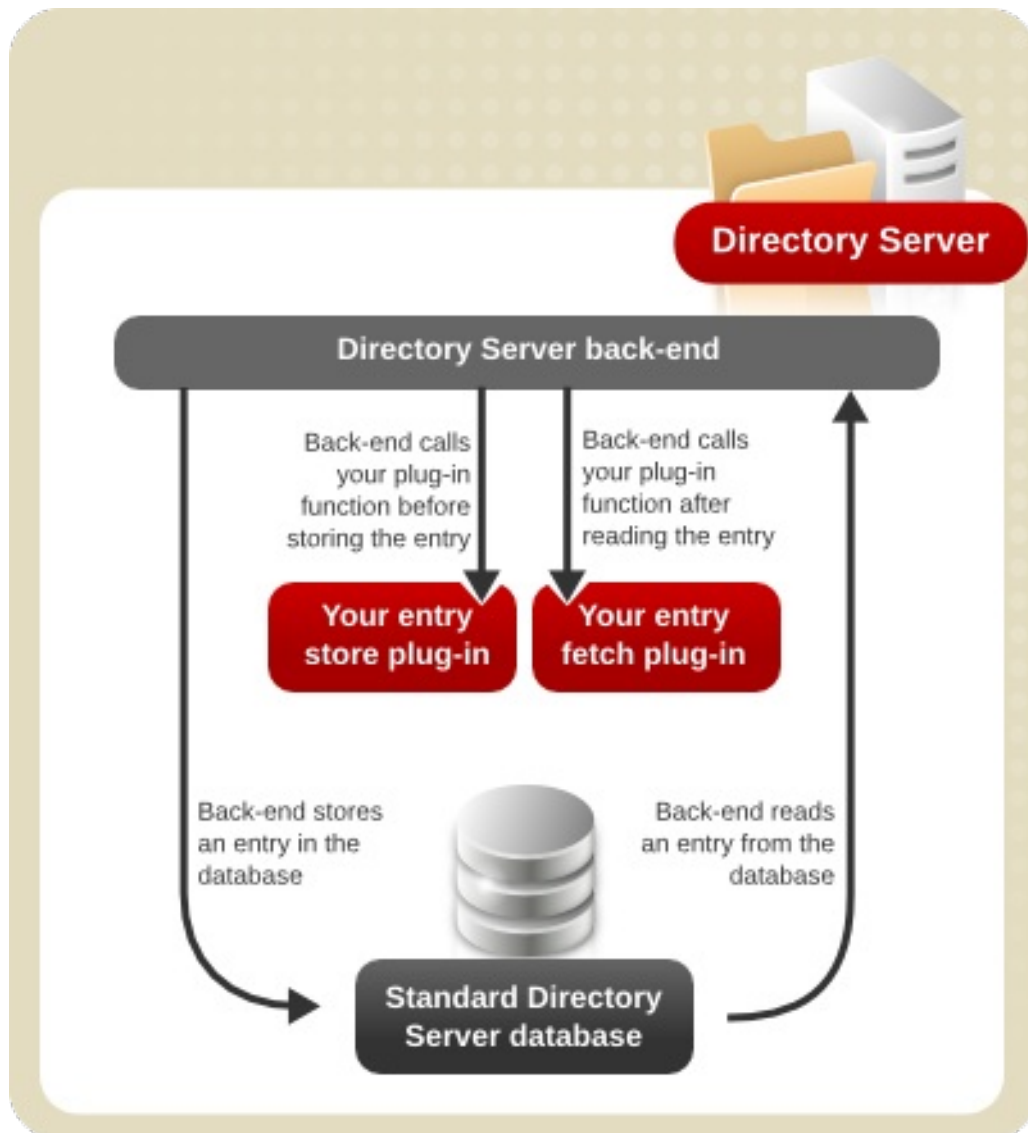


Figure 9.1. How the Server Calls Entry Store and Entry Fetch Plug-in Functions

9.2. WRITING ENTRY STORE/FETCH FUNCTIONS

Unlike most other types of plug-in functions, a parameter block is not passed to entry store and entry fetch plug-in functions when they are called. Instead, entry store and entry fetch plug-in functions must have the following prototype:

```
void function_name( char **entry, unsigned long *len );
```

The function parameters are described below:

- *entry* — Pointer to a string specifying the entry in LDIF format; for details on this format, see [slapi_str2entry\(\)](#) and [slapi_entry2str\(\)](#).
- *len* — Pointer to the length of the *entry* string.

Because the text of the entry is passed in as an argument, you can modify the entry before it gets saved to disk and modify the entry after it is read from disk. The pointer can be reallocated to get more room. For example:

```
void my_entry_fetch( char **entry, unsigned long *len )
{
    ...
    *len = newsize;
    *entry = slapi_ch_realloc(*entry, (*len) * sizeof(char));
    ... append to *entry ...
}
```

The server calls [slapi_ch_free\(\)](#) to free the memory, so to allocate more memory, use one of the slapi memory allocation functions.



NOTE

The **testentry.c** sample file has example entry store and entry fetch plug-in functions. This example file is with other examples in the **/usr/lib64/dirsrv/plugins** directory.

9.3. REGISTERING ENTRY STORE/FETCH FUNCTIONS

The plug-in configuration entry is much like that for other types of plug-ins. The **nsslapd-pluginType** is **ldbmentryfetchstore**. The plug-in init function should register the entry fetch callback using [slapi_pblock_set\(\)](#) with **SLAPI_PLUGIN_ENTRY_FETCH_FUNC** and register the entry store callback using [slapi_pblock_set\(\)](#) with **SLAPI_PLUGIN_ENTRY_STORE_FUNC**. It is not necessary to have both functions; it is possible to use only a fetch or only a store function.

It is also possible to register a start and a close function.

To register an entry store or entry fetch plug-in function, edit the Directory Server's **dse.ldif** file configuration file and add the entry:

1. Add the plug-in parameters to the **dse.ldif** file. For example:

```
# ldapmodify -D "cn=Directory Manager" -W -p 389 -h
server.example.com -x

dn: cn=Test entry,cn=plugins,cn=config
objectClass: top
objectClass: nsSlapdPlugin
objectClass: extensibleObject
cn: Test entry
nsslapd-pluginPath: /path/to/test-plugin.so
nsslapd-pluginInitfunc: testentry_init
```

```
nsslapd-pluginType: ldbmentryfetchstore  
nsslapd-pluginEnabled: on  
nsslapd-pluginId: test-entry
```

2. Restart the server to load the new plug-in.

```
# systemctl restart dirsrv.target
```

The **testentry.c** source file has an example plug-in function that implements entry store and entry fetch operations. Example files are installed in **/usr/lib64/dirsrv/plugins**.

CHAPTER 10. WRITING EXTENDED OPERATION PLUG-INS

This chapter explains how to write plug-in functions to handle extended operations. Extended operations are defined in the LDAP v3 protocol.

10.1. HOW EXTENDED OPERATION PLUG-INS WORK

You can define your own operation that you want the Red Hat Directory Server to perform. If you create a custom extended operation, you assign an *object identifier (OID)* to identify that operation. LDAP clients request the operation by sending an extended operation request. Within the request, the client specifies:

- The OID of the extended operation.
- Data specific to the extended operation.

When the Directory Server receives the request, the server calls the plug-in registered with the specified OID. The plug-in function has access to both the OID and the data in the client's request. The plug-in function can send a response back to the client containing an OID plus any additional data that might be needed.

In order to use extended operations, you need to configure both the Directory Server and the client so that they understand the specific extended operation that you want to perform.

10.2. WRITING EXTENDED OPERATION FUNCTIONS

Like other plug-in functions, extended operation functions pass a single parameter block (Slapi_PBlock) and return an integer value, as shown in the following example declaration:

```
int my_ext_func( Slapi_PBlock *pb );
```

When the Directory Server receives an extended operation request, the front-end calls the extended operation function with the OID value specified in the request. The front-end makes the following information available to the extended function in the form of parameters in a parameter block.

Table 10.1. Extended Function Parameter Block Arguments

Parameter ID	Data Type	Description
<i>SLAPI_EXT_OP_REQ_OID</i>	char *	Object ID (OID) of the extended operation specified in the request.
<i>SLAPI_EXT_OP_REQ_VALUE</i>	struct berval*	Value specified in the request.
<i>SLAPI_EXT_OP_RET_OID</i>	char *	Object ID (OID) that you want sent back to the client.

Parameter ID	Data Type	Description
<i>SLAPI_EXT_OP_RET_VALUE</i>	struct berval*	Value that you want sent back to the client.

Typically, your function should perform an operation on the value specified in the ***SLAPI_EXT_OP_REQ_VALUE*** parameter. After the extended operation completes, your function should return a single value, according to the following:

- If your function has sent a result code back to the client, you should return the value ***SLAPI_PLUGIN_EXTENDED_SENT_RESULT***. This indicates that the front-end does not need to send a result code.
- If your function has not sent a result code back to the client (for example, if the result is ***LDAP_SUCCESS***), your function should return an LDAP result code. The front-end will send this result code back to the client.
- If your function cannot handle the extended operation with the specified OID, your function should return the value ***SLAPI_PLUGIN_EXTENDED_NOT_HANDLED***. The front-end will send an ***LDAP_PROTOCOL_ERROR*** result code (with an **unsupported extended operation error message**) back to the client.



NOTE

Check out the ***testextendedop.c*** source file for a sample plug-in function (uncompiled C code) that implements an extended operation.

10.3. REGISTERING EXTENDED OPERATION FUNCTIONS

Extended operation functions are specified in a parameter block that you can set on server startup, in the same fashion as other server plug-in functions (refer to [Section 2.1.2, “Passing Data with Parameter Blocks”](#)).

In your initialization function, you can call the [slapi_pblock_set\(\)](#) function to set the ***SLAPI_PLUGIN_EXT_OP_FN*** parameter to your function and the ***SLAPI_PLUGIN_EXT_OP_OIDLIST*** parameter to the list of OIDs of the extended operations supported by your function.

NOTE

Previously, Red Hat recommended to use the ***SLAPI_PLUGIN_EXTENDEDOp*** parameter to register extended operation plug-ins. However, this function registration point has certain limitations. For example, this function registration is not correctly linked in a back end transaction, which can cause deadlock scenarios. To avoid similar issues, use the ***SLAPI_PLUGIN_BETXNEXTENDEDOp*** parameter. This parameter adds a database transaction to the extended operation.

The ***SLAPI_PLUGIN_EXT_OP_BACKEND_FN*** parameter enables the plug-in to inform the server which back end will be used. After that, the server initiates a transaction. The following diagram illustrates the process:

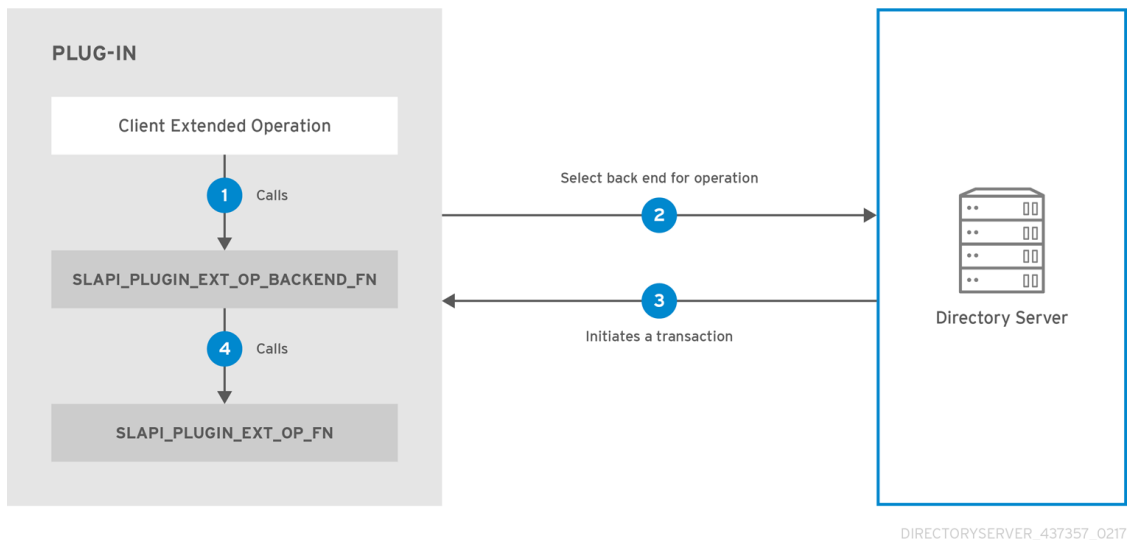


Figure 10.1. Registering Extended Operation Functions

You can write your initialization function so that the OID is passed in from the directive (refer to [Section 3.4, “Passing Extra Arguments to Plug-ins”](#), for details.) For example, the following initialization function sets the ***SLAPI_PLUGIN_EXT_OP_OIDLIST*** parameter to the additional parameters specified.

```
#include "dirsrv/slapi-plugin.h"

/*****
 * Pre Extended Operation, Backend selection
 *****/
static int extend_exop_backend(Slapi_PBlock *pb, Slapi_Backend **target)
{
    Slapi_DN *shared_sdn = NULL;
    char *shared_dn = NULL;
    int res = -1;
    /* Parse the oid and what exop wants us to do - You need to implement
    this function! */
    res = parse_exop_ber(pb, &shared_dn);
    if (res == LDAP_SUCCESS && shared_dn) {
        shared_sdn = slapi_sdn_new_dn_byref(shared_dn);
        /* Take the SDN of the operation, and use it to find
        The backend we plan to operate on.
        */
        *target = slapi_be_select(shared_sdn);
    }
}
```



```

        slapi_sdn_free(&shared_sdn);
        res = LDAP_SUCCESS;
    }
    return res;
}

int
extended_init( Slapi_PBlock *pb )
{
    int i;
    char **argv;
    char **oids;

    /* Get the additional arguments specified in the directive */
    if ( slapi_pblock_get( pb, SLAPI_PLUGIN_ARGV, &argv ) != 0 ) {
        slapi_log_error( SLAPI_LOG_PLUGIN, "extended_initx" , "Server
could not get argv.\n" );
        return( -1 );
    }
    if ( argv == NULL ) {
        slapi_log_error( SLAPI_LOG_PLUGIN, "extended_init" , "Required
argument oid is missing\n" );
        return( -1 );
    }

    /* Get the number of additional arguments and copy them. */
    for ( i = 0; argv[i] != NULL; i++ );

    oids = (char **) slapi_ch_malloc( (i+1) * sizeof(char *) );
    for ( i = 0; argv[i] != NULL; i++ ) {
        oids[i] = slapi_ch_strdup( argv[i] );
    }
    oids[i] = NULL;

    /* Specify the version of the plug-in */
    if ( slapi_pblock_set( pb, SLAPI_PLUGIN_VERSION,
SLAPI_PLUGIN_VERSION_03 ) != 0 ||

        /* Specify the OID of the extended operation */
        slapi_pblock_set( pb, SLAPI_PLUGIN_EXT_OP_OIDLIST, (void*) oids ) != 0
    ||
        slapi_pblock_set(pb, SLAPI_PLUGIN_EXT_OP_BACKEND_FN, (void
*)extend_exop_backend) != 0 ||

        /* Specify the function that the server should call */
        slapi_pblock_set( pb, SLAPI_PLUGIN_EXT_OP_FN, (void*)extended_op ) != 0
    )
    {
        slapi_log_error( SLAPI_LOG_PLUGIN, "extended_init" , "An error
occurred.\n" );
        return( -1 );
    }
    slapi_log_error( SLAPI_LOG_PLUGIN, "extended_init" , "Plugin

```

```
    successfully registered.\n" );
    return(0);
}
```

To add the plug-in configuration, use **ldapmodify** to add the entry. For example:

```
# ldapmodify -a -D "cn=Directory Manager" -W -p 389 -h server.example.com
-x

dn: cn=Test ExtendedOp,cn=plugins,cn=config
objectClass: top
objectClass: nsSlapdPlugin
objectClass: extensibleObject
cn: Test ExtendedOp
nsslapd-pluginPath: /path/to/test-plugin.so
nsslapd-pluginInitfunc: testexop_init
nsslapd-pluginType: beextendedop
nsslapd-pluginEnabled: on
nsslapd-plugin-depends-on-type: database
nsslapd-pluginId: test-extendedop
nsslapd-pluginarg0: 1.2.3.4
```

Alternatively, shut down the server, add the plug-in parameters to the **dse.ldif** file, and restart the server.

Check out the **testextendedop.c** source file has for an example plug-in function that implements an extended operation. Example files are installed in **/usr/lib64/dirsrv/plugins**.

10.4. SPECIFYING START AND CLOSE FUNCTIONS

For each extended operation plug-in, you can specify the name of a function to be called after the plug-in starts and before the plug-in is stopped or disabled.

Use the following parameters to specify these functions:

- **SLAPI_PLUGIN_START_FN** — Specifies the function called after the plug-in starts.
- **SLAPI_PLUGIN_CLOSE_FN** — Specifies the function called before the plug-in is stopped or disabled.

CHAPTER 11. WRITING MATCHING RULE PLUG-INS

This chapter explains how to write plug-in functions that handle matching rules.

11.1. UNDERSTANDING MATCHING RULES

A *matching rule* specifies how one or more attributes of a particular syntax should be compared against assertion values. For example, a matching rule that specifies a “sound-alike” comparison attempts to match values that sound like the specified value. Each matching rule is identified by a unique OID (for example, “1.2.3.4”).

LDAPv3 clients can specify a matching rule as part of a search filter in a search request. This type of search filter is called an *extensible match* filter.

11.1.1. Extensible Match Filters

In an *extensible match* filter, the client specifies that it wants to use the matching rule to compare a specified value against the values of entries in the directory. (For example, an extensible match filter might find all entries in which the *sn* attribute “sounds like” *melon*.)

An extensible match filter contains the following information:

- The OID of the matching rule or the attribute type that you want to search (or both).
- The value for which to search.
- A preference indicating whether to also search the attributes in the DN.

For example, if the OID **1.2.3.4** identifies a matching rule that performs “sounds like” matches, the following extensible match filter attempts to find entries where the *mail* attribute “sounds like” *moxie*: (**mail:1.2.3.4:=moxie**)

In the search filter, the client can specify the OID that identifies a matching rule, and the attribute type. This indicates that the value in the filter should be compared against the attribute using the matching rule.

For example, if the OID **1.2.3.4** specifies a “sound-alike” match and if the string representation of the search filter is (**uid:1.2.3.4:=moxie**), it indicates that the client wants to find entries in which the value of the *uid* attribute sounds like *moxie*.

Although the LDAPv3 standard allows clients to omit the OID or the attribute type, at this time, the Red Hat Directory Server only supports extensible match filters that specify both the OID and attribute type.

The filter can also specify a preference indicating whether to include DN attributes in the search. For example, if the OID 1.2.3.4 specifies a “sound-alike” match and if the string representation of the search filter is (**sn:dn:1.2.3.4:=moxie**), it indicates that the client wants to find all entries in which the value of the *sn* attribute or the attributes in the DN (for example, *uid*, *cn*, *ou*, or *o*) sound like *moxie*.

11.1.2. Extensible Match Filters in the Directory Server

Directory Server already includes support for certain matching rules, which are used to determine the collation order and operator for searches of international data.

You can enable the Directory Server to handle your own matching rules for extensible match searches by defining your own matching rules plug-ins and registering them with the server.

You can also build indexes to improve the performance of search operations that use extended match filters.

11.2. UNDERSTANDING MATCHING RULE PLUG-INS

A matching rule plug-in can create filters that the server can use when handling extensible search filters. A matching rule plug-in can also create indexes to index entries for extensible searches.

11.2.1. Functions Defined in Matching Rule Plug-ins

The matching rule plug-in consists of the following:

- An indexer function. [Optional].
- A filter function.
- A filter function that uses the index to speed up searches. [Optional].
- A function to destroy a filter. [Optional].
- A function to destroy an indexer. [Optional].
- A factory function to create filters.
- A factory function to create indexers. [Optional].
- A close function to clean up before server shutdown. [Optional].
- An initialization function to register the factory functions and the close function.

When the server starts and loads the matching rule plug-in, it calls the initialization function. In this function, you pass the server the pointers to the factory functions and the close function. The server calls these functions when needed. Refer to [Section 11.2.2, “How Matching Rules Are Identified”](#), and [Section 11.2.3, “How the Server Associates Plug-ins with OIDs”](#), for details.

11.2.2. How Matching Rules Are Identified

Matching rules are identified by OID. When the server encounters an OID in the following situations, it attempts to find the matching rule plug-in that handles the matching rule with that OID.

The server can encounter a matching rule OID in the following situations:

- When reading in the server configuration file, the server may encounter an index directive that specifies the OID of the matching rule. For example:

```
| index attribute_name filter_type matching_rule_oid
```

If the OID is associated with a matching rule plug-in, the server adds this OID to the list of matching rule OIDs to use for indexing.

For information on setting up the server to index based on matching rule, refer to [Section 11.3, “Indexing Based on Matching Rules”](#).

- The server may receive an LDAP search request with an “extensible match” filter specifying the OID of the matching rule. For example, a string representation of an extensible match filter might be:

```
(sn:dn:1.2.3.4:=Jensen)
```

The search filter above specifies that the server should use the matching rule identified by the OID **1.2.3.4** to search for the value **Jensen** in the **sn** attribute and in all attributes in the DN.

For information on setting up the server to handle extensible match filters, refer to [Section 11.4, “Handling Extensible Match Filters”](#).

- The server may receive an LDAP search request containing a sorting control, and the sorting control specifies the OID of the matching rule.

For information on setting up the server to sort based on matching rules, refer to [Section 11.5, “Handling Sorting by Matching Rules”](#).

In all of these situations, the server uses the matching rule OID to find the plug-in responsible for handling the rule. Refer to [Section 11.2.3, “How the Server Associates Plug-ins with OIDs”](#), for details.

11.2.3. How the Server Associates Plug-ins with OIDs

The server associates plug-ins with OIDs using the following process:

- When the server encounters the OID for a matching rule, it attempts to find the plug-in associated with that matching rule.
- If no plug-in is associated with the matching rule, the server calls each matching rule plug-in to find one that handles the specified matching rule.
- When the server finds a plug-in that handles the matching rule, the server creates an association between the plug-in and the matching rule OID for future reference.
- If no matching rule plug-in supports the specified OID, the server returns an LDAP_UNAVAILABLE_CRITICAL_EXTENSION error to the client.

11.2.3.1. Finding a Plug-in for Indexing

To determine which matching rule plug-in is responsible for indexing an attribute with a given matching rule (based on its OID), the server does the following for each plug-in:

1. In a new [Slapi_PBlock](#) parameter block, the server sets the OID in the **SLAPI_PLUGIN_MR_OID** parameter.
2. The server then calls the indexer factory function (specified in the **SLAPI_PLUGIN_MR_INDEXER_CREATE_FN** parameter) for the plug-in.
3. The server then checks the **SLAPI_PLUGIN_MR_INDEX_FN** parameter.

- If the parameter is NULL, the plug-in does not handle the matching rule specified by that OID.
 - If the parameter returns an indexer function, this plug-in handles the matching rule specified by that OID.
4. Finally, the server frees the parameter block from memory.

At some point, the server may also call the indexer destructor function (specified in the ***SLAPI_PLUGIN_MR_DESTROY_FN*** parameter) to free the indexer object that was created by the indexer factory function.

11.2.3.2. Finding a Plug-in for Searching

To determine which matching rule plug-in is responsible for handling an extensible match filter for a given matching rule (based on its OID), the server does the following for each plug-in:

1. In a new [Slapi_PBlock](#) parameter block, the server sets the following parameters:
 - Sets the OID in the ***SLAPI_PLUGIN_MR_OID*** parameter.
 - Sets the type (from the filter) in the ***SLAPI_PLUGIN_MR_TYPE*** parameter.
 - Sets the value (from the filter) in the ***SLAPI_PLUGIN_MR_VALUE*** parameter.
2. The server then calls the filter factory function (specified in the ***SLAPI_PLUGIN_MR_FILTER_CREATE_FN*** parameter) for the plug-in.
3. The server checks the ***SLAPI_PLUGIN_MR_FILTER_MATCH_FN*** parameter.
 - If the parameter is NULL, the plug-in does not handle the matching rule specified by that OID.
 - If the parameter returns a filter matching function, this plug-in handles the matching rule specified by that OID.
4. Finally, the server gets the following information from the plug-in for future use:
 - The filter index function specified in the ***SLAPI_PLUGIN_MR_FILTER_INDEX_FN*** parameter.
 - The value specified in the ***SLAPI_PLUGIN_MR_FILTER_REUSABLE*** parameter.
 - The filter reset function specified in the ***SLAPI_PLUGIN_MR_FILTER_RESET_FN*** parameter.
 - The filter object specified in the ***SLAPI_PLUGIN_OBJECT*** parameter.
 - The filter destructor function specified in the ***SLAPI_PLUGIN_DESTROY_FN*** parameter.

Information specified in the filter object is used by both the filter index function and the filter matching function.

11.2.4. How the Server Uses Parameter Blocks

The server uses parameter blocks as a means to pass information to and from plug-in functions.

When calling your matching rule plug-in functions, the server creates a new parameter block, set some input parameters, and pass the parameter block to your function. After retrieving output parameters from the block, the server typically frees the parameter block from memory.

In general, you should not expect a parameter block to be passed between plug-in functions. The value of a parameter set by one plug-in function may not necessarily be accessible to other plug-in functions, because each function is usually passed a new and different parameter block.

11.3. INDEXING BASED ON MATCHING RULES

This section explains how to set up the server to index entries using a matching rule.



NOTE

You also need to define an initialization function to register your indexer factory function.

11.3.1. How the Server Sets Up the Index

When the server encounters a matching rule OID in an *index* directive in the server configuration file, the server determines which plug-in supports the matching rule identified by the OID. Refer to [Section 11.2.3, “How the Server Associates Plug-ins with OIDs”](#), for details.

The server gets the OID returned in the **SLAPI_PLUGIN_MR_OID** parameter and associates this OID with the rest of the attribute indexing information (for example, the attribute type and the type of index) for future reference.

When adding, modifying, or deleting the values of an attribute, the server checks this information to determine if the attribute is indexed. Refer to [Section 11.3.2, “How the Server Updates the Index”](#), for information on how attributes are indexed.

11.3.2. How the Server Updates the Index

When a value is added, modified, or removed from an attribute in an entry (or when the RDN of an entry is changed), the server performs the following tasks if that attribute has an index that uses matching rules:

1. In a new **Slapi_PBlock** parameter block, the server sets the following parameters:
 - Sets the OID in the **SLAPI_PLUGIN_MR_OID** parameter.
 - Sets the attribute type (of the value being added, modified, or removed) in the **SLAPI_PLUGIN_MR_TYPE** parameter.
2. Next, the server calls the indexer factory function (specified in the **SLAPI_PLUGIN_MR_INDEXER_CREATE_FN** parameter) for the plug-in to create the indexer object.
3. The server generates the index keys for the values to be added or deleted:

- The server first verifies that the **SLAPI_PLUGIN_MR_INDEX_FN** parameter specifies an indexer function and the **SLAPI_PLUGIN_MR_OID** parameter specifies the official OID of the matching rule.
 - If these are both set, the server sets the **SLAPI_PLUGIN_MR_VALUES** parameter to the array of berval structures containing the new or modified values that need to be indexed and calls the indexer function.
 - Next, the server gets the value of the **SLAPI_PLUGIN_MR_KEYS** parameter, which is an array of berval structures containing the keys corresponding to the values.
4. The server inserts or deletes the keys and values in the index for that attribute.
 5. The server calls the indexer destructor function (specified in the **SLAPI_PLUGIN_MR_DESTROY_FN** parameter) to free the indexer object.

At the end of the process, the server frees any parameter blocks that were allocated during the process.

11.3.3. Writing the Indexer Factory Function

The indexer factory function takes a single [Slapi_PBlock](#) argument. This function should be thread-safe. The server may call this function concurrently.

The indexer factory function should perform the following tasks:

1. Get the OID from the **SLAPI_PLUGIN_MR_OID** parameter, and determine whether that OID is supported by your plug-in.
 - If the OID is not supported, you need to return the result code `LDAP_UNAVAILABLE_CRITICAL_EXTENSION`.
 - If the OID is supported, continue with this process.
2. Get the value of the **SLAPI_PLUGIN_MR_USAGE** parameter. This parameter should have one of the following values:
 - If the value is **SLAPI_PLUGIN_MR_USAGE_SORT**, the server is calling your function to sort search results. Refer to [Section 11.5, “Handling Sorting by Matching Rules”](#), for more information.
 - If the value is **SLAPI_PLUGIN_MR_USAGE_INDEX**, the server is calling your function to index an entry.

You can use this information to set different information in the indexer object or to set a different indexer function, based on whether the function is being called to index or to sort.

3. You can also get any data that you set in the **SLAPI_PLUGIN_PRIVATE** parameter during initialization. (Refer to [Section 11.7, “Writing an Initialization Function”](#).)
4. Create an indexer object containing any information that you want to pass to the indexer function.
5. Set the following parameters:
 - Set the **SLAPI_PLUGIN_MR_OID** parameter to the official OID of the matching rule (if the value of that parameter is not the official OID).

- Set the ***SLAPI_PLUGIN_OBJECT*** parameter to the indexer object.
 - Set the ***SLAPI_PLUGIN_MR_INDEX_FN*** parameter to the indexer function. (Refer to [Section 11.3.5, “Writing the Indexer Function”](#).)
 - Set the ***SLAPI_PLUGIN_DESTROY_FN*** parameter to the function responsible for freeing any memory allocated by the factory function, such as the indexer object. refer to [Section 11.3.3, “Writing the Indexer Factory Function”](#), for details.
6. Return 0 (or the result code LDAP_SUCCESS) if everything completed successfully.

11.3.4. Getting and Setting Parameters in Indexer Factory Functions

The following table summarizes the different parameters that the indexer factory function should get and set in the parameter block that is passed in.

Table 11.1. Input and Output Parameters Available to an Indexer Factory Function

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_MR_OID</i>	char *	Input parameter. Matching rule OID (if any) specified in the <i>index</i> directive.
<i>SLAPI_PLUGIN_MR_TYPE</i>	char *	Input parameter. Attribute type (if any) specified in the <i>index</i> directive.

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_MR_USAGE</i>	unsigned int	<p>Input parameter. Specifies the intended use of the indexer object. This parameter can have one of the following values:</p> <ul style="list-style-type: none"> • <i>SLAPI_PLUGIN_MR_USAGE_INDEX</i> specifies that the indexer object should be used to index entries. • <i>SLAPI_PLUGIN_MR_USAGE_SORT</i> specifies that the indexer object should be used to sort entries. <p>You can use this to specify different information in the indexer object or different indexer functions, based on whether the plug-in is used for indexing or sorting. For information on sorting search results, refer to Section 11.5, “Handling Sorting by Matching Rules”.</p>
<i>SLAPI_PLUGIN_PRIVATE</i>	void *	<p>Input parameter. Pointer to any private data originally specified in the initialization function. Refer to Section 11.7, “Writing an Initialization Function”, for details.</p>
<i>SLAPI_PLUGIN_MR_OID</i>	char *	<p>Output parameter. Official matching rule OID of the index.</p>
<i>SLAPI_PLUGIN_MR_INDEX_FN</i>	void * (function pointer)	<p>Output parameter. Name of the function called by the server to generate a list of keys used for indexing a set of values.</p>
<i>SLAPI_PLUGIN_DESTROY_FN</i>	void * (function pointer)	<p>Output parameter. Name of the function to be called to free the indexer object.</p>

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_OBJECT</i>	void *	Output parameter. Pointer to the indexer object created by your factory function.

11.3.5. Writing the Indexer Function

The indexer function takes a single [Slapi_PBlock](#) argument. This function will never be called for the same indexer object concurrently. (If you plan to manipulate global variables, remember that the server can call this function concurrently for different indexer objects.)

The indexer function should perform the following tasks:

1. Get the values of the following parameters:
 - Get the indexer object from the ***SLAPI_PLUGIN_OBJECT*** parameter (if the parameter is set).
 - Get the array of values that you want indexed from the ***SLAPI_PLUGIN_MR_VALUES*** parameter.
2. Generate index keys for these values, and set the ***SLAPI_PLUGIN_MR_KEYS*** parameter to the array of these keys.
3. Return 0 (or the result code ***LDAP_SUCCESS***) if everything completed successfully.

The server adds or removes the keys and the corresponding values from the appropriate indexes.

11.3.6. Getting and Setting Parameters in Indexer Functions

The following table summarizes the different parameters that the indexer function should get and set in the parameter block that is passed in.

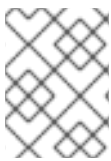
Table 11.2. Input and Output Parameters Available to an Indexer Function

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_MR_VALUES</i>	struct berval **	Input parameter. Pointer to an array of berval structures containing the values of the entry's attributes that need to be indexed.
<i>SLAPI_PLUGIN_OBJECT</i>	void *	Input parameter. Pointer to the indexer object created by the indexer factory function. Refer to Section 11.3.3, “Writing the Indexer Factory Function” , for details.

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_MR_KEYS</i>	struct berval **	Output parameter. Keys generated for the values specified in the <i>SLAPI_PLUGIN_MR_VALUES</i> parameter. The server creates indexes using these keys.

11.4. HANDLING EXTENSIBLE MATCH FILTERS

This section explains how to set up the server to process searches that use extensible match filters (matching rules).



NOTE

You also need to define an initialization function to register your filter factory function.

11.4.1. How the Server Handles the Filter

When the server processes a search request that has an extensible match filter, the server performs the following tasks:

1. First, the server finds the plug-in associated with this OID, if an association between the OID and plug-in has already been made.

If no association has been made, the server attempts to find a matching rule plug-in that handles the OID. Refer to [Section 11.2.3, “How the Server Associates Plug-ins with OIDs”](#), for details.

2. The server then attempts to generate a list of search result candidates from the indexes. In a new Slapi_PBlock parameter block:
 1. The server puts the filter object in the ***SLAPI_PLUGIN_OBJECT*** parameter and calls the filter index function (specified in the ***SLAPI_PLUGIN_MR_FILTER_INDEX_FN*** parameter).
 2. The server checks the value of the ***SLAPI_PLUGIN_MR_QUERY_OPERATOR*** parameter. If the operator is a known type (such as ***SLAPI_OP_EQUAL***), the server will use the operator when searching the index for candidates. For details, refer to [Section 11.4.2, “Query Operators in Matching Rules”](#).
 3. The server sets the ***SLAPI_PLUGIN_MR_VALUES*** parameter to each of the values specified in the filter and calls the indexer function (which is specified in the ***SLAPI_PLUGIN_MR_INDEX_FN*** parameter) to generate the key (specified in the ***SLAPI_PLUGIN_MR_KEYS*** parameter).
 4. The server uses the keys and the query operator to find potential candidates in the indexes.

The server considers all entries to be potential candidates if at least one of the following is true:

- The matching rule plug-in has no indexer function (specified in the ***SLAPI_PLUGIN_MR_INDEX_FN*** parameter).
 - No index applies to the search (for example, if the query operator does not correspond to an index).
 - No keys are generated for the specified values.
3. For each candidate entry, the server performs the following tasks to determine if the entry matches the search filter:
 1. The server calls the filter matching function (which is specified in the ***SLAPI_PLUGIN_MR_FILTER_MATCH_FN*** parameter), passing in the filter object, the entry, and the attributes of the entry.
 2. If the entry does not match, but the search request also specifies that the attributes in the DN should be searched, the server calls the filter matching function again, passing in the filter object, the entry, and the attributes in the DN.
 4. The server then checks the value returned by the filter matching function:
 - If the function returns 0, the entry matched the search filter.
 - If the function returns -1, the entry did not match the search filter.
 - If the function returns an LDAP error code (a positive value), an error occurred.
 5. If the entry matches the filter, the server verifies that the entry is in the scope of the search before returning the entry to the LDAP client as a search result.

11.4.2. Query Operators in Matching Rules

As discussed in [Section 11.4.1, “How the Server Handles the Filter”](#), the server uses a query operator when searching the index for possible candidates.

This applies to the ldbm default backend database. If you are using your own back-end or if you have not set up indexing by matching rules, the server does not make use of the query operator.

The server checks the value of the ***SLAPI_PLUGIN_MR_QUERY_OPERATOR*** parameter to determine which operator is specified. The following table lists the possible values for this parameter.

Table 11.3. Query Operators in Extensible Match Filters

Operator	Description
<i>SLAPI_OP_LESS</i>	<
<i>SLAPI_OP_LESS_OR_EQUAL</i>	<=
<i>SLAPI_OP_EQUAL</i>	=
<i>SLAPI_OP_GREATER_OR_EQUAL</i>	>=

Operator	Description
<i>SLAPI_OP_GREATER</i>	>

If the query operator is ***SLAPI_OP_EQUAL***, the server attempts to find the keys in the index that match the value specified in the search filter. In the case of the other query operators, the server attempts to find ranges of keys that match the value.

11.4.3. Writing a Filter Factory Function

The filter factory function takes a single [Slapi_PBlock](#) argument. This function should be thread-safe. The server may call this function concurrently. (Each incoming LDAP request is handled by a separate thread. Multiple threads may call this function if processing multiple requests that have extensible match filters.)

file:///home/joakes/svn/8.0/Plugin_Programming_Guide/tmp/en-US/html/Plugin_Programming_Guide-Handling_Extensible_Match_Filters-Getting_and_Setting_Parameters_in_Filter_Factory_Functions.html The filter factory function should perform the following tasks:

1. Get the OID from the ***SLAPI_PLUGIN_MR_OID*** parameter and determine whether that OID is supported by your plug-in.
 - If the OID is not supported, you need to return the result code ***LDAP_UNAVAILABLE_CRITICAL_EXTENSION***. The server will send this back to the client.
 - If the OID is supported, continue with this process.
2. Get and check the values of the ***SLAPI_PLUGIN_MR_TYPE*** and ***SLAPI_PLUGIN_MR_VALUE*** parameters.

The values of these parameters are the attribute type and value specified in the extensible match filter.

3. You can also get any data that you set in the ***SLAPI_PLUGIN_PRIVATE*** parameter during initialization. Refer to [Section 11.7, “Writing an Initialization Function”](#).
4. Create a filter object, and include the following information:
 - The official OID of the matching rule. [Optional].
 - The attribute type specified in the filter.
 - The value specified in the filter.
 - Any additional data that you want made available to the filter index function (for example, the query operator, if specified in the filter).

The server will call your filter index function at a later time to extract this information from the filter object.

5. Set the following parameters:

- Set the ***SLAPI_PLUGIN_MR_OID*** parameter to the official OID of the matching rule if the value of that parameter is not the official OID. [Optional].
 - Set the ***SLAPI_PLUGIN_OBJECT*** parameter to the filter object.
 - Set the ***SLAPI_PLUGIN_MR_FILTER_INDEX_FN*** parameter to the filter index function if you have set up indexes based on this matching rule. (cf. [Section 11.4.5, “Writing a Filter Index Function”](#)) [Optional].
 - Set the ***SLAPI_PLUGIN_MR_FILTER_MATCH_FN*** parameter to the filter matching function. Refer to [Section 11.4.7, “Writing a Filter Matching Function”](#).
 - Set the ***SLAPI_PLUGIN_DESTROY_FN*** parameter to the function responsible for freeing the filter object, if you have defined this function. [Optional]. Refer to [Section 11.6, “Writing a Destructor Function”](#), for details.
6. Return 0 (or the result code `LDAP_SUCCESS`) if everything completed successfully.

11.4.4. Getting and Setting Parameters in Filter Factory Functions

The following table summarizes the different parameters that the filter factory function should get and set in the parameter block that is passed in.

Table 11.4. Input and Output Parameters Available to a Filter Factory Function

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_MR_OID</i>	char *	Input parameter. Matching rule OID (if any) specified in the extensible match filter.
<i>SLAPI_PLUGIN_MR_TYPE</i>	char *	Input parameter. Attribute type (if any) specified in the extensible match filter.
<i>SLAPI_PLUGIN_MR_VALUE</i>	struct berval *	Input parameter. Value specified in the extensible match filter.
<i>SLAPI_PLUGIN_PRIVATE</i>	void *	Input parameter. Pointer to any private data originally specified in the initialization function. Refer to Section 11.7, “Writing an Initialization Function” , for details.
<i>SLAPI_PLUGIN_MR_FILTER_MATCH_FN</i>	mrFilterMatchFn (function pointer)	Output parameter. Name of the function called by the server to match an entry's attribute values against the value in the extensible search filter.

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_MR_FILTER_INDEX_FN</i>	void * (function pointer)	Output parameter. Name of the function called by the server to generate a list of keys used for indexing a set of values.
<i>SLAPI_PLUGIN_DESTROY_FN</i>	void * (function pointer)	Output parameter. Name of the function to be called to free the filter object.
<i>SLAPI_PLUGIN_OBJECT</i>	void *	Output parameter. Pointer to the filter object created by your factory function.

11.4.5. Writing a Filter Index Function

The filter index function takes a single [Slapi_PBlock](#) argument. This function will never be called for the same filter object concurrently. (If you plan to manipulate global variables, remember that the server can call this function concurrently for different filter objects.)

The filter index function should perform the following tasks:

1. Get the filter object from the ***SLAPI_PLUGIN_OBJECT*** parameter (if the parameter is set).
2. Using data from the object, determine and set the values of the following parameters:
 - ***SLAPI_PLUGIN_MR_OID*** - Set to the official OID of the matching rule.
 - ***SLAPI_PLUGIN_MR_TYPE*** - Set to the attribute type in the filter object.
 - ***SLAPI_PLUGIN_MR_VALUES*** - Set to the values in the filter object.
 - ***SLAPI_PLUGIN_MR_QUERY_OPERATOR*** - Set to the query operator that corresponds to this search filter. Refer to [Section 11.4.2, “Query Operators in Matching Rules”](#), for possible values for this parameter.
 - ***SLAPI_PLUGIN_OBJECT*** - Set to the filter object.
 - ***SLAPI_PLUGIN_MR_INDEX_FN*** - Set to the indexer function. Refer to [Section 11.3.5, “Writing the Indexer Function”](#).
3. Return 0 (or the result code LDAP_SUCCESS) if everything completed successfully.

11.4.6. Getting and Setting Parameters in Filter Index Functions

The following table summarizes the different parameters that the filter index function should get and set in the parameter block that is passed in.

Table 11.5. Input and Output Parameters Available to a Filter Index Function

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_OBJECT</i>	void *	Input and Output parameter. Pointer to the filter object created by the factory function. For details, refer to Section 11.4.3, “Writing a Filter Factory Function” .
<i>SLAPI_PLUGIN_MR_QUERY_OPERATOR</i>	int	Output parameter. Query operator used by the server to determine how to compare the keys generated from <i>SLAPI_PLUGIN_MR_VALUES</i> and <i>SLAPI_PLUGIN_MR_INDEX_FN</i> against keys in the index. For a list of possible values for this parameter, refer to Section 11.4.2, “Query Operators in Matching Rules” .
<i>SLAPI_PLUGIN_MR_OID</i>	char *	Output parameter. Official matching rule OID (if any) specified in the extensible match filter.
<i>SLAPI_PLUGIN_MR_TYPE</i>	char *	Output parameter. Attribute type (if any) specified in the extensible match filter.
<i>SLAPI_PLUGIN_MR_VALUES</i>	struct berval **	Output parameter. Pointer to an array of berval structures containing the values specified in the extensible match filter.
<i>SLAPI_PLUGIN_MR_INDEX_FN</i>	void * (function pointer)	Output parameter. Name of the function called by the server to generate a list of keys used for indexing a set of values.

11.4.7. Writing a Filter Matching Function

The filter matching function has the following prototype:

```
#include "slapi-plugin.h"
typedef int (*mrFilterMatchFn) (void* filter, Slapi_Entry* entry,
Slapi_Attr* attrs);
```

This function passes the following arguments:

- **filter** is a pointer to the filter object.
- **entry** is a pointer to the [Section 14.22, “Slapi_Entry”](#) entry that should be compared against the filter.
- **attrs** is the first [Slapi_Attr](#) attribute in the entry or in the set of DN attributes. (The extensible match filter might specify that the attributes in the DN of an entry should also be included in the search.)

This function will never be called for the same filter object concurrently. (If you plan to manipulate global variables, remember that the server can call this function concurrently for different filter objects.)

The filter matching function should perform the following tasks:

1. From the filter object, get the attribute type, the values, and the query operator.
2. Find the corresponding attribute in the attributes passed into the function. Remember to search for subtypes of an attribute (for example, **cn=lang-ja**) in the filter and in the attributes specified by **attrs**.

You can call the [slapi_attr_type_cmp\(\)](#) function to compare the attribute in the filter against the attributes passed in as arguments.

3. Using the query operator to determine how the values should be compared, compare the values from the filter against the values in the attribute.
4. Return one of the following values:
 - 0 if the values of the attribute match the value specified in the filter.
 - -1 if the values do not match.
 - An LDAP error code (a positive number) if an error occurred.

11.5. HANDLING SORTING BY MATCHING RULES

If you have set up indexing by a matching rule, you can also sort search results by that matching rule. The server can use the keys in the index to sort the search results.

When processing a request to sort by a matching rule, the server does the following:

1. In a new [Slapi_PBlock](#) parameter block, the server sets the following parameters:
 - Sets the OID in the **SLAPI_PLUGIN_MR_OID** parameter.
 - Sets the attribute type (of the value being added, modified, or removed) in the **SLAPI_PLUGIN_MR_TYPE** parameter.
 - Sets the **SLAPI_PLUGIN_MR_USAGE** parameter to **SLAPI_PLUGIN_MR_USAGE_SORT**. (This indicates that the created indexer object will be used for sorting, not indexing.)
2. The server then calls the indexer factory function (specified in the **SLAPI_PLUGIN_MR_INDEXER_CREATE_FN** parameter) for the plug-in.
3. The server generates the index keys for the values to be sorted:

- The server sets the ***SLAPI_PLUGIN_MR_VALUES*** parameter to the array of berval structures containing the values to be sorted.
 - The server calls the indexer function (specified by the ***SLAPI_PLUGIN_MR_INDEXER_FN*** parameter).
 - The server then gets the value of the ***SLAPI_PLUGIN_MR_KEYS*** parameter, which is an array of berval structures containing the keys corresponding to the values.
4. The server compares the keys to sort the results.

11.6. WRITING A DESTRUCTOR FUNCTION

The server calls the destructor function to free any memory that was allocated; for example, to the indexer object or the filter object.

The destructor function takes a single [Slapi_PBlock](#) argument. The following table summarizes the different parameters that the destructor function should get and set in the parameter block that is passed in.

Table 11.6. Input and Output Parameters Available to a Destructor Function

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_OBJECT</i>	void *	Input parameter. Pointer to the filter object or indexer object created by your factory function.

For example, your destructor function can get the indexer object from the ***SLAPI_PLUGIN_OBJECT*** parameter and free the object from memory.

This function will never be called for the same indexer or filter object concurrently. (If you plan to manipulate global variables, remember that the server can call this function concurrently for different filter or indexer objects.)

11.7. WRITING AN INITIALIZATION FUNCTION

Internally, the server keeps a list of matching rule plug-ins. When dealing with matching rules, the server attempts to find the matching rule plug-in to handle the given matching rule. Refer to [Section 11.2.3, “How the Server Associates Plug-ins with OIDs”](#), for details.

In order to add your plug-in to that internal list, you need to write an initialization function. The initialization function takes a single [Slapi_PBlock](#) argument. The function should set the following parameters:

- The ***SLAPI_PLUGIN_MR_FILTER_CREATE_FN*** parameter should be set to the filter factory function.

Refer to [Section 11.4.1, “How the Server Handles the Filter”](#), and [Section 11.4.3, “Writing a Filter Factory Function”](#), for details.

- The ***SLAPI_PLUGIN_MR_INDEXER_CREATE_FN*** parameter should be set to the indexer factory function if you have defined one. [Optional]

Refer to [Section 11.3.1, “How the Server Sets Up the Index”](#), and [Section 11.3.3, “Writing the Indexer Factory Function”](#), for details.

- The ***SLAPI_PLUGIN_CLOSE_FN*** parameter should be set to the close function if you have defined one. [Optional]

Refer to [Section 11.9, “Specifying Start and Close Functions”](#), for details.

- The ***SLAPI_PLUGIN_PRIVATE*** parameter should be set to any private data you want made accessible to the plug-in functions. [Optional]

You need to register the initialization function so that the server runs the function when starting. For how to register matching rule functions, refer to [Section 11.8, “Registering Matching Rule Functions”](#).

The following table summarizes the different parameters that the initialization function should get and set in the parameter block that is passed in.

Table 11.7. Input and Output Parameters Available to a Matching Rule Plug-in Initialization Function

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_ARGC</i>	int	Input parameter. Number of arguments in the plugin directive, not including the library name and initialization function name.
<i>SLAPI_PLUGIN_ARGV</i>	char **	Input parameter. Array of string arguments in the plugin directive, not including the library name and initialization function name.
<i>SLAPI_PLUGIN_MR_FILTER_CREATE_FN</i>	void * (function pointer)	Output parameter. The factory function used for creating filters.
<i>SLAPI_PLUGIN_MR_INDEXER_CREATE_FN</i>	void * (function pointer)	Output parameter. The factory function used for creating indexers.
<i>SLAPI_PLUGIN_CLOSE_FN</i>	void * (function pointer)	Output parameter. The close function, which the server calls before shutting down.
<i>SLAPI_PLUGIN_PRIVATE</i>	void *	Output parameter. Pointer to any private data you want passed to your plug-in functions.

11.8. REGISTERING MATCHING RULE FUNCTIONS

Depending on the Directory Server version, add the appropriate information for your plug-in function.

In current releases of Directory Server, shut down the server, add the plug-in parameters to the **dse.ldif** file, and restart the server. Refer to [Chapter 3, Configuring Plug-ins](#).

For example, your plug-in entry might appear as follows:

```
dn: cn=Test MatchingRule,cn=plugins,cn=config
objectClass: top
objectClass: nsSlapdPlugin
objectClass: extensibleObject
cn: Test MatchingRule
nsslapd-pluginPath: libtest-plugin
nsslapd-pluginInitfunc: testmatchrule_init
nsslapd-pluginType: matchingRule
nsslapd-pluginEnabled: on
nsslapd-pluginId: test-matchingrule
nsslapd-pluginarg0: /usr/lib64/dirsrv/slapd-instance/
customplugins/filename.conf
```

11.9. SPECIFYING START AND CLOSE FUNCTIONS

For each matching rule operation plug-in, you can specify the name of a function to be called after the plug-in starts and before the plug-in is stopped or disabled. These functions take a single [Slapi_PBlock](#) argument.

The following table summarizes the different parameters that the initialization function should get and set in the parameter block that is passed in.

Table 11.8. Input and Output Parameters Available to a Matching Rule Plug-in Initialization Function

Parameter Name	Data Type	Description
<i>SLAPI_PLUGIN_START_FN</i>	void * (function pointer)	Output parameter. The function called after the plug-in starts.
<i>SLAPI_PLUGIN_CLOSE_FN</i>	void * (function pointer)	Output parameter. The function called before the plug-in is stopped or disabled.

If you register multiple plug-ins with different start and close functions, the functions are called in the order that the plug-ins are registered; in other words, in the order that the plugin directives appear in the server configuration file.

CHAPTER 12. USING THE CUSTOM DISTRIBUTION LOGIC

The distribution plug-in provided with Red Hat Directory Server distributes a flat namespace, allowing you to associate several databases with a single suffix.

12.1. ABOUT DISTRIBUTING FLAT NAMESPACES

You can distribute entries located in a flat tree structure. Imagine you administer the directory for **example.com**, an ISP. The directory for **example.com** contains the following entries distributed in a flat tree structure:

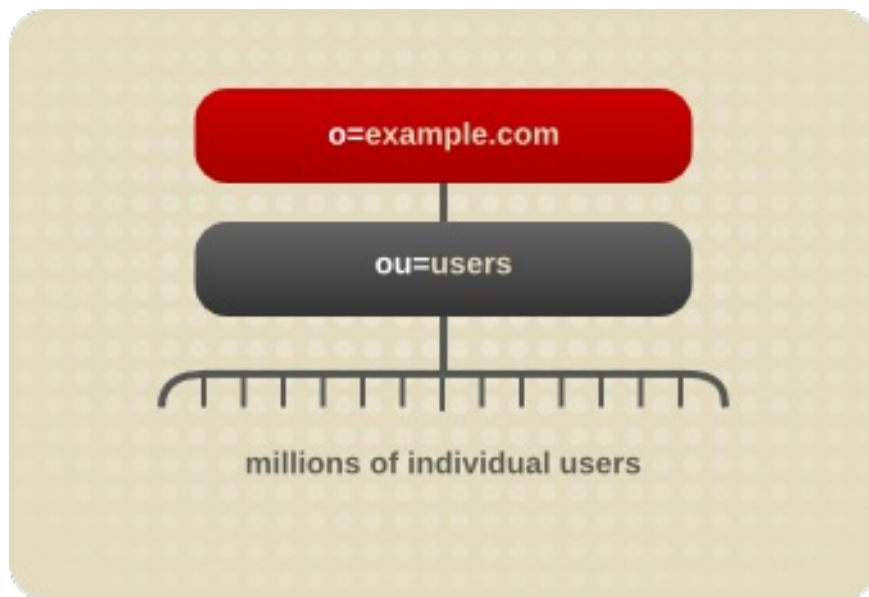


Figure 12.1.

Because the number of users is too large, you decide to distribute the entries according to the first letter of each user's last name, assigning each letter to a separate database. To do this, you need to create your own distribution function. Your function determines how each operation received by the **ou=users** suffix is routed to the database containing the information for a particular user.

After you have distributed entries, you cannot redistribute them. The following restrictions apply:

- You cannot change your distribution function after you have deployed entry distribution.
- You cannot use the **ldapmodrdn** operation to rename entries if the modification would cause them to be distributed into a different database.
- You cannot use the **ldapmodify** operation to change entries if that would cause them to be distributed into a different database.

For example, if you distribute entries according to their telephone number, you cannot change the telephone number attribute of an entry without breaking entry distribution.

- After you have deployed entry distribution you cannot add more databases.

12.2. CREATING A DISTRIBUTION FUNCTION

Using the distribution function, you can set a suffix to have any number of database pointers. This allows you to distribute requests made to a suffix over a number of databases.

For example, the entry for the **dc=example,dc=com** suffix appears as follows:

```
dn: cn=dc=example,dc=com,cn=mapping tree,cn=config
objectclass: top
objectclass: extensibleObject
nsslapd-backend: example.com database A-M
nsslapd-backend: example.com database N-Z
```

When the suffix receives a request from a client, it uses the distribution function to determine which database services the operation. The plug-in consists of a function and a library. This function and library are part of the entry for the **dc=example,dc=com** suffix and appear as follows:

```
nsslapd-distribution-plugin: /path/to/a/shared/library
nsslapd-distribution-funct: distribution-function-name
```

Each time the server gets a request for a suffix that contains the distribution function, the function is called. The function then decides which database (backend) processes the request. The decision made by the function is based on some piece of information present in the pblock at the time of the request, such as the entry DN, a hash function, the time of day, or the type of operation.

For search operations, the function can forward the operation to a single database, or to all databases.

The distribution function itself appears as follows:

```
int distribution_function(Slapi_PBlock *pb, Slapi_DN * dn, Slapi_Backend
**mtn_be, int be_count, Slapi_DN * node_dn);
```

The following table lists the parameters taken by the distribution function:

Table 12.1. Distribution Function Parameters

Parameter Name	Description
<i>pb</i>	The pblock of the plug-in which contains all of the information about the operations in progress. This structure is the same as that for other plug-ins; for example, operation type, IP address of the client, target DN, search filter, bind credentials, etc.
<i>mtn_be</i>	A table of size be_count containing the list of databases defined for the suffix. If a database is defined in the suffix entry but is not available, the corresponding pointer in the table will be set to NULL.

Parameter Name	Description
<i>node_dn</i>	The suffix containing the distribution function.
<i>be_count</i>	The size of the table that contains the list of databases involved in the distribution of entries.
<i>int</i>	The row number of the database in the <i>mtn_be</i> table. For search operations, you can return a value of MTN_ALL_BACKENDS to specify that all databases must be searched.

The root entry for the suffix must be present in each database in order for the distribution function to work properly. For example, for the suffix **dc=example,dc=com**, you need to create an entry corresponding to **dc=example,dc=com** in each database involved in the distribution.

You can create the root entry in two ways:

- Import the same LDIF file into each database using the **ldif2db** command-line utility.

This LDIF file should contain the root entry as well as data that you want to distribute across the databases. Only the data determined by the distribution function to be appropriate for each database will be imported.

- Create an LDIF file that contains the root entry. You can do this by exporting the root entry (for example, **dc=example,dc=com**) in LDIF format using the **ldapsearch** command-line utility.

You then need to import the LDIF file into each database using the **ldif2db** command-line utility. If you have three databases for a single suffix, you need to import the suffix entry three times.

When a subtree search is performed above a suffix that is distributed among several databases, the search will be performed on all databases. This means that the shared suffix entry that you create on each database (in the example, **dc=example,dc=com**) can be returned multiple times.

12.3. ADDING THE DISTRIBUTION FUNCTION TO YOUR DIRECTORY

To declare the distribution function to your directory, you need to add multiple databases to a single suffix and then declare the distribution function in the suffix.

These procedures assume you have already created a suffix and associated it with a single database. For the procedure on creating a new suffix and database, refer to the *Creating Directory Entries* chapter in the *Red Hat Directory Server Administration Guide*.



NOTE

You need to add all of the databases required for your distribution algorithm before you deploy entry distribution. You cannot add more databases later without changing the algorithm. Whenever you install new distribution functions, ensure that you restart the Directory Server. Otherwise, the new libraries will not be loaded properly.

12.3.1. Adding Multiple Databases to a Single Suffix

12.3.1.1. Using the Console

The following procedure describes how to add multiple databases to a suffix using the Console.

1. In the Directory Server Console, select the **Configuration** tab.
2. Expand the **Data** tree, and select the suffix to which you want to add another database.
3. From the **Object** menu, select **New Database**. You can also right click the suffix and select **New Database** from the menu.

The **Create New Database** dialog box appears.

4. Enter the name of the new database in the **Database Name** field.
5. In the **Create database in** field, enter the path to the location where the data for your new database will be stored.

You can also click **Browse** to locate the path on your local machine.

6. Click **OK** to save your changes.

A warning message displays telling you to declare a distribution function in the suffix. The next procedure describes how to declare the function in a suffix.

12.3.1.2. Using the Command-Line

The following procedure describes how to add multiple databases to a suffix using the command-line.

1. Use the **ldapmodify** command-line utility to add another database to your directory configuration file. The database configuration information is stored in the **cn=ldb database,cn=plugins,cn=config** entry.

For example, to add a new database to the server **example1**, you add a new entry to the configuration file by performing an **ldapmodify** as follows:

```
# ldapmodify -a -D "cn=Directory Manager" -W -p 389 -h
server.example.com -x
```

The **ldapmodify** utility binds to the server and prepares it to add an entry to the configuration file.

2. Create the entry for the new database as follows:

```
dn: cn=Data2,cn=ldbm database,cn=plugins,cn=config
objectclass: extensibleObject
objectclass: nsBackendInstance
nsslapd-suffix: ou=people,dc=example,dc=com
```

The entry added corresponds to a database named **Data2** that contains the data for the root suffix **ou=people,dc=example,dc=com**.

The database name, given in the DN attribute, must correspond with one of the values in the **nsslapd-backend** attribute of the suffix entry.

12.3.2. Adding Distribution Logic to a Suffix

The distribution logic is a function declared in a suffix. This function is called for every operation reaching this suffix, including the subtree search operations that start above the suffix. You can add a distribution function to a suffix using both the Console and the command-line.

12.3.2.1. Using the Console

To use the Console to add a distribution function to a suffix:

1. In the Directory Server Console, select the **Configuration** tab.
2. Expand the **Data** tree, and select the suffix to which you want to add the distribution function.
3. Select the **Databases** tab in the right pane.
4. Click **Add** to add new databases to the suffix from the Database List.
5. Enter the path to the distribution library in the **Distribution Library** field, or click **Browse** to locate the library on your local machine.
6. Enter the name of your distribution function in the **Function Name** field.
7. Click **Save** to save your changes.

12.3.2.2. Using the Command-Line

To use the Command-line to add a distribution function to a suffix:

Use the **ldapmodify** command-line utility to add the following lines to the suffix entry:

```
nsslapd-distribution-plugin: path_to_shared_library
nsslapd-distribution-funct: distribution_function_name
```

The first line provides the name of the library that your plug-in uses. The second line provides the name of the distribution function itself.

12.4. USING THE DISTRIBUTION LOGIC EXAMPLES

The directory provides three distribution logic examples. The examples illustrate the following:

- Distributing entries based on the first letter of their RDN (**alpha_distribution**).

The example uses as many databases as you like to contain the data. For example, if you create three databases for a single suffix, entries starting with the letters A-I go to database 0, entries starting with the letters J-R go to database 1, and entries starting with the letters S-Z go to database 2. If you create 26 databases, each database would receive the entries for one letter of the alphabet.

- Distributing entries based on a simple hash algorithm (**hash_distribution**).

In this example, entries are randomly distributed using a hash algorithm on the RDN to compute the database to which the entry will be written.

- Chaining entries to a read-write replica from a read-only replica (**chaining_distribution**).

Usually the directory returns a referral to clients making update requests to a read-only replica. This example uses a distribution function on a suffix that contains both a read-only database and a database link. When the read-only database receives an update request, it forwards the request using the database link to a read-write database. The database link needs to be configured to chain on update.

For information on configuring database links, refer to the *Creating Directory Entries* chapter in the *Red Hat Directory Server Administration Guide*.

The following directory contains the uncompiled C code examples:

/usr/lib64/dirsrv/plugins/

This directory contains the **distrib.c** file, which contains three example functions (**alpha_distribution**, **hash_distribution**, and **chaining_distribution**) and a Makefile for compiling them.

After you have compiled the source code, there is a **distrib-plugin.so**.

For example, to use the hash distribution function:

1. Create a suffix.
2. Create several databases under that suffix.
3. Import the suffix entry to each of the databases you created.
4. Add the following lines to the suffix:

```
nsslapd-distribution-plugin: /plugin/distrib-plugin.so
nsslapd-distribution-funct: hash_distribution
```

12.5. CUSTOM DISTRIBUTION CHECKLIST

In summary, the following steps are involved in adding the distribution function to your directory:

- Create the distribution function.
- Create a suffix.
- Add as many databases to the suffix as required by your distribution algorithm.
- Declare the distribution function in the suffix. You must specify the library path and the function name.
- Import data into the databases. If you do not import data, you need to import the root entry to each database.

CHAPTER 13. USING DATA INTEROPERABILITY PLUG-INS

This chapter explains how to use the Data Interoperability (DIOP) feature of Red Hat Directory Server (Directory Server). The DIOP feature refers to Directory Server's ability to work with a proprietary database, instead of the default database created during installation.

You can now use the enhanced pre-operation interfaces to implement plug-ins that are designed to provide access to alternative directory data stores, instead of the database backend plug-in interface, which is not supported in current releases. You do this by developing a custom pre-operation plug-in to provide an alternate functionality for the LDAP operations, such as search, modify, add, and so on. These operations are generally targeted at the root suffix or the null DN (meaning **dn:**), and your plug-in will have to be designed to intercept these operations and divert them to be serviced by an alternate data source or alternate access methods.

This chapter covers deployment considerations, configuration changes required to use the DIOP feature, a list of supported and unsupported features, and other useful information.

13.1. INSTALLING DIRECTORY SERVER

This section explains how to install Directory Server in order to test and use the DIOP feature.

13.1.1. Understanding Deployment Configuration

To verify whether the DIOP feature works in Directory Server, your deployment must comprise two instances of Directory Server:

- An instance of Directory Server that will be used for storing configuration data. This instance is identified as the **configurationDirectory Server**.
- An instance of Directory Server that will be used for enabling the DIOP plug-in. This instance is identified as the **DIOP-enabledDirectory Server**.

For detailed information on directory deployments, check the *Red Hat Directory Server Deployment Guide*. To understand the role of a configuration Directory Server in a directory deployment, check *Managing Servers with Red Hat Console*.

Because the DIOP plug-in is a pre-operation plug-in, enabling the plug-in will impose certain limitations on the default behavior of Directory Server.

- The Directory Server Console will not be fully functional in the DIOP-enabled Directory Server, and you will not be able to administer the server via the Console. However, you will be able to use the configuration Directory Server Console to manage the DIOP-enabled Directory Server.
- Some of the default plug-ins that are provided with the server will not work in the DIOP-enabled Directory Server. The DIOP plug-in is a pre-operation plug-in, and intercepting all LDAP operations will result in the other plug-ins being unusable. [Table 13.1, “Plug-in Status in DIOP-Enabled Directory Server”](#) identifies plug-ins that are unsupported in the DIOP-enabled Directory Server. All unsupported plug-ins must be disabled before using the DIOP plug-in.

Table 13.1. Plug-in Status in DIOP-Enabled Directory Server

Default Red Hat Directory Server Plug-in Guide (Names as they appear in the Directory Server Console)	Unsupported Plug-ins (Indicated by X)
7-bit check	X
ACL	-
ACL preoperation	-
Binary Syntax	-
Boolean Syntax	-
Case Exact String Syntax	-
Case Ignore String Syntax	-
chaining database	X
Class of Service	X
Country String Syntax	-
Distinguished Name Syntax	-
Generalized Time Syntax	-
HTTP Client	-
Integer Syntax	-
Internationalization Plugin	-
JPEG Syntax	-
ldbm database	-
Legacy Replication	X
Multimaster Replication	X
Octet String Syntax	-
OID Syntax	-

Default Red Hat Directory Server Plug-in Guide (Names as they appear in the Directory Server Console)	Unsupported Plug-ins (Indicated by X)
Pass-through Authentication	X
Postal Address Syntax	-
Referential Integrity Postoperation	X
Retro Changelog	X
Roles	X
Space Insensitive Syntax	-
State Change	X
Telephone Syntax	-
UID Uniqueness	X
URI Syntax	-
Views	X
CLEAR	-
CRYPT	-
DES	-
NS-MTA-MD5	-
SHA	-
SSHA	-

The figure below illustrates Directory Server deployment required for testing the DIOP feature.

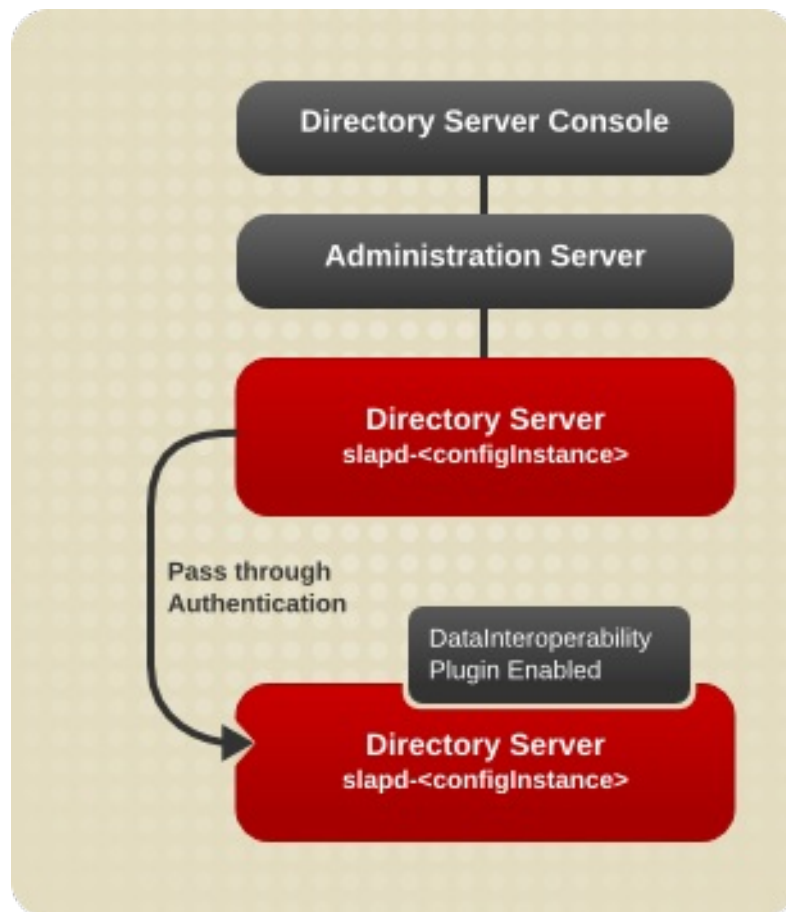


Figure 13.1. A typical Directory Server Deployment

In the above figure, **slapd-configuration** is the configuration Directory Server and **slapd-dio** is the Directory Server instance with the DIOP plug-in turned on.

- The management and administration of **slapd-configuration** is done via the corresponding Directory Server Console, accessible from within Red Hat Console.
- The management and administration of **slapd-dio** is done via the Directory Server Console of the **slapd-configuration** instance. This is because the **slapd-dio** instance does not support the full functionality of Red Hat Console.

To summarize the installation requirements for testing the DIOP feature:

- You install two instances of Directory Server under the same server root (by specifying the same installation directory). For example, you can install two Directory Server instances:

```
/usr/lib64/dirsrv/slapd-configuration
```

```
/usr/lib64/dirsrv/slapd-dio
```

where **/usr/lib64/dirsrv/slapd-configuration** is the default installation directory. In the sections that follow, the installation directory is identified as **/usr/lib64/dirsrv/slapd-configuration**.

- After you install the two instances, you designate the second Directory Server instance (**slapd-dio**) for testing the DIOP feature.

- You manage the first Directory Server instance (**slapd-*<configInstance>***) using Red Hat Console and the corresponding Administration Server, which is running under the same server root.
- You can indirectly manage the second Directory Server instance (**slapd-*<diopInstance>***) through the first Directory Server instance (**slapd-*<configInstance>***).
- You disable the unsupported plug-ins in the second Directory Server instance (**slapd-*<diopInstance>***).

For general information about installing Directory Server, refer to the accompanying *Red Hat Directory Server Installation Guide*. You can also find the documentation for Directory Server at <https://access.redhat.com/documentation/en/>

13.1.2. Installing Two Instances of Directory Server

Follow these instructions to create two instances of Directory Server:

1. Read the installation-specific documents (*Red Hat Directory Server Installation Guide* and *Release Notes*), and verify that your system meets the requirements specified in the documentation. Ensure that all patches are installed.
2. Unset the environment variable: **unsetenv LD_LIBRARY_PATH**
3. Unpack the binaries.
4. Run the setup program, and install an instance of Directory Server: **slapd-*<configInstance>***.
5. Start the Administration Server:

```
# systemctl start dirsrv-admin.service
```

6. Start the Directory Server Console:

```
# redhat-idm-console
```

7. Use the Directory Server Console to create a second instance of Directory Server, **slapd-*diopInstance***.

In the navigation pane, select the **Server Group**, right click, select **Create Instance of Red Hat Directory Server**, and follow the prompts.

8. Disable the unsupported plug-ins in the second instance (**slapd-*diopInstance***), which you will use for enabling the DIOP plug-in.
 1. In Red Hat Console, locate and double-click the entry for the second instance of Directory Server.

This opens the Directory Server Console for the second instance.

2. Select the **Configuration** tab, and expand **Plugins**.
3. Disable each of these plug-ins listed in [Table 13.1, “Plug-in Status in DIOP-Enabled Directory Server”](#).

To disable a plug-in, select the plug-in, and then, on the right panel, uncheck the **Enable the Plug-in** option. Some plug-ins may depend on other plug-ins, and you may see messages that reflect such a dependency.

9. Use the second instance to enable the DIOP feature, which is explained in the next section.

13.2. ENABLING THE DIOP FEATURE IN DIRECTORY SERVER

For a Directory Server instance to support the DIOP feature, its default configuration needs to be modified - the server needs to be configured to use the null DN or the root suffix in the server. This can be done by adding the following entry to the **dse.ldif** file of the server instance in which you want to enable the DIOP feature:

```
dn: cn=,cn=mapping tree,cn=config objectClass: top objectClass:
extensibleObject objectClass: nsMappingTree cn: nsslapd-state: container
```

You can modify the **dse.ldif** in either of the following ways:

1. By editing the **dse.ldif** file directly:

1. Shut down your Directory Server.

```
# systemctl stop dirsrv.target
```

2. In the text editor, open the **dse.ldif** file.

The file is located in the **/etc/dirsrv/slapd-*instance*** directory.

3. Add the above-mentioned entry, save your changes, and close the file.
4. Restart the server.

```
# systemctl start dirsrv.target
```

2. By using the **ldapmodify** command.

1. You can also add the above entry by running the **ldapmodify** command on the **slapd-*<diopInstance>*** server with the LDIF input file containing the above entry. For example, your command might look like this:

```
# ldapmodify -h <host> -p <port> -W -D cn=Directory Manager -vcaf
<ldif_file_containing_the_entry>
```

Once you add the above entry to the server configuration, the DIOP functionality is enabled in the server.



NOTE

An example plug-in is shipped with the server to show how a DIOP pre-operation plug-in can be used to work with the DIOP-enabled server. For details about the sample plug-in, see [Section 13.4, “Sample DIOP Plug-in”](#). To use the sample plug-in or your own custom plug-in in the server, see [Section 13.3, “Using the DIOP Feature”](#).

13.3. USING THE DIOP FEATURE

This section explains how you can verify whether the DIOP feature works in Directory Server. After you have successfully installed and configured two instances of Directory Servers, as explained in the preceding sections, follow the instructions in this section.

To help you understand the DIOP feature, a sample plug-in is provided. Details of this plug-in is covered in [Section 13.3, “Using the DIOP Feature”](#). It is recommended that you review the sample plug-in first and use that as an example to develop your own plug-in.

To verify the DIOP feature, you may use the sample plug-in or reconfigure Directory Server to use your own plug-in. You can also remove the plug-in altogether from the server.

1. If you want to use the sample plug-in, first build the plug-in, and then load it into the server:

1. Shut down your DIOP-enabled Directory Server.
2. Go to the directory in which the sample plug-in is located.

```
# cd /usr/lib64/dirsrv/plugins/
```

3. Build the plug-in.

```
# gmake
```

4. Modify the **/etc/dirsrv/slapd-*instance*/dse.ldif** file to include an entry for the plug-in. For instructions on to modify the **dse.ldif** file, refer to [Section 13.3, “Using the DIOP Feature”](#). The entry shown below is for the sample plug-in.

```
dn: cn=datainterop,cn=plugins,cn=config objectClass: top
objectClass: nsSlapdPlugin
cn: datainterop nsslapd-pluginPath:
/usr/lib64/dirsrv/plugins/libtest-plugin.so
nsslapd-pluginInitfunc: nullsuffix_init nsslapd-pluginType:
preoperation nsslapd-pluginEnabled: on
nsslapd-pluginId: nullsuffix-preop nsslapd-pluginVersion: 7.1
nsslapd-pluginVendor: Red Hat, Inc.
nsslapd-pluginDescription: sample pre-operation null suffix
plugin
```

5. Restart the server to load the modified configuration.

2. If you want to reconfigure the server to use your own plug-in:

1. Shut down your DIOP-enabled Directory Server.
2. Open the **/etc/dirsrv/slapd-*instance*/dse.ldif** file in a text editor.
3. Modify the **cn=datainterop,cn=plugins,cn=config** entry, which holds the plug-in information, with data from your proprietary database plug-in.
4. After you have done the required changes, restart the server to load the modified configuration.

3. If you want to delete the sample plug-in from the server:
 1. Shut down your DIOP-enabled Directory Server.
 2. Open the `/etc/dirsrv/slapd-instance/dse.ldif` file in a text editor.
 3. Delete the `cn=datapinterop,cn=plugins,cn=config` entry, which holds the plug-in information.
 4. Restart the server to load the modified configuration.

You can also use the `ldapmodify` command to make these changes.

13.4. SAMPLE DIOP PLUG-IN

To help you understand the DIOP feature, a sample DIOP plug-in is included with Directory Server. This section provides an overview of the sample plug-in and explains how you can use the plug-in to verify whether the DIOP feature works in Directory Server. The preceding section, [Section 13.3, “Using the DIOP Feature”](#), explains how you can use the sample plug-in. The next section, [Section 13.4.1, “Debugging the Plug-in”](#), explains how to troubleshoot the plug-in.

The sample plug-in is located in the `/usr/lib64/dirsrv/plugins/` directory. The shared library for the plug-in is named `libtest-plugin.so` and is implemented by

```
testdatapinterop.c[.h]
testdbinterop.c
db.h
```

located in the same directory.

Note the following:

- The main goal of the sample plug-in is to show how to create a simple plug-in that supports data interoperability.
- The plug-in does not attempt to create any usable functionality to access backends (database or files) but returns observable output uniformly to verify that the functions in the pre-operation plug-in have been accessed and executed for different LDAP operations.
- The plug-in demonstrates the use of APIs, which meet the requirements of the DIOP feature.

In the following table, the various required elements of the pre-operation plug-in are identified by the function calls used in the `testdatapinterop.c` (to illustrate the use and simplify understanding).

Table 13.2. Elements of Pre-Operation Plug-in

Element	Description
---------	-------------

Element	Description
Description of the Plug-in	<pre>#define PLUGIN_NAME "nullsuffix- preop" static Slapi_PluginDesc plugindesc = { PLUGIN_NAME, Red Hat, 7.1, sample pre-operation null suffix plugin }x</pre>
Initialization of the Plug-in by the Server	<p>nullsuffix_init(Slapi_PBlock *pb)</p> <p>In this function, all the callbacks are set up and will be called by the server for each LDAP operation.</p>
<p>Reserved Naming Contexts</p> <p>(cn=schema, cn=config, cn=monitor)</p>	<p>slapi_op_reserved() is called to determine whether the operation should be handled internally by Directory Server; for example, whether the base on which the operation is applied is a reserved naming context. If returns a non-zero value, the plug-in does not attempt to handle that operation. This is performed by the following code snippet:</p> <pre>if(slapi_op_reserved(pb)) { return PLUGIN_OPERATION_IGNORED; }</pre> <p>Refer to testdatainterop.c for details.</p> <p>The slapi_op_reserved() function, which can be used for reserving some of the naming contexts in the Directory Server (cn=schema, cn=config, cn=monitor), is called first in the database plug-in and then the call for turning off access control.</p>
Sparse Tree Support	<p>Any modifications done to the server on the null suffix are processed by the plug-in. The plug-in writes the DN of all modifications received to a standalone BerkleyDB, and trying a simple test using LDIF entries without the required object classes or parent entries will still get processed by the server, populating the database created by the plug-in. See nullsuffix_modify and testdbinterop.c for details. The plug-in has not been coded for the retrieval of those entries but has been coded to demonstrate sparse tree support only.</p>

Element	Description
Access Control	Switching off access control for the operation is done by: <code>slapi_operation_set_flag(op, SLAPI_OP_FLAG_NO_ACCESS_CHECK);</code> See testdatainterop.c for details.
Null Suffix Support	The plug-in cannot control the support for null-suffix in the server. The support for null-suffix is done through configuration modification of the server as shown in Section 13.4, “Sample DIOP Plug-in” .
Building the Data Interoperability Plug-in	The compiler used on Solaris is Forte. For example: <code>cd /usr/lib64/dirsrv/plugins</code> <code>gmake libtest-plugin.so</code> is generated.

Flag used for LDAP operation	Callback
<code>SLAPI_PLUGIN_PRE_SEARCH_FN</code>	<code>nullsuffix_search</code>
<code>SLAPI_PLUGIN_PRE_ADD_FN</code>	<code>nullsuffix_add</code>
<code>SLAPI_PLUGIN_PRE_MODIFY_FN</code>	<code>nullsuffix_modify</code>
<code>SLAPI_PLUGIN_PRE_DELETE_FN</code>	<code>nullsuffix_delete</code>
<code>SLAPI_PLUGIN_PRE_BIND_FN</code>	<code>nullsuffix_bind</code>
<code>SLAPI_PLUGIN_PRE_MODRDN_FN</code>	<code>nullsuffix_modrdn</code>

13.4.1. Debugging the Plug-in

If you need to debug the plug-in installed on a Solaris machine, you can use **dbx**:

1. **`cd /usr/share/dirsrv/bin/slapd/server`**
2. **`setenv NETSITE_ROOT /usr/share/dirsrv`**
3. **`dbx ns-slapd`**
4. **`run -d 65536 -D /usr/lib64/dirsrv/slapd-<diopInstance>`**
5. Once the server starts up and error logs show that the server has started, press **Ctrl + C**.
6. **`stop`** in *user-defined-function-in-the-plugin*

Similar steps can be done on other platforms, using the platform-specific debuggers and commands.

13.5. PLUG-IN API REFERENCE

This section contains reference information on APIs that enable the following:

- [Section 13.5.1, “Preserving the Default Behavior of the Server”](#)
- [Section 13.5.2, “Bypassing Access Control Checks”](#)

13.5.1. Preserving the Default Behavior of the Server

Directory Server implements internal backends for supporting subtrees **cn=config**, **cn=schema**, and **cn=monitor**, which are the reserved naming contexts for the server. For more information about these, check the *Red Hat Directory Server Configuration, Command, and File Reference*.

It may be required in some cases to let the default behavior of the server be preserved and not be intercepted by the custom pre-operation plug-ins. To implement a custom DIOP plug-in without affecting the default behavior of the Directory Server, a new function named **slapi_op_reserved()** is being made available. For details about this function, see [Chapter 49, Functions Related to Data Interoperability](#).

13.5.2. Bypassing Access Control Checks

It may be desirable to disable access control checking for operations that are handled by the custom DIOP plug-in. To enable the plug-ins to bypass access control, a new flag, **SLAPI_OP_FLAG_NO_ACCESS_CHECK**, has been defined. You allow a custom plug-in to bypass access control by setting the flag on the operation-data structure, which is available to the plug-in through the parameter (pblock) setting; see [Part V, “Parameter Block Reference”](#).

The following functions have been defined for this purpose:

- [slapi_operation_set_flag\(\)](#)
- [slapi_operation_clear_flag\(\)](#)
- [slapi_operation_is_flag_set\(\)](#)

For details about these functions, see [Chapter 49, Functions Related to Data Interoperability](#).

PART III. DATA TYPE AND STRUCTURE REFERENCE

CHAPTER 14. DATA TYPE AND STRUCTURE REFERENCE

This part summarizes the data types and structures that you can use when writing Red Hat Directory Server plug-in functions.

14.1. Berval

Represents binary data that is encoded using simplified Basic Encoding Rules (BER).

Syntax

This struct definition uses the following syntax:

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

Fields

This struct definition contains the following fields:

Table 14.1. Berval Field Listing

Field	Description
<i>bv_len</i>	The length of the data.
<i>bv_val</i>	The binary data.

Description

The berval data structure represents binary data that is encoded using simplified Basic Encoding Rules (BER). The data and size of the data are included in a berval structure.

Use a berval structure when working with attributes that contain binary data (such as a JPEG or audio file).

14.2. COMPUTED_ATTR_CONTEXT

Represents information used for a computed attribute.

Syntax

This struct definition uses the following syntax:

```
typedef struct _computed_attr_context computed_attr_context;
```

Description

computed_attr_context is the data type for an opaque structure that represents information about a computed attribute.

Before the Directory Server sends an entry back to a client, it determines if any of the attributes are computed, generates the attributes, and includes the generated attributes in the entry.

As part of this process, the server creates a `computed_attr_context` structure to pass relevant information to the functions generating the attribute values. Relevant information might include the attribute type, the BER-encoded request so far, and the parameter block.

14.3. LDAPCONTROL

Represents a client or server control associated with an LDAP operation.

The `LDAPControl` data type represents a client or server control associated with an LDAP operation. Controls are part of the LDAPv3 protocol. You can use a client or server control to extend the functionality of an LDAP control.

For example, a server control can specify that the server must sort search results in an LDAP search operation.

[Table 14.2, “Frontend API Functions for LDAP Controls”](#) summarizes the frontend API functions that can be called to work with LDAP controls.

Table 14.2. Frontend API Functions for LDAP Controls

To perform this action...	Call this function
Append a control to the end of an array or to a new array.	slapi_add_control_ext()
Append an array of controls to the end of an array or to a new array.	slapi_add_controls()
Create an <code>LDAPControl</code> structure based on a <code>BerElement</code> , an OID, and a criticality flag. It returns an LDAP error code.	slapi_build_control()
Create an <code>LDAPControl</code> structure based on a <code>struct berval</code> , an OID, and a criticality flag. It returns an LDAP error code.	slapi_build_control_from_berval()
Check for the presence of a specific <code>LDAPControl</code> . It returns non-zero for presence and zero for absence.	slapi_control_present()
Retrieve the <i>LDAPMod</i> contained in a <code>Slapi_Mod</code> structure.	slapi_mod_get_ldapmod_passout()
Register the specified control with the server. This function associates the control with an object identification (OID).	slapi_register_supported_control()
Retrieve an allocated array of object identifiers (OIDs) representing the controls supported by the Directory Server.	slapi_get_supported_controls_copy()

Syntax

This struct definition uses the following syntax:

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

Parameters

This function has the following parameters:

Table 14.3. LDAPControl Parameters

Field	Description
<i>ldctl_oid</i>	Object ID (OID) of the control.
<i>ldctl_value</i>	berval structure containing the value used by the control for the operation.
<i>ldctl_iscritical</i>	Specifies whether the control is critical to the operation. This field can have one of the following values: <ul style="list-style-type: none"> • LDAP_OPT_ON specifies that the control is critical to the operation. • LDAP_OPT_OFF specifies that the control is not critical to the operation.

14.4. LDAPMOD

Specifies changes to an attribute in a directory entry.

LDAPMod is a type of structure that specifies changes to an attribute in a directory entry. Before you call the **slapi_add_internal_pb()** and **slapi_modify_internal_pb()** routines 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.

The following section of code sets up an LDAPMod structure to change the email address of a user's entry to **bab@example.com**:

```
Slapi_PBlock *mod_pb = slapi_pblock_new();
LDAPMod attributel;
LDAPMod *list_of_attrs[2];
char *mail_values[] = { "bab@example.com" , NULL };
Slapi_DN *dn;
int ret = 0;

...
...
```

```
/* Identify the entry that you want changed */
dn = "cn=Barbara Jensen, ou=Product Development, l=US, 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 */
slapi_modify_internal_set_pb(mod_pb, config_entry->dn,
                             list_of_attrs, 0, 0, getPluginID(), 0);
slapi_modify_internal_pb(mod_pb);
slapi_pblock_get(mod_pb, SLAPI_PLUGIN_INTOP_RESULT, &ret);
slapi_pblock_destroy(mod_pb);

...
...
```

Table 14.4, “Frontend API Functions for Entry Attribute Changes” summarizes the functions available to specify changes to an attribute in a directory entry.

Table 14.4. Frontend API Functions for Entry Attribute Changes

To perform this action...	Call this function
Translate from entry to LDAPMod.	slapi_entry2mods()
Dump the contents of an LDAPMod to the server log.	slapi_mod_dump()
Get a reference to the LDAPMod in a Slapi_Mod structure.	slapi_mod_get_ldapmod_byref()
Retrieve the reference to the LDAPMod contained in a Slapi_Mod structure.	slapi_mod_get_ldapmod_passout()

Syntax

This struct definition uses the following syntax:

```
typedef struct ldapmod {
    int mod_op;
    char *mod_type;
    union mod_vals_u{
```

```

char **modv_strvals;
struct berval **modv_bvals;
} mod_vals;

#define mod_values mod_vals.modv_strvals
#define mod_bvalues mod_vals.modv_bvals
} LDAPMod;

```

Fields

This struct definition contains the following fields:

Table 14.5. ldapmod Field Listing

Field	Description
<i>mod_op</i>	<p>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:</p> <ul style="list-style-type: none"> <pre>#define LDAP_MOD_ADD 0x00</pre> <p>LDAP_MOD_ADD specifies to add the attribute values to the entry.</p> <pre>#define LDAP_MOD_DELETE 0x01</pre> <p>LDAP_MOD_DELETE specifies to remove the attribute values from the entry.</p> <pre>#define LDAP_MOD_REPLACE 0x02</pre> <p>LDAP_MOD_REPLACE specifies to replace the existing value of the attribute with the values in <i>mod_values</i> or <i>mod_bvalues</i>.</p> <pre>#define LDAP_MOD_BVALUES 0x80</pre> <p>In addition, if you are specifying binary values (as opposed to strings), you should OR () LDAP_MOD_BVALUES with the operation type. For example:</p> <pre>mod->mod_op = LDAP_MOD_ADD LDAP_MOD_BVALUES</pre>
<i>mod_type</i>	<p>Pointer to the attribute type that you want to add, delete, or replace.</p>

Field	Description
<i>mod_values_u</i>	A NULL-terminated array of string values for the attribute.
<i>modv_strvals</i>	Pointer to a NULL terminated array of string values for the attribute.
<i>mod_bvalues</i>	Pointer to a NULL-terminated array of berval structures for the attribute.
<i>mod_vals</i>	Values that you want to add, delete, or replace.

See Also

[Section 14.26, “Slapi_Mods”](#)

14.5. MRFILTERMATCHFN

mrFilterMatchFn specifies the prototype for a **filter_matching** function that is called by the server when processing an *extensible match* filter.

An extensible match filter specifies either the OID of a matching rule or an attribute type (or both) that indicates how matching entries are found. For example, a *sound-alike* matching rule might find all entries that sound like a given value.

To handle an extensible match filter for a matching rule, you can write a matching rule plug-in.

You need to define the filter matching function, which is the function that has prototype specified by **mrFilterMatchFn**. The server calls this function for each potential matching candidate entry. The server passes pointers to a filter structure that you create in your filter factory function, the candidate entry, and the entry's attributes.

In your filter matching function, you can retrieve information about the filter, such as the attribute type and value specified in the filter, from the filter structure. You can then use this information to compare the value in the filter against the attribute values in the candidate entry.

Syntax

```
#include "slapi-plugin.h"
typedef int (*mrFilterMatchFn) (void* filter, Slapi_Entry* entry,
Slapi_Attr* attrs);
```

Parameters

This function takes the following parameters:

Table 14.6. mrFilterMatchFn Parameter Listing

Parameter	Description
<i>filter</i>	Pointer to the filter structure created by your filter factory function. Refer to Section 11.4.3, “Writing a Filter Factory Function” for more information.
<i>entry</i>	Pointer to the <code>Slapi_Entry</code> structure representing the candidate entry being checked by the server.
<i>attrs</i>	Pointer to the <code>Slapi_Attr</code> structure representing the first attribute in the entry. To iterate through the rest of the attributes in the entry, call <code>slapi_entry_next_attr()</code> .

Returns

This function returns an integer value of **0** if the filter is matched or **-1** if the filter did not match. If an LDAP error occurs, it returns a value greater than **0**.

See Also

[Section 11.4.3, “Writing a Filter Factory Function”](#)

14.6. PLUGIN_REFERRAL_ENTRY_CALLBACK

This typedef is used for LDAP referral entry callback functions, which are plugin-defined functions that process LDAP references generated by some internal searches.

Syntax

```
#include "slapi-plugin.h"
typedef int (*plugin_referral_entry_callback)
(char *referral, void *callback_data);
```

Parameters

The function takes the following parameters:

Table 14.7. plugin_referral_entry_callback Parameter Listing

Parameter	Description
<i>referral</i>	The URL of a reference that is returned in response to an internal search call.
<i>callback_data</i>	This value matches the <i>callback_data</i> pointer that was passed to the original internal operation function.

Returns

The following table lists this function's possible return values.

Table 14.8. plugin_referral_entry_callback Return Values

Return Value	Description
0	Success
-1	An error occurred.

Description

A function that matches this typedef can be passed as the *prec* parameter of **slapi_search_internal_callback_pb()**, or as the *ref_callback* parameter of the **slapi_seq_internal_callback_pb()** function.

The LDAP referral entry callback function is called once for each referral entry found by a search operation, which means it could be called zero or any number of times.

The *callback_data* parameter can be used to pass arbitrary plug-in or operation-specific information to a referral entry callback function.

14.7. PLUGIN_RESULT_CALLBACK

This typedef is used for LDAP result callback functions, which are plugin-defined functions that process result messages that are generated by some internal search functions.

Syntax

This typedef uses the following syntax:

```
#include "slapi-plugin.h"
typedef void (*plugin_result_callback)(int rc, void *callback_data);
```

Parameters

This typedef takes the following parameters:

Table 14.9. plugin_result_callback Parameters

<i>rc</i>	The LDAP result code of the internal operation; for example, LDAP_SUCCESS.
<i>callback_data</i>	This value matches the <i>callback_data</i> pointer that was passed to the original internal operation function.

Returns

The following table lists this function's possible return values.

Table 14.10. plugin_result_callback Return Values

Return Value	Description
0	Success
-1	An error occurred.

Description

A function that matches this typedef can be passed as the *prc* parameter of `slapi_search_internal_callback_pb()` or as *theres_callback* parameter of `slapi_seq_internal_callback_pb()`.

The LDAP result callback function should be called once for each search operation, unless the search is abandoned, in which case it will not be called.

The *callback_data* parameter can be used to pass arbitrary plug-in or operation-specific information to a result callback function.

14.8. PLUGIN_SEARCH_ENTRY_CALLBACK

This typedef is used for LDAP search entry callback functions, which are plug-in defined functions that process LDAP entries that are located by an internal search.

Syntax

This typedef uses the following syntax:

```
#include "slapi-plugin.h"
typedef int (*plugin_search_entry_callback)(Slapi_Entry *e, void
*callback_data);
```

Parameters

This typedef takes the following parameters:

Table 14.11. plugin_search_entry_callback Parameters

<i>e</i>	Pointer to the <code>Slapi_Entry</code> structure representing an entry found by the search.
<i>callback_data</i>	This value matches the <i>callback_data</i> pointer that was passed to the original internal operation function.

Returns

The following table lists this function's possible return values.

Table 14.12. plugin_search_entry_callback Return Values

Return Value	Description
--------------	-------------

Return Value	Description
0	Success
-1	An error occurred.

Description

A function that matches this typedef can be passed as the *psec* parameter of **slapi_search_internal_callback_pb()** or as the *srch_callback* parameter of **slapi_seq_internal_callback_pb()**.

The LDAP referral entry callback function will be called once for each referral entry found by a search operation, which means it could be called zero or any number of times.

The *callback_data* parameter can be used to pass arbitrary plug-in or operation-specific information to a referral entry callback function.

14.9. SEND_LDAP_REFERRAL_FN_PTR_T

`send_ldap_referral_fn_ptr_t` specifies the prototype for a callback function that you can write to send LDAPv3 referrals (search result references) back to the client. You can register your function so that it is called whenever the **slapi_send_ldap_result()** function is called.

Syntax

```
#include "slapi-plugin.h"
typedef int (*send_ldap_referral_fn_ptr_t)( Slapi_PBlock *pb,
      Slapi_Entry *e, struct berval **refs, struct berval ***urls);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block.
<i>e</i>	Pointer to the <code>Slapi_Entry</code> structure representing the entry with which you are working.
<i>refs</i>	Pointer to the NULL-terminated array of <code>berval</code> structures containing the LDAPv3 referrals (search result references) found in the entry.
<i>urls</i>	Pointer to the array of <code>berval</code> structures used to collect LDAP referrals for LDAPv2 clients.

Returns

This function returns **0** if successful, or **-1** if an error occurs.

Description

The `slapi_send_ldap_result()` function is responsible for sending LDAPv3 referrals (search result references) back to the client. You can replace the function that sends LDAPv3 referrals to the client with your own function. To do this:

1. Write a function with the prototype specified by `send_ldap_result_fn_ptr_t`.
2. In your plug-in initialization function, register your function by setting the `SLAPI_PLUGIN_PRE_REFERRAL_FN` parameter in the parameter block to the name of your function if you are using the pre-operation plug-in. If you are using the post-operation plug-in, register your function by setting the `SLAPI_PLUGIN_POST_REFERRAL_FN` parameter in the parameter block to the name of your function.

See [slapi_send_ldap_result\(\)](#) for information on the default function that sends LDAPv3 referrals to clients.

See Also

`send_result()`

14.10. SEND_LDAP_RESULT_FN_PTR_T

`send_ldap_result_fn_ptr_t` specifies the prototype for a callback function that you can write to send LDAP result codes back to the client. You can register your function so that it is called whenever the `slapi_send_ldap_result()` function is called.

Syntax

```
#include "slapi-plugin.h"
typedef void (*send_ldap_result_fn_ptr_t)( Slapi_PBlock *pb,
    int err, char *matched, char *text, int nentries, struct berval **urls );
```

Parameters

The function has the following parameters:

<i>pb</i>	Parameter block.
<i>err</i>	LDAP result code that you want sent back to the client; for example, <code>LDAP_SUCCESS</code> .
<i>matched</i>	When sending back an <code>LDAP_NO_SUCH_OBJECT</code> result code, use this argument to specify the portion of the target DN that could be matched.
<i>text</i>	Error message that you want sent back to the client. Use <code>NULL</code> if you do not want an error message sent back.

<i>nentries</i>	When sending back the result code for an LDAP search operation, use this argument to specify the number of matching entries found.
<i>urls</i>	When sending back an LDAP_PARTIAL_RESULTS result code to an LDAPv2 client or an LDAP_REFERRAL result code to an LDAPv3 client, use this argument to specify the array of berval structures containing the referral URLs.

Description

The **slapi_send_ldap_result()** function is responsible for sending LDAP result codes back to the client. You can replace the function that sends LDAP result codes to the client with your own function. To do this:

1. Write a function with the prototype specified by **send_ldap_result_fn_ptr_t**.
2. In your plug-in initialization function, register your function for sending results to the client by setting the *SLAPI_PLUGIN_PRE_RESULT_FN* or *SLAPI_PLUGIN_POST_RESULT_FN* parameter, depending on the type of plug-in and if it is a pre-operation or post-operation, respectively, in the parameter block to the name of your function.

See Also

See [slapi_send_ldap_result\(\)](#) for information on the default function that sends LDAP result codes to clients.

14.11. SEND_LDAP_SEARCH_ENTRY_FN_PTR_T

send_ldap_result_fn_ptr_t specifies the prototype for a callback function that you can write to send search results (entries found by a search) back to the client. You can register your function so that it is called whenever the **slapi_send_ldap_search_entry()** function is called.

Syntax

```
#include "slapi-plugin.h"
typedef int (*send_ldap_search_entry_fn_ptr_t)
( Slapi_PBlock *pb, Slapi_Entry *e, LDAPControl **ectrls, char **attrs,
  int attrsonly );
```

Description

The **slapi_send_ldap_search_entry()** function is responsible for sending entries found by a search back to the client. You can replace the function that sends entries to the client with your own function. To do this:

1. Write a function with the prototype specified by **send_ldap_search_entry_fn_ptr_t**.
2. In your plug-in initialization function, register your function by setting the

`SLAPI_PLUGIN_PRE_ENTRY_FN` parameter in the parameter block to the name of your function if you are using the pre-operation plug-in. If you are using the post-operation plug-in, register your function by setting the `SLAPI_PLUGIN_POST_ENTRY_FN` parameter in the parameter block to the name of your function.

See Also

See [slapi_send_ldap_search_entry\(\)](#) for information on the default function that sends entries to clients.

14.12. SLAPI_ATTR

Represents an attribute in an entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_attr Slapi_Attr;
```

Description

`Slapi_Attr` is the data type for an opaque structure that represents an attribute in a directory entry. In certain cases, your server plug-in may need to work with an entry's attributes. The following table summarizes the front-end API functions that you can call to work with attributes.

To Call this function
Add an attribute value.	slapi_attr_add_value()
Return the base type of an attribute.	slapi_attr_basetype()
Duplicate an attribute.	slapi_attr_dup()
Get the first value of an attribute.	slapi_attr_first_value()
Determine if certain flags are set.	slapi_attr_flag_is_set()
Free an attribute.	slapi_attr_free()
Put the values contained in an attribute into an array of <code>berval</code> structures.	slapi_attr_get_bervals_copy()
Get the flags associated with an attribute.	slapi_attr_get_flags()
Put the count of values of an attribute into an integer.	slapi_attr_get_numvalues()

To Call this function
Search for an attribute type and give its OID string.	slapi_attr_get_oid_copy()
Get the type of an attribute.	slapi_attr_get_type()
Get the next value of an attribute.	slapi_attr_get_valueset()
Determine the next value of an attribute.	slapi_attr_next_value()
Initialize a valueset in a Slapi_Attr structure from a specified Slapi_ValueSet structure.	slapi_attr_set_valueset()
Get information about the plug-in responsible for handling an attribute type.	slapi_attr_type2plugin()
Compare two attribute names to determine if they represent the same attribute.	slapi_attr_types_equivalent()
Find the first attribute in an entry.	slapi_entry_first_attr()
Iterate through the attributes in an entry.	slapi_entry_next_attr()
Determine if an attribute contains a given value.	slapi_entry_attr_find()
Determine if an attribute has the specified value.	slapi_attr_value_find()
Compare two attribute values.	slapi_attr_value_cmp()
Add the changes in a modification to a valueset.	slapi_valueset_set_from_smod()
Initialize a Slapi_ValueSet structure from another Slapi_ValueSet structure.	slapi_valueset_set_valueset()

See Also

[Section 14.22, “Slapi_Entry”](#)

14.13. SLAPI_BACKEND

Represents a backend operation in the server plug-in-in.

Syntax

```
#include "slapi-plugin.h"
typedef struct backend Slapi_Backend;
```

Description

Slapi_Backend is the data type for an opaque structure that represents a backend operation. The following table summarizes the front-end API functions that you can call to work with the backend operations.

To...	... Call this function
Add the specified suffix to the given backend and increment the backend's suffix count.	slapi_be_addsuffix()
Set the flag to denote that the backend will be deleted on exiting.	slapi_be_delete_onexit()
Check if the backend that contains the specified DN exists.	slapi_be_exist()
Free memory and linked resources from the backend structure.	slapi_be_free()
Get the instance information of the specified backend.	slapi_be_get_instance_info()
Return the name of the specified backend.	slapi_be_get_name()
Indicate if the database associated with the backend is in read-only mode.	slapi_be_get_readonly()
Get pointer to a callback function that corresponds to the specified entry point into a given backend.	slapi_be_getentrypoint()
Return the n+1 suffix associated with the specified backend.	slapi_be_getsuffix()
Return the type of the backend.	slapi_be_gettype()
Check if a flag is set in the backend configuration.	slapi_be_is_flag_set()
Verify that the specified suffix matches a registered backend suffix.	slapi_be_issuffix()
Indicate if the changes applied to the backend should be logged in the changelog.	slapi_be_logchanges()

To...	... Call this function
Create a new backend structure, allocate memory for it, and initialize values for relevant parameters.	<code>slapi_be_new()</code>
Verify if the backend is private.	<code>slapi_be_private()</code>
Find the backend that should be used to service the entry with the specified DN.	<code>slapi_be_select()</code>
Find the backend that matches by the name of the backend. Backends can be identified by name and type.	<code>slapi_be_select_by_instance_name()</code>
Set the specified flag in the backend.	<code>slapi_be_set_flag()</code>
Set the instance information of the specified backend with given data.	<code>slapi_be_set_instance_info()</code>
Set a flag to denote that the backend is meant to be read-only.	<code>slapi_be_set_readonly()</code>
Set the entry point in the backend to the specified function.	<code>slapi_be_setentrypoint()</code>
Return a pointer to the backend structure of the first backend.	<code>slapi_get_first_backend()</code>
Return a pointer to the next backend, selected by index.	<code>slapi_get_next_backend()</code>
Return the first root suffix of the DIT.	<code>slapi_get_first_suffix()</code>
Return the DN of the next root suffix of the DIT.	<code>slapi_get_next_suffix()</code>
Check if a suffix is a root suffix of the DIT.	<code>slapi_is_root_suffix()</code>

14.14. SLAPI_BACKEND_STATE_CHANGE_FNPTR

`slapi_backend_state_change_fnptr` specifies the prototype for a callback function, which allows a plug-in to register for callback when a backend state changes.

Syntax


```
#include "slapi-plugin.h"
typedef void (*slapi_backend_state_change_fnptr)
(void *handle, char *be_name, int old_be_state, int new_be_state);
```

Parameters

The function has the following parameters:

<i>handle</i>	Pointer or reference to the address of the specified function.
<i>be_name</i>	Name of the backend.
<i>old_be_state</i>	Old backend state.
<i>new_be_state</i>	New backend state.

Description

The function enables a plug-in to register for callback when the state of a backend changes. You may need to keep track of backend state changes when writing custom plug-ins.

See Also

- [slapi_register_backend_state_change\(\)](#)
- [slapi_unregister_backend_state_change\(\)](#)

14.15. SLAPI_COMPONENTID

Represents the component ID in a directory entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_componentid Slapi_ComponentId;
```

Description

Slapi_ComponentID is the data type for an opaque structure that represents the component ID in a directory entry.

14.16. SLAPI_COMPUTE_CALLBACK_T

Represents a callback for evaluating computed attributes.

Syntax

```
#include "slapi-plugin.h"
typedef int (*slapi_compute_callback_t)
```

```
(computed_attr_context *c, char* type, Slapi_Entry *e,  
slapi_compute_output_t outputfn);
```

Parameters

The function has the following parameters:

<i>c</i>	Pointer to the <code>computed_attr_context</code> structure containing information relevant to the computed attribute.
<i>type</i>	Attribute type of the attribute to be generated.
<i>e</i>	Pointer to the <code>Slapi_Entry</code> structure representing the entry to be sent back to the client.
<i>outputfn</i>	Pointer to the <code>slapi_compute_output_t</code> function responsible for BER-encoding the computed attribute and for adding it to the BER element to be sent to the client.

Returns

One of the following values:

- -1 if the function is not responsible for generating the computed attribute.
- 0 if the function successfully generates the computed attribute.
- An LDAP error code if an error occurred.

Description

`slapi_compute_callback_t` specifies the prototype for a callback function that is called by the server when generating a computed attribute. If you want to use computed attributes, you should write a function of this type.

See Also

[Section 14.17, “slapi_compute_output_t”](#)

14.17. SLAPI_COMPUTE_OUTPUT_T

Represents a prototype for an output function for contributed attributes.

Syntax

```
#include "slapi-plugin.h"  
typedef int (*slapi_compute_output_t)  
    (computed_attr_context *c, Slapi_Attr *a, Slapi_Entry *e);
```

Parameters

The function has the following parameters:

Returns

One of the following values:

- 0 if the function successfully BER-encodes the computed attribute and adds it to the BER element to be sent to the client.
- An LDAP error code if an error occurred.

Description

slapi_compute_output_t specifies the prototype for a callback function that BER-encodes a computed attribute and appends it to the BER element to be sent to the client. You do not need to define a function of this type. The server will pass a function of this type your **slapi_compute_callback_t** function. In your **slapi_compute_callback_t** function, you need to call this **slapi_compute_output_t** function.

For example:

```
static int my_compute_callback(computed_attr_context *c, char* type,
    Slapi_Entry *e, slapi_compute_output_t outputfn)
{
    ...
    int rc;
    Slapi_Attr my_computed_attr;
    ...

    /* Call the output function after creating the computed
    attribute and setting its values. */
    rc = (*outputfn) (c, &my_computed_attr, e);
    ...
}
```

In the example above, the **slapi_compute_output_t** function **outputfn** is passed in as an argument to **my_compute_callback** function. After generating the computed attribute, you need to call **outputfn**, passing it the context, the newly created attribute, and the entry. **outputfn** BER-encodes the attribute and appends it to the BER element to be sent to the client. You do not need to define **outputfn** yourself. You just need to call the function passed in as the last statement from the callback.

See Also

[slapi_compute_callback_t](#)

14.18. SLAPI_CONNECTION

Represents a connection.

Syntax

```
#include "slapi-plugin.h"
typedef struct conn Slapi_Connection;
```

Description

Slapi_Connection is the data type for an opaque structure that represents a connection.

14.19. SLAPI_CONDVAR

Represents a condition variable in a directory entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_condvar Slapi_CondVar;
```

Description

Slapi_CondVar is the data type for an opaque structure that represents a synchronization lock in the server plug-in. The following table summarizes the front-end API functions that you can call to modify synchronization locks in the server plug-in.

To Call this function
Destroy a condition variable.	slapi_destroy_condvar()
Create a new condition variable.	slapi_new_condvar()
Send notification about a condition variable.	slapi_notify_condvar()
Wait for a condition variable.	slapi_wait_condvar()

14.20. SLAPI_COUNTER

Provides 64-bit integers with support for atomic operations, even on 32-bit systems. **Slapi_Counter** allows plug-ins to use global integers that can be updated by multiple worker threads in a thread-safe manner.

The **Slapi_Counter** structure is a wrapper around the actual counter value.

A mutex is used on platforms that do not provide 64-bit atomic operations.

Syntax

```
#include "slapi-plugin.h"
typedef struct Slapi_Counter {
    PRUint64 value;
#ifdef ATOMIC_64BIT_OPERATIONS
    Slapi_Mutex *mutex;
#endif
} Slapi_Counter;
```

Associated Functions

Slapi_Counter defines settings for counters. The different functions available to

Slapi_Counter structures are listed in [Table 14.13](#), “Functions for Slapi_Counter”.

Table 14.13. Functions for Slapi_Counter

To Call this function
Create a new counter.	slapi_counter_new()
Initialize a new counter.	slapi_counter_init()
Increment the counter value and return the new value.	slapi_counter_increment()
Decrement the counter and return the new value.	slapi_counter_decrement()
Add a certain amount to the counter value.	slapi_counter_add()
Subtract a certain amount from the counter value.	slapi_counter_subtract()
Set the counter to a new, specified value and returns the updated value.	slapi_counter_set_value()
Get the current value of the counter.	slapi_counter_get_value()
Destroy an existing counter.	slapi_counter_destroy()

14.21. SLAPI_DN

Represents a distinguished name in a directory entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_dn Slapi_DN;
```

Description

Slapi_DN is the data type for an opaque structure that represents a distinguished name in the server plug-in. The following table summarizes the front-end API functions that you can call to work with distinguished names.

To Call this function
Supply authentication information from an LDAP bind operation.	slapi_add_auth_response_control()

To Call this function
Specify a distinguished name is a root.	slapi_dn_isroot()
Convert a DN to canonical format and all characters to lower case	slapi_dn_normalize_case()
Normalize part of a DN value	slapi_dn_normalize_to_end()
Build the new DN of an entry.	slapi_moddn_get_newdn()
Add the RDN contained in a Slapi_RDN structure to the DN contained in a Slapi_DN structure.	slapi_sdn_add_rdn()
Compare two DN's.	slapi_sdn_compare()
Copy a DN.	slapi_sdn_copy()
Clear a Slapi_DN structure.	slapi_sdn_done()
Duplicate a Slapi_DN structure.	slapi_sdn_dup()
Free a Slapi_DN structure.	slapi_sdn_free()
Get the DN of the parent within a specific backend.	slapi_sdn_get_backend_parent()
Get the DN from a Slapi_DN structure.	slapi_sdn_get_dn()
Get the normalized DN of a Slapi_DN structure.	slapi_sdn_get_ndn()
Get the length of the normalized DN of a Slapi_DN structure.	slapi_sdn_get_ndn_len()
Get the parent DN of a given Slapi_DN structure.	slapi_sdn_get_parent()
Get the RDN from a normalized DN.	slapi_sdn_get_rdn()
<i>Not implemented; do not use.</i> Check if there is a RDN value that is a component of the DN structure.	slapi_sdn_is_rdn_component()
Check if there is a DN value stored in a Slapi_DN structure.	slapi_sdn_isempty()

To Call this function
Check if a DN is the parent of the parent of a DN.	slapi_sdn_isgrandparent()
Check if a DN is the parent of a DN.	slapi_sdn_isparent()
Check if a Slapi_DN structure contains a suffix of another.	slapi_sdn_issuffix()
Allocate new Slapi_DN structure.	slapi_sdn_new()
Create a new Slapi_DN structure.	slapi_sdn_new_dn_byref()
Create a new Slapi_DN structure.	slapi_sdn_new_dn_byval()
Create a new Slapi_DN structure.	slapi_sdn_new_dn_passin()
Create a new Slapi_DN structure.	slapi_sdn_new_ndn_byref()
Create a new Slapi_DN structure.	slapi_sdn_new_ndn_byval()
Check if an entry is in the scope of a certain base DN.	slapi_sdn_scope_test()
Set a DN value in a Slapi_DN structure.	slapi_sdn_set_dn_byref()
Set a DN value in a Slapi_DN structure.	slapi_sdn_set_dn_byval()
Set a DN value in a Slapi_DN structure.	slapi_sdn_set_dn_passin()
Set a normalized DN in a Slapi_DN structure.	slapi_sdn_set_ndn_byref()
Set a normalized DN in a Slapi_DN structure.	slapi_sdn_set_ndn_byval()
Set a new parent in an entry.	slapi_sdn_set_parent()
Set a new RDN for an entry.	slapi_sdn_set_rdn()
Convert the second RDN type value to the berval value.	slapi_rdn2typeval()

See Also[Slapi_PBlock](#)**14.22. SLAPI_ENTRY**

Represents an entry in the directory.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_entry Slapi_Entry;
```

Description

Slapi_Entry is the data type for an opaque structure that represents an entry in the directory. In certain cases, your server plug-in may need to work with an entry in the directory. The following table summarizes the front-end API functions that you can call to work with entries.

To Call this function
Generate an LDIF string description.	slapi_entry2str()
Generate an LDIF string descriptions with options.	slapi_entry2str_with_options()
Add components in an entry's RDN.	slapi_entry_add_rdn_values()
Add a string value to an attribute in an entry.	slapi_entry_add_string()
Add a data value to an attribute in an entry.	slapi_entry_add_value()
Add an array of data values to an attribute in an entry.	slapi_entry_add_values_sv()
Add a data value to an attribute in an entry.	slapi_entry_add_valueset()
Allocate memory for an entry structure.	slapi_entry_alloc()
Applies an array of LDAPMod to a Slapi_Entry .	Section 24.9, “slapi_entry_apply_mods()”
Delete an attribute from an entry.	slapi_entry_attr_delete()
Check if an entry contains a specific attribute.	slapi_entry_attr_find()
Get the first value as a string.	slapi_entry_attr_get_charptr()
Get the values of a multi-valued attribute of an entry.	slapi_entry_attr_get_chararray()
Get the first value as an integer.	slapi_entry_attr_get_int()

To Call this function
Get the first value as a long.	slapi_entry_attr_get_long()
Get the first value as an unsigned integer.	slapi_entry_attr_get_uint()
Get the first value as an unsigned long.	slapi_entry_attr_get_ulong()
Check if an attribute in an entry contains a value.	slapi_entry_attr_has_syntax_value()
Add an array to the attribute values in an entry.	slapi_entry_attr_merge_sv()
Replace the values of an attribute with a string.	slapi_entry_attr_replace_sv()
Set the first value as a string.	slapi_entry_attr_set_charptr()
Set the first value as an integer.	slapi_entry_attr_set_int()
Set the first value as a long.	slapi_entry_attr_set_long()
Set the first value as an unsigned integer.	slapi_entry_attr_set_uint()
Set the first value as an unsigned long.	slapi_entry_attr_set_ulong()
Delete a string from an attribute.	slapi_entry_delete_string()
Remove a Slapi_Value array from an attribute.	slapi_entry_delete_values_sv()
Copy an entry, its DN, and its attributes.	slapi_entry_dup()
Find the first attribute in an entry.	slapi_entry_first_attr()
Free an entry from memory.	slapi_entry_free()
Get the DN from an entry.	slapi_entry_get_dn()
Return the DN of an entry as a constant.	slapi_entry_get_dn_const()
Return the normalized distinguished name (NDN) of an entry.	slapi_entry_get_ndn()
Return the Slapi_DN from an entry.	slapi_entry_get_sdn()

To Call this function
Return a Slapi_DN from an entry as a constant.	slapi_entry_get_sdn_const()
Get the unique ID from an entry.	slapi_entry_get_uniqueid()
Determine if the specified entry has child entries.	slapi_entry_has_children()
Initialize the values of an entry.	slapi_entry_init()
Add an array of data values to an attribute in an entry.	slapi_entry_merge_values_sv()
Find the next attribute in an entry.	slapi_entry_next_attr()
Check if values present in an entry's RDN are also present as attribute values.	slapi_entry_rdn_values_present()
Determine if an entry complies with the schema for its object class.	slapi_entry_schema_check()
Set the DN of an entry.	slapi_entry_set_dn()
Set the Slapi_DN value in an entry.	slapi_entry_set_sdn()
Set the unique ID in an entry.	slapi_entry_set_uniqueid()
Return the size of an entry.	slapi_entry_size()
Determine if an entry is the root DSE.	slapi_is_rootdse()
Convert an LDIF description into an entry.	slapi_str2entry()

See Also

[Slapi_Attr](#)

14.23. SLAPI_FILTER

Represents a search filter.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_filter Slapi_Filter;
```

Description

Slapi_Filter is the data type for an opaque structure that represents an search filter. (For more information on search filters, see [Section 5.5, “Working with Entries, Attributes, and Values”](#).) The following table summarizes the front-end API functions that you can call to work with filters.

To Call this function
Determine if an entry matches a filter's criteria.	slapi_filter_test()
Get the filter type.	slapi_filter_get_choice()
Get the attribute type and value used for comparison in a filter (only applicable to <i>LDAP_FILTER_EQUALITY</i> , <i>LDAP_FILTER_GE</i> , <i>LDAP_FILTER_LE</i> , and <i>LDAP_FILTER_APPROX</i> searches).	slapi_filter_get_ava()
Get the type of attribute that the filter is searching for (only applicable to <i>LDAP_FILTER_PRESENT</i> searches).	slapi_filter_get_type()
Get the substring pattern used for the filter (applicable only to <i>LDAP_FILTER_SUBSTRING</i> searches).	slapi_filter_get_subfilt()
Convert a string representation of a filter to a filter of the data type Slapi_Filter.	slapi_str2filter()
Construct a new <i>LDAP_FILTER_AND</i> , <i>LDAP_FILTER_OR</i> , or <i>LDAP_FILTER_NOT</i> filter from other filters.	slapi_filter_join()
Get the components of a filter (only applicable to <i>LDAP_FILTER_AND</i> , <i>LDAP_FILTER_OR</i> , and <i>LDAP_FILTER_NOT</i> searches).	slapi_filter_list_first()
Free a filter from memory.	slapi_filter_free()

14.24. SLAPI_MATCHINGRULEENTRY

Slapi_MatchingRuleEntry is the data type for an opaque structure that represents a matching rule.

The matching rule definition can be specified as a dynamic declaration using functions such as [slapi_matchingrule_new\(\)](#) or [slapi_matchingrule_set\(\)](#).

Alternatively, the matching rule definition can also be specified as a static declaration. For example:

■

```
static Slapi_MatchingRuleEntry
integerMatch = { INTEGERMATCH_OID, NULL /* no alias? */,
  "integerMatch", "The rule evaluates to TRUE if and only if the
attribute value and the assertion value are the same integer value.",
  INTEGER_SYNTAX_OID, 0 /* not obsolete */ };
...
int
int_init( Slapi_PBlock *pb )
{
  int rc;
  ...
  rc = slapi_matchingrule_register(&integerMatch);
  ...
}
```

Table 14.14, “Frontend API Functions for Slapi_MatchingRuleEntry” summarizes the front-end API functions that can be called to work with matching rules.

Syntax

This function has the following syntax:

```
typedef struct slapi_matchingRuleEntry {
  char *mr_oid;
  char *mr_oidalias;
  char *mr_name;
  char *mr_desc;
  char *mr_syntax;
  int mr_obsolete;
} slapi_MatchingRuleEntry;
typedef struct slapi_matchingRuleEntry Slapi_MatchingRuleEntry;
```

See Also

Table 14.14. Frontend API Functions for Slapi_MatchingRuleEntry

To perform this action...	Call this function
Compare two berval structures to determine if they are equal.	slapi_berval_cmp()
Call the indexer function associated with an extensible match filter.	slapi_mr_filter_index()
Free the specified matching rule structure (and optionally, its members) from memory.	slapi_matchingrule_free()
Get information about a matching rule.	slapi_matchingrule_get()
Call the indexer factory function for the plug-in responsible for a specified matching rule.	slapi_mr_indexer_create()

To perform this action...	Call this function
Allocate memory for a new Slapi_MatchingRuleEntry structure.	slapi_matchingrule_new()
Register the specified matching rule with the server.	slapi_matchingrule_register()
Set information about the matching rule.	slapi_matchingrule_set()
Determines if a matching rule is a valid ordering matching rule for the given syntax.	slapi_matchingrule_is_ordering()
Reserved for future use.	slapi_matchingrule_unregister()

14.25. SLAPI_MOD

Represents a single LDAP modification to a directory entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_mod Slapi_Mod;
```

Description

Slapi_Mod is the data type for an opaque structure that represents LDAPMod modifications to an attribute in a directory entry.

The following table summarizes the front-end API functions that you can call to manipulate directory entries.

To Call this function
Add a value to a Slapi_Mod structure.	slapi_mod_add_value()
Free internals of Slapi_Mod structure.	slapi_mod_done()
Dump the contents of an LDAPMod to the server log.	slapi_mod_dump()
Free a Slapi_Mod structure.	slapi_mod_free()
Initialize a Slapi_Mod iterator and return the first attribute value.	slapi_mod_get_first_value()
Get a reference to the LDAPMod in a Slapi_Mod structure.	slapi_mod_get_ldapmod_byref()

To Call this function
Retrieve the LDAPMod contained in a Slapi_Mod structure.	slapi_mod_get_ldapmod_passout()
Increment the Slapi_Mod iterator and return the next attribute value.	slapi_mod_get_next_value()
Get the number of values in a Slapi_Mod structure.	slapi_mod_get_num_values()
Get the operation type of Slapi_Mod structure.	slapi_mod_get_operation()
Get the attribute type of a Slapi_Mod structure.	slapi_mod_get_type()
Initialize a Slapi_Mod structure.	slapi_mod_init()
Initialize a Slapi_Mod structure that is a wrapper for an existing LDAPMod.	slapi_mod_init_byref()
Initialize a modification by value.	slapi_mod_init_byval()
Initialize a Slapi_Mod from an LDAPMod.	slapi_mod_init_passin()
Initializes the given <i>smod</i> with the given LDAP operation and attribute type.	Section 33.17 , “slapi_mod_init_valueset_byval()”
Determine whether a Slapi_Mod structure is valid.	slapi_mod_isvalid()
Allocate a new Slapi_Mod structure.	slapi_mod_new()
Remove the value at the current Slapi_Mod iterator position.	slapi_mod_remove_value()
Set the operation type of a Slapi_Mod structure.	slapi_mod_set_operation()
Set the attribute type of a Slapi_Mod.	slapi_mod_set_type()

See Also

[LDAPMod](#) and [Section 14.26, “Slapi_Mods”](#)

14.26. SLAPI_MODS

Represents two or more LDAP modifications to a directory entry

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_mods Slapi_Mods;
```

Description

Slapi_Mods is the data type for an opaque structure that represents LDAPMod manipulations that can be made to a directory entry.

The following table summarizes the front-end API functions that you can call to manipulate directory entries.

To Call this function
Create a Slapi_Entry from an array of LDAPMod.	slapi_mods2entry()
Append a new mod with a single attribute value to Slapi_Mods structure.	slapi_mods_add()
Append an LDAPMod to a Slapi_Mods structure.	slapi_mods_add_ldapmod()
Append a new mod to a Slapi_Mods structure, with attribute values provided as an array of berval.	slapi_mods_add_modbvps()
Append a new mod to a Slapi_Mods structure, with attribute values provided as an array of Slapi_Value.	slapi_mods_add_mod_values()
Append a new mod to Slapi_Mods structure with a single attribute value provided as a string.	slapi_mods_add_string()
Complete a modification.	slapi_mods_done()
Dump the contents of a Slapi_Mods structure to the server log.	slapi_mods_dump()
Free a Slapi_Mods structure.	slapi_mods_free()
Initialize a Slapi_Mods iterator and return the first LDAPMod.	slapi_mods_get_first_mod()
Get a reference to the array of LDAPMod in a Slapi_Mods structure.	slapi_mods_get_ldapmods_byref()
Retrieve the array of LDAPMod contained in a Slapi_Mods structure.	slapi_mods_get_ldapmods_passout()

To Call this function
Increment the Slapi_Mods iterator and return the next LDAPMod.	slapi_mods_get_next_mod()
Increment the Slapi_Mods iterator and return the next mod wrapped in a Slapi_Mods.	slapi_mods_get_next_smod()
Get the number of mods in a Slapi_Mods structure.	slapi_mods_get_num_mods()
Initialize a Slapi_Mods.	slapi_mods_init()
Initialize a Slapi_Mods that is a wrapper for an existing array of LDAPMod.	slapi_mods_init_byref()
Initialize a Slapi_Mods structure from an array of LDAPMod.	slapi_mods_init_passin()
Insert an LDAPMod into a Slapi_Mods structure after the current iterator position.	slapi_mods_insert_after()
Insert an LDAPMod anywhere in a Slapi_Mods.	slapi_mods_insert_at()
Insert an LDAPMod into a Slapi_Mods structure before the current iterator position.	slapi_mods_insert_before()
Decrement the Slapi_Mods current iterator position.	slapi_mods_iterator_backone()
Allocate a new uninitialized Slapi_Mods structure.	slapi_mods_new()
Remove the mod at the current Slapi_Mods iterator position.	slapi_mods_remove()

See Also

[LDAPMod](#) and [Slapi_Mod](#)

14.27. SLAPI_MUTEX

Represents a mutually exclusive lock in the server plug-in.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_mutex Slapi_Mutex;
```


Description

Slapi_Mutex is the data type for an opaque structure that represents a mutual exclusive lock (mutex) in the server plug-in.

The following table summarizes the front-end API functions that you can call to work with mutually exclusive locks.

To Call this function
Destroy a mutex.	slapi_destroy_mutex()
Lock a mutex.	slapi_lock_mutex()
Create a new mutex.	slapi_new_mutex()
Unlock a mutex.	slapi_unlock_mutex()

14.28. SLAPI_OPERATION

Represents an operation pending from an LDAP client.

Syntax

```
#include "slapi-plugin.h"
typedef struct op Slapi_Operation;
```

Description

Slapi_Operation is the data type for an opaque structure that represents an operation pending from an LDAP client.

The following table summarizes the front-end API functions that you can call to work with mutually exclusive locks.

To Call this function
Determine if the client has abandoned the current operation.	slapi_op_abandoned()
Get the type of a Slapi_Operation.	slapi_op_get_type()

14.29. SLAPI_PBLOCK

Contains name-value pairs, known as parameter blocks, that you can get or set for each LDAP operation.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_pblock Slapi_PBlock;
```

Description

Slapi_PBlock contains name-value pairs that you can use to retrieve information from the server and set information to be used by the server.

For most types of plug-in functions, the server passes in a Slapi_PBlock structure that typically includes data relevant to the operation being processed. You can get the value of a parameter by calling the **slapi_pblock_get()** function.

For example, when the plug-in function for an LDAP bind operation is called, the server puts the DN and credentials in the *SLAPI_BIND_TARGET* and *SLAPI_BIND_CREDENTIALS* parameters of the Slapi_PBlock structure. You can call **slapi_pblock_get()** to get the DN and credentials of the client requesting authentication.

For plug-in initialization functions, you can use the Slapi_PBlock structure to pass information to the server, such as the description of your plug-in and the names of your plug-in functions. You can set the value of a parameter by calling the **slapi_pblock_set()** function.

For example, in order to register a pre-operation bind plug-in function, you need to call **slapi_pblock_set()** to set the version number, description, and name of the plug-in function as the *SLAPI_PLUGIN_VERSION*, *SLAPI_PLUGIN_DESCRIPTION*, and *SLAPI_PLUGIN_PRE_BIND_FN* parameters.

The available parameters that you can use depends on the type of plug-in function you are writing.

The following table summarizes the front-end API functions that you can call to work with block parameters.

To Call this function
Set up a parameter block so that it can be used by slapi_add_internal_pb() for an internal add operation.	slapi_add_entry_internal_set_pb()
Add an LDAP add operation based on a parameter block to add a new directory entry.	slapi_add_internal_pb()
Set up a parameter block so that it can be used by slapi_add_internal_pb() for an internal add operation; the entry is constructed from a DN and a set of attributes.	slapi_add_internal_set_pb()
Perform an LDAP delete operation based on a parameter block to remove a directory entry.	slapi_delete_internal_pb()
Delete an internal parameter block set.	slapi_delete_internal_set_pb()

To Call this function
Perform an LDAP modify operation based on a parameter block to modify a directory entry.	slapi_modify_internal_pb()
Set up a parameter block so that it can be used by slapi_modify_internal_pb() for an internal modify operation.	slapi_modify_internal_set_pb()
Perform an LDAP modify RDN operation based on a parameter block to rename a directory entry.	slapi_modrdn_internal_pb()
Free a pblock from memory.	slapi_pblock_destroy()
Get the value from a pblock.	slapi_pblock_get()
Initialize an existing parameter block so that it can be reused.	Section 35.3, “slapi_pblock_init()”
Create a new pblock.	slapi_pblock_new()
Set the value of a pblock.	slapi_pblock_set()
Set up a parameter block so that it can be used by slapi_modrdn_internal_pb() for an internal rename operation.	slapi_rename_internal_set_pb()
Perform an LDAP search operation based on a parameter block to search the directory.	slapi_search_internal_callback_pb()
Search for an internal parameter block.	slapi_search_internal_pb()
Set up a parameter block so that it can be used by slapi_search_internal_pb() for an internal search operation.	slapi_search_internal_set_pb()
Perform an internal sequential access operation.	slapi_seq_internal_callback_pb()
Set up a parameter block for use by slapi_seq_internal_callback_pb() for an internal, sequential-access operation.	slapi_seq_internal_set_pb()

14.30. SLAPI_PLUGINDESC

Represents information about a server plug-in.

Syntax

```
typedef struct slapi_plugin_desc {
    char *spd_id;
    char *spd_vendor;
    char *spd_version;
    char *spd_description;
} Slapi_PluginDesc;
```

Parameters

The function has the following parameters:

<i>spd_id</i>	Unique identifier for the server plug-in.
<i>spd_vendor</i>	Name of the vendor supplying the server plug-in; for example, example.com .
<i>spd_version</i>	Version of the server plug-in used for your own tracking purposes; for example, 0.5 . This is different from the value of the <i>SLAPI_PLUGIN_VERSION</i> , which specifies the general version of plug-in technology; the Directory Server uses that version to determine if it supports a plug-in.
<i>spd_description</i>	Description of the server plug-in.

Description

Slapi_PluginDesc represents information about a server plug-in. In your initialization function, you specify information about your plug-in in this structure and call **slapi_pblock_set()** to put the structure in the *SLAPI_PLUGIN_DESCRIPTION* parameter.

See Also

For more information on using Slapi_PluginDesc to specify plug-in information, see [Section 2.2.2, “Specifying Information about the Plug-in”](#).

14.31. SLAPI_RDN

Represents a relative distinguished name in a directory entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_rdn Slapi_RDN;
```

Description

Slapi_RDN is the data type for an opaque structure that represents a relative distinguished name in the server plug-in.

The following table summarizes the front-end API functions that you can call to work with relative distinguished names.

To Call this function
Add a new RDN to an existing RDN structure.	slapi_rdn_add()
Compare two RDNs.	slapi_rdn_compare()
Check if a Slapi_RDN structure holds any RDN matching a given type/value pair.	slapi_rdn_contains()
Check if a Slapi_RDN structure contains any RDN matching a given type.	slapi_rdn_contains_attr()
Clear a Slapi_RDN structure.	slapi_rdn_done()
Free a Slapi_RDN structure.	slapi_rdn_free()
Get the type/value pair of the first RDN.	slapi_rdn_get_first()
Get the index of the RDN.	slapi_rdn_get_index()
Get the position and the attribute value of the first RDN.	
Get the RDN type/value pair from the RDN.	slapi_rdn_get_next()
Get the number of RDN type/value pairs.	slapi_seq_internal_set_pb()
Get the RDN from a Slapi_RDN structure.	slapi_seq_internal_set_pb()
Initialize a Slapi_RDN structure.	slapi_rdn_init()
Initialize a Slapi_RDN structure with an RDN value taken from a given DN.	slapi_rdn_init_dn()
Initializes Slapi_RDN structure with an RDN value.	slapi_rdn_init_rdn()
Initialize a Slapi_RDN structure with an RDN value taken from the DN contained in a given Slapi_RDN.	
Check if an RDN value is stored in a Slapi_RDN structure.	slapi_rdn_isempty()
Allocate a new Slapi_RDN structure.	slapi_rdn_new()

To Call this function
Create a new Slapi_RDN structure.	slapi_rdn_new_dn()
Create a new Slapi_RDN structure and set an RDN value.	slapi_rdn_new_rdn()
Create a new Slapi_RDN structure and set an RDN value taken from the DN contained in a given Slapi_RDN structure.	slapi_rdn_new_sdn()
Remove an RDN type/value pair.	slapi_rdn_remove()
Remove an RDN type/value pair from a Slapi_RDN.	slapi_rdn_remove_attr()
Remove an RDN type/value pair from a Slapi_RDN structure.	slapi_rdn_remove_index()
Set an RDN value in a Slapi_RDN structure.	slapi_rdn_set_dn()
Set an RDN in a Slapi_RDN structure.	slapi_rdn_set_rdn()
Set an RDN value in a Slapi_RDN structure.	slapi_rdn_set_sdn()
Add an RDN to a DN.	

14.32. SLAPI_TASK

An opaque structure that represents a task that has been initiated.

Common Directory Server tasks, including importing, exporting, and indexing databases, can be initiated through a special task configuration entry in **cn=tasks,cn=config**. These task operations are managed using the **Slapi_Task** structure.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_task Slapi_Task;
```

Associated typedefs

There are two additional **typedef** declarations associated with the **Slapi_Task** structure.

Table 14.15. typedefs for the Slapi_Task Structure

typedef	Description
dseCallbackFn	Sets callback information for the frontend plug-in.
TaskCallbackFn	Defines a callback used by Slapi_Task cancel and destructor functions.

Associated Functions

All of the available functions for Directory Server tasks are listed in [Table 14.16, “Functions for the Slapi_Task Structure”](#).

Table 14.16. Functions for the Slapi_Task Structure

To Call this function
Create a new server task and returns a pointer to the Slapi_Task structure.	slapi_plugin_new_task()
Update a task entry to indicate that the task is running.	slapi_task_begin()
Cancel a current task.	slapi_task_cancel()
Decrement the task reference count.	slapi_task_dec_refcount()
Write that the task is complete and returns the result code.	slapi_task_finish()
Retrieve an opaque data pointer from the task.	slapi_task_get_data()
Check the current reference count of the task. If a task has multiple threads, this shows whether the individual tasks have completed.	slapi_task_get_refcount()
Show the current state of the task.	slapi_task_get_state()
Automatically increment the task progress, which updates the task entry.	slapi_task_inc_progress()
Increment the task reference count, if the task uses multiple threads.	slapi_task_inc_refcount()
Write changes to a log attribute for the task entry.	slapi_task_log_notice()

To Call this function
Update the task status attribute in the entry to maintain a running display of the task status.	slapi_task_log_status()
Register a task handler function.	slapi_plugin_task_register_handler()
Unregister a plug-in task.	slapi_plugin_task_unregister_handler()
Set a callback to be used when a task is cancelled.	slapi_task_set_cancel_fn()
Append an opaque object pointer to the task process.	slapi_task_set_data()
Set a callback to be used when a task is destroyed.	slapi_task_set_destructor_fn()
Contain the task's status.	slapi_task_status_changed()

14.32.1. dseCallbackFn

Sets callback information for the frontend plug-in.

Syntax

```
#include "slapi-plugin.h"
typedef int (*dseCallbackFn)(Slapi_PBlock *, Slapi_Entry *, Slapi_Entry *,
                             int *, char*, void *);
```

Returns

This callback must return one of the following messages:

- **SLAPI_DSE_CALLBACK_OK (0)** for successful operations, meaning that any directory changes can be performed.
- **SLAPI_DSE_CALLBACK_ERROR (1)** for any errors, which means that any directory changes cannot be performed.
- **SLAPI_DSE_CALLBACK_DO_NOT_APPLY (-1)**, which is returned if there are no errors but the changes should still not be applied. This only applies for modify operations; otherwise, the server interprets **SLAPI_DSE_CALLBACK_DO_NOT_APPLY** as **SLAPI_DSE_CALLBACK_ERROR**.

See Also

- [Slapi_DN](#)
- [Slapi_PBlock](#)

14.32.2. TaskCallbackFn

Defines a callback used specifically by **Slapi_Task** structure cancel and destructor functions.

Syntax

```
#include "slapi-plugin.h"
typedef void (*TaskCallbackFn)(Slapi_Task *task);
```

Parameters

This function takes the following parameter:

Parameter	Description
<i>task</i>	Points to the task operation which is being performed by the server.

Returns

Currently, this callback only returns a success message (**0**). The actual return values for the functions are not checked by the callback.

14.33. SLAPI_UNIQUEID

Represents the unique identifier of a directory entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct _guid_t Slapi_UniqueID;
```

Description

Slapi_UniqueID is the data type for an opaque structure that represents the unique identifier of a directory entry. All directory entries contain a unique identifier. Unlike the distinguished name (DN), the unique identifier of an entry never changes, providing a good way to refer unambiguously to an entry in a distributed/replicated environment.

14.34. SLAPI_VALUE

Represents the value of the attribute in a directory entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_value Slapi_Value;
```

Description

Slapi_Value is the data type for an opaque structure that represents the value of an attribute in a directory entry.

The following table summarizes the front-end API functions that you can call to work with directory entry values.

To Call this function
Compare a value.	slapi_value_compare()
Duplicate a value.	
Free a Slapi_Value structure from memory.	slapi_value_free()
Get the berval structure of the value.	slapi_value_get_berval()
Get flags from a Slapi_Value structure.	slapi_value_get_flags()
Convert the value of an integer.	slapi_value_get_int()
Get the length of a value.	slapi_value_get_length()
Get the actual length of the value.	slapi_value_get_long()
Convert a value into a long integer.	slapi_value_get_long()
Return the value as a string. The value returned may not be null-terminated.	slapi_value_get_string()
Convert the value into an unsigned integer.	slapi_value_get_uint()
Convert the value into an unsigned long.	slapi_value_get_ulong()
Initialize a Slapi_Value structure with no values.	slapi_value_init()
Initialize a Slapi_Value structure from the berval structure.	slapi_value_init_berval()
Initialize a Slapi_Value structure from a string.	slapi_value_init_string()
Initialize a Slapi_Value structure with a value contained in a string.	slapi_value_init_string_passin()
Allocate a new Slapi_Value structure.	slapi_value_new()

To Call this function
Allocate a new Slapi_Value structure from a berval structure.	slapi_value_new_berval()
Allocate a new Slapi_Value structure from a string.	slapi_value_new_string()
Allocate a new Slapi_Value structure and initializes it from a string.	slapi_value_new_string_passin()
Allocate a new Slapi_Value from another Slapi_Value structure.	slapi_value_new_value()
Set the value.	slapi_value_set()
Copy the value from a berval structure into a Slapi_Value structure.	slapi_value_get_berval()
Set flags for a Slapi_Value structure.	slapi_value_set_flags()
Set the integer value of a Slapi_Value structure.	slapi_value_set_int()
Copy a string into thevalue.	slapi_value_set_string()
Set the value.	slapi_value_set_string_passin()
Copy the value of a Slapi_Value structure into another Slapi_Value structure.	slapi_value_set_value()
Set flags to the array of a Slapi_Value structure.	slapi_values_set_flags()

See Also[Slapi_ValueSet](#)**14.35. SLAPI_VALUESET**

Represents a set of Slapi_Value (or a list of Slapi_Value).

Slapi_ValueSet is the data type for an opaque structure that represents set of Slapi_Value (or a list of Slapi_Value).

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_value_set Slapi_ValueSet;
```

■

The following table summarizes the front-end API functions that you can call to work with sets of Slapi_Value.

To Call this function
Add a Slapi_Value in the Slapi_ValueSet structure.	slapi_valueset_add_value()
Count the values in a valueset.	slapi_valueset_count()
Free the values contained in the Slapi_ValueSet structure.	slapi_valueset_done()
Find the value in a valueset using the syntax of an attribute.	slapi_valueset_find()
Get the first value of a Slapi_ValueSet structure.	slapi_valueset_first_value()
Free the specified Slapi_ValueSet structure and its members from memory.	slapi_valueset_free()
Reset a Slapi_ValueSet structure to no values.	slapi_valueset_init()
Allocate a new Slapi_ValueSet structure.	slapi_valueset_next_value()
Get the next value from a Slapi_ValueSet structure.	
Add the changes in a modification to a valueset.	slapi_valueset_set_from_smod()
Initialize a Slapi_ValueSet structure from another Slapi_ValueSet structure.	slapi_valueset_set_valueset()

See Also

[Section 14.34, “Slapi_Value”](#)

14.36. REPLICATION SESSION HOOKS CALLBACKS

Client applications can have some control over when the Directory Server performs replication by using custom plug-ins that define *replication session hooks*. The hooks apply to both the master sending the information and the consumer receiving the information. Basically, the master sends a preliminary message to the consumer and the consumer sends a response message. If either side sends a message that does not meet the expected criteria (such as the server version or some schema requirement), then the replication operation is terminated.

The intent of replication session hooks is to require some kind of parity between the servers

involved in replication. If there is a situation that could cause conflicts or data corruption — such as different server versions which use different sets of schema — then the session hooks identify that potential problem. This prevents replication when it could hurt server performance.

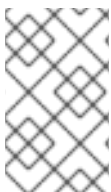
The session hooks are implemented through callback functions in the Directory Server replication functions.

The call backs are defined in an ordered array that follows the replication operation process from initializing the replication agreement to receiving the response from the consumer.

```
static void *test_repl_session_api[] = {
    NULL, /* reserved for api broker use, must be zero */
    test_repl_session_plugin_agmt_init_cb,
    test_repl_session_plugin_pre_acquire_cb,
    test_repl_session_plugin_reply_acquire_cb,
    test_repl_session_plugin_post_acquire_cb,
    test_repl_session_plugin_recv_acquire_cb,
    test_repl_session_plugin_destroy_cb
};
```

Then, the custom replication plug-in registers the callbacks. In this example, it requires a certain version of the server.

```
int test_repl_session_plugin_init(Slapi_PBlock *pb)
{
    ...
    if( slapi_apib_register(REPL_SESSION_v1_0_GUID, test_repl_session_api)
    ) {
        slapi_log_error( SLAPI_LOG_FATAL, test_repl_session_plugin_name,
            "<-- test_repl_session_plugin_start -- failed to
register repl_session api -- end\n");
        return -1;
    }
    ...
}
```



NOTE

The callbacks themselves are described in the *Plug-in Guide*. This example is provided to make administrators and programmers aware that the replication process can be controlled by using these session hooks.

These **typedef** declarations are used to define replication session hooks for Directory Server and its clients, including applications like Red Hat Enterprise IPA.

Table 14.17. Replication Session Hooks Types

typedef	Description
<code>repl_session_plugin_agmt_init_cb</code>	Used to initialize the replication agreement to begin replication.

typedef	Description
repl_session_plugin_pre_acquire_cb	Defines the data about the initiating master server to send to the replica.
repl_session_plugin_reply_acquire_cb	Defines the information about the replica to send to the master.
repl_session_plugin_post_acquire_cb	Posts the response from the replica to the master server.
repl_session_plugin_rcv_acquire_cb	Receives the data from the replica.
repl_session_plugin_destroy_agmt_cb	Destroys a replication agreement when the operation is complete.

14.36.1. repl_session_plugin_agmt_init_cb

Initializes a replication agreement.

Syntax

```
typedef void * (*repl_session_plugin_agmt_init_cb)(const Slapi_DN
*repl_subtree);
```

Parameters

This function takes the following parameters:

Parameter	Description
<i>repl_subtree</i>	Gives the subtree that is involved in replication.

14.36.2. repl_session_plugin_pre_acquire_cb

Defines the data about the initiating master server to send to the replica.

Syntax

```
typedef int (*repl_session_plugin_pre_acquire_cb)(void *cookie, const
Slapi_DN *repl_subtree,
int is_total, char
**data_guid, struct berval **data);
```

Parameters

This function takes the following parameters:

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the replication plug-in init function (if any) and passed to all plug-in functions.
<i>repl_subtree</i>	Gives the subtree that is involved in replication.
<i>data_guid</i>	Sends an identifier for the expected data.
<i>data_guid</i>	Contains the data.

14.36.3. repl_session_plugin_reply_acquire_cb

Defines the information about the replica to send to the master.

Syntax

```
typedef int (*repl_session_plugin_reply_acquire_cb)(const char
*repl_subtree, int is_total,
                                                    char **data_guid,
struct berval **data);
```

Parameters

This function takes the following parameters:

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the replication plug-in init function (if any) and passed to all plug-in functions.
<i>repl_subtree</i>	Gives the subtree that is involved in replication.
<i>data_guid</i>	Contains the identifier for the expected data.
<i>data_guid</i>	Contains the data.

14.36.4. repl_session_plugin_post_acquire_cb

Posts the response from the replica to the master server.

Syntax

```
typedef int (*repl_session_plugin_post_acquire_cb)(void *cookie, const
Slapi_DN *repl_subtree,
                                                    int is_total, const char
*data_guid, const struct berval *data);
```

Parameters

This function takes the following parameters:

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the replication plug-in init function (if any) and passed to all plug-in functions.
<i>repl_subtree</i>	Gives the subtree that is involved in replication.
<i>data_guid</i>	Contains the identifier for the expected data.
<i>data_guid</i>	Contains the data.

14.36.5. repl_session_plugin_rcv_acquire_cb

Receives the data from the replica.

Syntax

```
typedef int (*repl_session_plugin_rcv_acquire_cb)(const char
*repl_subtree, int is_total,
                                                    const char *data_guid, const
struct berval *data);
```

Parameters

This function takes the following parameters:

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the replication plug-in init function (if any) and passed to all plug-in functions.
<i>repl_subtree</i>	Gives the subtree that is involved in replication.

Parameter	Description
<i>data_guid</i>	Contains the identifier for the expected data.
<i>data_guid</i>	Contains the data.

14.36.6. repl_session_plugin_destroy_agmt_cb

Destroys a replication agreement when the operation is complete.

Syntax

```
typedef void (*repl_session_plugin_destroy_agmt_cb)(void *cookie, const
Slapi_DN *repl_subtree);
```

Parameters

This function takes the following parameters:

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the replication plug-in init function (if any) and passed to all plug-in functions.
<i>repl_subtree</i>	Gives the subtree that is involved in replication.

14.37. SYNCHRONIZATION CALLBACKS AND DATA TYPES

These **typedef** declarations are used for both directions of Directory Server and Active Directory synchronization and can be used for adding or modifying entries in both servers.

Table 14.18. WinSync Types

typedef	Description
winsync_can_add_to_ad_cb	Used to determine if a Directory Server entry should be added to the Active Directory server.
winsync_get_new_dn_cb	Specifies a DN for a new entry being synced from the Active Directory server over to the Directory Server.
winsync_plugin_init_cb	Initializes the synchronization plug-in.

typedef	Description
<code>winsync_pre_add_cb</code>	Called whenever a new entry is being added to the Directory Server.
<code>winsync_pre_ad_mod_mods_cb</code>	Specifies modifications that must be synced over to the Active Directory server.
<code>winsync_pre_mod_cb</code>	Sets the main entry points that allow the sync plug-in to intercept modifications between local and remote entries.
<code>winsync_search_params_cb</code>	Sets the search parameters for the Active Directory and Directory Server instances, based on the sync agreement.

14.37.1. winsync_can_add_to_ad_cb

Sets a callbacks to determine if an entry in the Directory Server should be added to the Active Directory server if the entry does not already exist in the Windows domain.

Syntax

```
#include "slapi-plugin.h"
typedef int (*winsync_can_add_to_ad_cb)(void *cookie, const Slapi_Entry
*local_entry, const Slapi_DN *remote_dn);
```

Parameters

This function takes the following parameters:

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the Windows Sync plug-in init function (if any) and passed to all plug-in functions.
<i>local_entry</i>	The Directory Server copy of the entry being checked by the server.
<i>remote_dn</i>	The remote copy of the entry to add to the Active Directory server.

Returns

If the server is allowed to add the entry to the Active Directory server, then **winsync_can_add_to_ad_cb** declaration returns **16**.

14.37.2. winsync_get_new_dn_cb

Specifies a DN for a new entry being synced from the Active Directory server over to the Directory Server instance.

When a new entry is created on the Active Directory server and it synced over to the Directory Server, it may be necessary to generate a new DN for the entry. This can result in a naming conflict between the name on one server and the generated name from the synchronization plug-in. This callback function allows the sync plug-in to set the new DN for the entry.

The `map_entry_dn_inbound` function is called to identify the DN for the new entry is needed. The `winsync_plugin_call_pre_ds_add_*` callbacks can also be used to set the DN for the new entry before it is stored in the Directory Server.

This data type is also used when an attribute with a DN as a value, such as ***owner*** or ***secretary***, is synchronized.

Syntax

```
#include "slapi-plugin.h"
typedef void (*winsync_get_new_dn_cb)(void *cookie, const Slapi_Entry
*rawentry, Slapi_Entry *ad_entry, char **new_dn_string,
const Slapi_DN *ds_suffix, const Slapi_DN *ad_suffix);
```

Parameters

This function takes the following parameters:

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the Windows Sync plug-in init function (if any) and passed to all plug-in functions.
<i>rawentry</i>	The unprocessed Active Directory entry, as it is read directly from Active Directory. This entry is read-only.
<i>ad_entry</i>	The processed Active Directory entry.
<i>new_dn_string</i>	The given value of the DN for the new entry. The default value is created by the sync code using memory allocated by slapi_ch_malloc() . slapi_ch_free_string() is called to free this memory when it is not longer needed.
<i>ds_suffix</i>	The Directory Server suffix being synchronized.
<i>ad_suffix</i>	The Active Directory suffix being synchronized.

Returns

There are two possible returns:

- For a Directory Server user, this returns **12**.
- For a Directory Server group, this returns **13**.

See Also

- `winsync_plugin_call_pre_ds_add_*`
- `map_entry_dn_inbound`

14.37.3. winsync_plugin_init_cb

Initializes the synchronization plug-in.

Whenever synchronization begins, the sync plug-in defines this callback to initialize the other sync callbacks.

The Directory Server and Active Directory subtrees are passed in from the sync agreement as read-only attributes. The return value is private data for the sync plug-in which is passed with each sync callback. If the sync plug-in returns a value, the plug-in must define a **winsync_plugin_destroy_agmt_cb** callback so that the private data can be freed. This private data is passed to every other callback function as the **void *cookie** argument.

Syntax

```
#include "slapi-plugin.h"
typedef void * (*winsync_plugin_init_cb)(const Slapi_DN *ds_subtree, const
Slapi_DN *ad_subtree);
```

Parameters

This function takes the following parameters:

Table 14.19. winsync_plugin_init_cb Parameters

Parameter	Description
<i>ds_subtree</i>	The Directory Server subtree being synchronized.
<i>ad_subtree</i>	The Active Directory subtree being synchronized.

Returns

If the plug-in is successfully initialized, the server returns **1**.

14.37.4. winsync_pre_add_cb

Called whenever a new entry is being added to the Directory Server after being synced over from the Active Directory server.

Syntax

```
#include "slapi-plugin.h"
typedef void (*winsync_pre_add_cb)(void *cookie, const Slapi_Entry
*rawentry, Slapi_Entry *ad_entry, Slapi_Entry *ds_entry);
```

Parameters

This function takes the following parameters:

Table 14.20. winsync_pre_add_cb Parameters

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the Windows Sync plug-in init function (if any) and passed to all plug-in functions.
<i>rawentry</i>	The unprocessed Active Directory entry, as it is read directly from Active Directory. This entry is read-only.
<i>ad_entry</i>	The processed Active Directory entry.
<i>ds_entry</i>	The entry to be added to the Directory Server. Any modifications to the new entry should be made to this entry. This includes changing the DN, since the DN of this processed entry is used as the target DN for the final new entry in the Directory Server. This processed entry already has the default schema mapping applied.

Returns

There are two possible returns:

- For a user, this returns **10**.
- For a group, this returns **11**.

14.37.5. winsync_pre_ad_mod_mods_cb

Specifies modifications to be synced over to the Active Directory server from the results of an LDAP modify operation.

The plug-in may alter the list of modifications (specified as an array of **LDAPMod*** objects) before they are sent to Active Directory as an LDAP modify operation.

Syntax

```
#include "slapi-plugin.h"
typedef void (*winsync_pre_ad_mod_mods_cb)(void *cookie, const Slapi_Entry
*rawentry, const Slapi_DN *local_dn, const Slapi_Entry *ds_entry, LDAPMod
* const *origmods, Slapi_DN *remote_dn, LDAPMod ***modstosend);
```

Parameters

This function takes the following parameters:

Table 14.21. winsync_pre_ad_mod_mods_cb Parameters

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the Windows Sync plug-in init function (if any) and passed to all plug-in functions.
<i>rawentry</i>	The unprocessed Active Directory entry, as it is read directly from Active Directory. This entry is read-only. This can be NULL .
<i>local_dn</i>	The original Directory Server DN specified in the modification.
<i>ds_entry</i>	The Directory Server entry which is the source of these modifications. The modifications have already been made to this entry.
<i>origmods</i>	The list of actual modifications made to the local entry.
<i>remote_dn</i>	The DN of the entry on the Active Directory server to which to write the changes; this may be calculated by the sync plug-in.
<i>modstosend</i>	The list of modifications which will be written to the Active Directory entry. The changes being sent have the attributes mapped between Directory Server and Active Directory schema so that the changes to be sent fit with Active Directory schema.

Returns

There are two possible returns:

- If modifications are applied to a user, this returns **14**.
- If modifications are applied to a group, this returns **15**.

14.37.6. winsync_pre_mod_cb

These callbacks are called as the result of an LDAP add operation, when the Win Sync code determines that the destination entry already exists. In that case, the Win Sync code converts the add operation to a modify operation, calculates the difference between the entries, and formats those differences as a list of LDAP modify operations, specified in the *smods* parameter. The **PRE_AD** functions are used when the destination is the Active Directory entry, and the **PRE_DS** functions are used when the destination is the Directory Server entry.

Syntax

```
#include "slapi-plugin.h"
typedef void (*winsync_pre_mod_cb)(void *cookie, const Slapi_Entry
*rawentry, Slapi_Entry *ad_entry, Slapi_Entry *ds_entry, Slapi_Mods
*smods, int *do_modify);
```

Parameters

This function takes the following parameters:

Table 14.22. winsync_pre_mod_cb Parameters

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the Windows Sync plug-in init function (if any) and passed to all plug-in functions.
<i>rawentry</i>	The unprocessed Active Directory entry, as it is read directly from Active Directory. This entry is read-only.
<i>ad_entry</i>	The processed Active Directory entry. This DN is set if the modify is against the Active Directory entry.
<i>ds_entry</i>	The entry to be added to the Directory Server. This DN is set if the modify is against the Directory Server entry.
<i>smods</i>	Pointer to an initialized Slapi_Mod. These contain the post-processing modifications. These modifications should be updated by the sync plug-in to perform any mappings or other changes.

Parameter	Description
<i>do_modify</i>	Indicates whether an operation will be performed on the entry. If there are changes to be synced to the entry or if the sync plug-in has changed any of the <i>smods</i> , then this value is true , meaning that an operation should be performed on the entry. If all of the <i>smods</i> were removed by the sync plug-in, meaning there is no operation to perform, then the value is false .

Returns

There are four possible returns:

- If modifications have been performed on an Active Directory user, this returns **6**.
- If modifications have been performed on an Active Directory group, this returns **7**.
- If modifications have been performed on a Directory Server user, this returns **8**.
- If modifications have been performed on a Directory Server group, this returns **9**.

14.37.7. winsync_search_params_cb

Allows configurable search parameters for searches of the Directory Server and Active Directory instances during synchronization.

Syntax

```
#include "slapi-plugin.h"
typedef void (*winsync_search_params_cb)(void *cookie, const char
*agmt_dn, char **base, int *scope, char **filter, char ***attrs,
LDAPControl ***serverctrls);
#define WINSYNC_PLUGIN_DIRSYNC_SEARCH_CB 2
#define WINSYNC_PLUGIN_PRE_AD_SEARCH_CB 3
#define WINSYNC_PLUGIN_PRE_DS_SEARCH_ENTRY_CB 4
#define WINSYNC_PLUGIN_PRE_DS_SEARCH_ALL_CB 5
```

- **WINSYNC_PLUGIN_DIRSYNC_SEARCH_CB 2** is called when the Win Sync code does the DirSync search of Active Directory looking for the initial list of entries during the init phase or for new changes during the incremental phase.
- **WINSYNC_PLUGIN_PRE_AD_SEARCH_CB 3** is called when the Win Sync code needs to search for an Active Directory entry to get more information from it during the sync process. This is used to get the *rawentry* or the *ad_entry* passed as a parameter to many of the Win Sync functions.
- **WINSYNC_PLUGIN_PRE_DS_SEARCH_ENTRY_CB 4** is called when the Win Sync code needs to search for a Directory Server entry to get more information from it during the sync process. This is used to get the *ds_entry* passed as a parameter to many of the Win Sync functions.

- **WINSYNC_PLUGIN_PRE_DS_SEARCH_ALL_CB 5** is called when the Win Sync code does the initial internal search of Directory Server to get all of the entries that will be synced to Active Directory.

Parameters

This function takes the following parameters:

Table 14.23. winsync_search_params_cb Parameters

Parameter	Description
<i>cookie</i>	Private data used by the plug-in. This is the value returned by the Windows Sync plug-in init function (if any) and passed to all plug-in functions.
<i>agmt_dn</i>	The original Active Directory base DN which is specified in the sync agreement.
<i>scope</i>	<p>The original scope of the search on the Active Directory server. This value is explicitly set. For example:</p> <pre>*scope = LDAP_SCOPE_SUBTREE;</pre>
<i>base</i>	<p>The base DN on the Directory Server to search for synchronization. To set this value, free the <i>base</i> first using slapi_ch_free_string() and allocate new memory using one of the slapi memory allocation functions. This value is then freed using slapi_ch_free_string() after use.</p>
<i>filter</i>	<p>The filter to use to search for entries in the Directory Server <i>base</i> . To set the filter, free it along with the <i>base</i> using slapi_ch_free_string(). For example:</p> <pre>slapi_ch_free_string(filter); *base = slapi_ch_strdup(" (objectclass=foobar)");</pre>
<i>attrs</i>	<p>Pointer to the Slapi_Attr structure representing the first attribute in the entry. This can be either NULL or a null-terminated array of strings. The attributes can be added using slapi_ch_array_add. For example:</p> <pre>slapi_ch_array_add(attrs, slapi_ch_strdup("myattr"));</pre> <p><i>attrs</i> are freed using slapi_ch_array_free, so the caller must own the memory.</p>

Parameter	Description
<i>serverctrls</i>	<p>Pointer to the LDAPControl* structure. This can be either NULL or a null-terminated array of controls. To define the LDAPControl, use slapi_add_control_ext:</p> <pre>slapi_add_control_ext(serverctrls , mynewctrl, 1 / add a copy /);</pre> <p><i>serverctrls</i> are freed with ldap_controls_free, so the caller must own memory.</p>

Returns

There are four possible returns:

- For a DirSync search, this returns **2**.
- To search the Active Directory subtree, this returns **3**.
- To search the Directory Server subtree, this returns **4**.
- To search the Directory Server from the base DN, this returns **5**.

PART IV. FUNCTION REFERENCE

This part contains reference information on Red Hat Directory Server (Directory Server) server plug-in API. The server plug-in API includes the above functions. Each chapter summarizes the frontend functions in a table followed by the function details.

CHAPTER 15. DISTRIBUTION ROUTINES

This chapter contains reference information on distribution routines.

Table 15.1. Distribution Routines

Function	Description
<code>distribution_plugin_entry_point()</code>	Allows for backend distribution.

15.1. DISTRIBUTION_PLUGIN_ENTRY_POINT()

Description

Backend distribution is the capability to span the LDAP subtree contents under a specified DIT node into multiple backends in the same server and/or database links to other servers. Under such a configuration, this function is responsible for deciding where the database or database link under the DIT node will be applied. This function will be called for every operation reaching a DIT node, including subtree search operations that are started above the node.

This function can only be called if the server has been configured to take advantage of such capability.

Returns

This function should return the index of the backend in the `mtn_be_names` table that is used to resolve the current operation. For search operations, `SLAPI_BE_ALL_BACKENDS` can be returned to specify that backends must be searched. The use of `SLAPI_BE_ALL_BACKENDS` for non-search operations is not supported and may give random results.

Syntax

```
#include "slapi-plugin.h"
int distribution_plugin_entry_point (Slapi_PBlock *pb, Slapi_DN
*target_dn, char **mtn_be_names, int be_count, Slapi_DN * node_dn);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Pointer to the parameter block of the operation.
<i>target_dn</i>	Pointer to the target DN of the operation.
<i>mtn_be_names</i>	Pointer to the list of names of backends declared for this node.
<i>be_count</i>	The number of backends declared under a specified DIT node.

<i>node_dn</i>	DN of the node where the distribution function is set.
----------------	--

CHAPTER 16. FUNCTIONS FOR ACCESS CONTROL

This chapter contains reference information on access control routines.

Table 16.1. Access Control Routines

Function	Description
<code>slapi_access_allowed()</code>	Determines if the user who is requesting the current operation has the access rights to perform an operation on a given entry, attribute, or value.
<code>slapi_acl_check_mods()</code>	Determines if a user has the rights to perform the specified modifications on an entry.
<code>slapi_acl_verify_aci_syntax()</code>	Determines whether the access control items (ACIs) on an entry are valid.

16.1. SLAPI_ACCESS_ALLOWED()

Description

Call this function to determine if a user has access rights to a specified entry, attribute, or value. The function performs this check for users who request the operation that invokes this plug-in.

For example, suppose you are writing a pre-operation plug-in for the add operation. You can call this function to determine if users have the proper access rights before they can add an entry to the directory.

As part of the process of determining if the user has access rights, the function does the following:

- Checks to see if the user requesting the operation is the root DN.

If so, the function returns **LDAP_SUCCESS**. (The root DN has permission to perform any operation.)

- Gets information about the operation being requested, the connection to the client, and the backend database where directory information is stored.
 - If for some reason the function cannot determine which operation is being requested, the function returns **LDAP_OPERATIONS_ERROR**.
 - If no connection to a client exists (in other words, if the request for the operation was made by the server or its backend), the function returns **LDAP_SUCCESS**. (The server and its backend are not restricted by access control lists.)
 - If the backend database is read-only and the request is checking for write access (**SLAPI_ACL_WRITE**), the function returns **LDAP_UNWILLING_TO_PERFORM**.
- Determines if the user requesting the operation is attempting to modify his or her own entry.

ACLs can be set up to allow users the rights to modify their own entries. The `slapi_access_allowed()` function checks for this condition.

The caller must ensure that the backend specified in the pblock is set prior to calling this function. For example:

```
be = slapi_be_select( slapi_entry_get_sdn_const( seObjectEntry ));
if ( NULL == be ) {
    cleanup("backend selection failed for entry: \"%s\"\n",
        szObjectDN);
    slapi_send_ldap_result( pb, LDAP_NO_SUCH_OBJECT, NULL,
        " Object could not be found", 0, NULL );
    return( SLAPI_PLUGIN_EXTENDED_SENT_RESULT );
}
slapi_pblock_set( pb, SLAPI_BACKEND, be );
nAccessResult = slapi_access_allowed( pb, seObjectEntry, "*", bval,
    SLAPI_ACL_DELETE);
```

Determines if a user (who is requesting the current operation) has the access rights to perform an operation on a given entry, attribute, or value.

Syntax

```
#include "slapi-plugin.h"
int slapi_access_allowed( Slapi_PBlock *pb, Slapi_Entry *e, char *attr,
    struct berval *val, int access );
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block passed into this function.
<i>e</i>	Entry for which you want to check the access rights.
<i>attr</i>	Attribute for which you want to check the access rights.
<i>val</i>	Pointer to the berval structure containing the value for which you want to check the access rights.
<i>access</i>	Type of access rights for which you want to check; for example, to check for write access, pass SLAPI_ACL_WRITE as the value of this argument.

The value of the **access** argument can be one of the following:

SLAPI_ACL_ADD	Permission to add a specified entry.
SLAPI_ACL_COMPARE	Permission to compare the specified values of an attribute in an entry.
SLAPI_ACL_DELETE	Permission to delete a specified entry.
SLAPI_ACL_READ	Permission to read a specified attribute.
SLAPI_ACL_SEARCH	Permission to search on a specified attribute or value.
SLAPI_ACL_WRITE	Permission to write a specified attribute or value or permission to rename a specified entry.

Returns

This function returns one of the following values:

- **LDAP_SUCCESS** if the user has the specified rights to the entry, attribute, or value.
- **LDAP_INSUFFICIENT_ACCESS** if the user does not have the specified rights to the entry, attribute, or value.

If a problem occurs during processing, the function will return one of the following error codes:

LDAP_OPERATIONS_ERROR	An error occurred while executing the operation. This error can occur if, for example, the type of access rights you've specified are not recognized by the server (in other words, you did not pass a value from the previous table).
LDAP_INVALID_SYNTAX	Invalid syntax was specified. This error can occur if the ACL associated with an entry, attribute, or value uses the wrong syntax.
LDAP_UNWILLING_TO_PERFORM	The Directory Server is unable to perform the specified operation. This error can occur if, for example, you are requesting write access to a read-only database.

16.2. SLAPI_ACL_CHECK_MODS()

Description

Call this function to determine if a user has access rights to modify the specified entry. The function performs this check for users who request the operation that invokes this plug-in.

Suppose you are writing a database plug-in. You can call this function to determine if users have the proper access rights before they can add, modify, or delete entries from the database.

As part of the process of determining if the user has access rights, the `slapi_acl_check_mods()` function does the following:

- Checks if access control for the directory is disabled (for example, if the `dse.ldif` file contains the directive **access control off**).

If access control is disabled, the function returns **LDAP_SUCCESS**.

- For each value in each attribute specified in the LDAPMod array, the function determines if the user has permissions to write to that value. Essentially, the function calls `slapi_acl_check_mods()` with **SLAPI_ACL_WRITE** as the access right to check.
 - If for some reason the function cannot determine which operation is being requested, the function returns **LDAP_OPERATIONS_ERROR**.
 - If no connection to a client exists (in other words, if the request for the operation was made by the server or its backend), the function returns **LDAP_SUCCESS**. (The server and its backend are not restricted by access control lists.)
 - If the backend database is read-only and the request is checking for write access (**SLAPI_ACL_WRITE**), the function returns **LDAP_UNWILLING_TO_PERFORM**.

Syntax

```
#include "slapi-plugin.h"
int slapi_acl_check_mods( Slapi_PBlock *pb, Slapi_Entry *e, LDAPMod
**mods, char **errbuf );
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block passed into this function.
<i>e</i>	Entry for which you want to check the access rights.
<i>mods</i>	Array of LDAPMod structures that represent the modifications to be made to the entry.
<i>errbuf</i>	Pointer to a string containing an error message if an error occurs during the processing of this function.

Returns

This function returns one of the following values:

- **LDAP_SUCCESS** if the user has write permission to the values in the specified attributes.
- **LDAP_INSUFFICIENT_ACCESS** if the user does not have write permission to the values of the specified attribute.
- If a problem occurs during processing, the function will return one of the following error codes:

LDAP_OPERATIONS_ERROR	An error occurred while executing the operation.
LDAP_INVALID_SYNTAX	Invalid syntax was specified. This error can occur if the ACL associated with an entry, attribute, or value uses the wrong syntax.
LDAP_UNWILLING_TO_PERFORM	The Directory Server is unable to perform the specified operation. This error can occur if, for example, you are requesting write access to a read-only database.

Memory Concerns

You must free the *errbuf* buffer by calling [slapi_ch_free\(\)](#) when you are finished using the error message.

See Also

- [slapi_access_allowed\(\)](#)
- [slapi_ch_free\(\)](#)

16.3. SLAPI_ACL_VERIFY_ACI_SYNTAX()

Description

Determines whether the access control items (ACIs) on an entry are valid.

Syntax

```
#include "slapi-plugin.h"
int slapi_acl_verify_aci_syntax (Slapi_Entry *e, char **errbuf);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry for which you want to check the ACIs.
<i>errbuf</i>	Pointer to the error message returned if the ACI syntax is invalid.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs.

Memory Concerns

You must free the *errbuf* buffer by calling [slapi_ch_free\(\)](#) when you are finished using the error message.

See Also

[slapi_ch_free\(\)](#)

CHAPTER 17. FUNCTIONS FOR INTERNAL OPERATIONS AND PLUG-IN CALLBACK

This chapter contains reference information on routines for internal operations and plug-in callbacks. These functions can be used for internal operations based on DN as well as on unique ID. These functions should be used by all new plug-ins, and, preferably, old plug-ins should be changed to use them to take advantage of new plug-in configuration capabilities and to use an extensible interface.

Table 17.1. Internal Operations and Plug-in Callback Routines

Function	Description
<code>slapi_add_internal_pb()</code>	Performs an LDAP add operation based on a parameter block to add a new directory entry.
<code>slapi_delete_internal_pb()</code>	Performs an LDAP delete operation based on a parameter block to remove a directory entry.
<code>slapi_free_search_results_internal()</code>	Frees search results.
<code>slapi_modify_internal_pb()</code>	Performs an LDAP modify operation based on a parameter block to modify a directory entry.
<code>slapi_modrdn_internal_pb()</code>	Performs an LDAP modify RDN operation based on a parameter block to rename a directory entry.
<code>slapi_search_internal_callback_pb()</code>	Performs an LDAP search operation based on a parameter block to search the directory.
<code>slapi_search_internal_get_entry()</code>	Performs an internal search operation to read one entry.
<code>slapi_search_internal_pb()</code>	Performs an LDAP search operation based on a parameter block to search the directory.

17.1. SLAPI_ADD_INTERNAL_PB()

Description

The function performs an internal LDAP add operation based on a parameter block. The parameter block should be initialized by calling `slapi_add_internal_set_pb()` or `slapi_add_entry_internal_set_pb()`.

Syntax

```
#include "slapi-plugin.h"
int slapi_add_internal_pb ( Slapi_PBlock *pb );
```

Parameters

This function takes the following parameter:

<i>pb</i>	A parameter block that has been initialized using slapi_add_internal_set_pb() .
-----------	--

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs. If -1 is returned, the **SLAPI_PLUGIN_INTOP_RESULT** field of the parameter block should be consulted to determine the precise LDAP result code.

Memory Concerns

None of the parameters that are passed **slapi_add_internal_set_pb()** are altered or consumed by this function. The entry parameter that is passed to **slapi_add_entry_internal_set_pb()** is consumed by a successful call to this function.

17.2. SLAPI_DELETE_INTERNAL_PB()

Description

This function performs an internal delete operation based on a parameter block to remove a directory entry. The parameter block should be initialized by calling **slapi_delete_internal_set_pb()**.

Syntax

```
#include "slapi-plugin.h"
int slapi_delete_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

<i>pb</i>	A parameter block that has been initialized using slapi_delete_internal_set_pb() .
-----------	---

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs. If -1 is returned, the **SLAPI_PLUGIN_INTOP_RESULT** field of the parameter block should be consulted to determine the precise LDAP result code.

Memory Concerns

None of the parameters that are passed to **slapi_delete_internal_set_pb()** are altered or consumed by this function.

17.3. SLAPI_FREE_SEARCH_RESULTS_INTERNAL()

Description

Frees search results returned by the [slapi_search_internal_pb\(\)](#) and [slapi_search_internal_callback_pb\(\)](#) functions.

This function must be called when you are finished with the entries before freeing the pblock.

Syntax

```
#include "slapi-plugin.h"
void slapi_free_search_results_internal(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block returned by the slapi_search_internal_pb() and slapi_search_internal_callback_pb() functions.
-----------	---

17.4. SLAPI_MODIFY_INTERNAL_PB()

Description

This function performs an internal modify operation based on a parameter block. The parameter block should be initialized by calling **slapi_modify_internal_set_pb()**.

Syntax

```
#include "slapi-plugin.h"
int slapi_modify_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

<i>pb</i>	A parameter block that has been initialized using slapi_modify_internal_set_pb() .
-----------	---

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs. If -1 is returned, the **SLAPI_PLUGIN_INTOP_RESULT** field of the parameter block should be consulted to determine the precise LDAP result code.

Memory Concerns

None of the parameters that are passed to `slapi_modify_internal_set_pb()` are altered or consumed by this function.

17.5. SLAPI_MODRDN_INTERNAL_PB()

Description

This function performs an internal modify RDN operation based on a parameter block to rename a directory entry. The parameter block should be initialized by calling `slapi_rename_internal_set_pb()`.

Syntax

```
#include "slapi-plugin.h"
int slapi_modrdn_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

<i>pb</i>	A parameter block that has been initialized using <code>slapi_rename_internal_set_pb()</code> .
-----------	---

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs. If -1 is returned, the `SLAPI_PLUGIN_INTOP_RESULT` field of the parameter block should be consulted to determine the precise LDAP result code.

Memory Concerns

None of the parameters that are passed to `slapi_modrdn_internal_set_pb()` are altered or consumed by this function.

17.6. SLAPI_SEARCH_INTERNAL_CALLBACK_PB()

Description

Performs an LDAP search operation based on a parameter block to search the directory. Unlike `slapi_search_internal_pb()`, this function allows you to specify callback functions that are invoked when the search operation finds matching entries or entries with referrals.

Syntax

```
#include "slapi-plugin.h"
int slapi_search_internal_callback_pb(Slapi_PBlock *pb, void
*callback_data, plugin_result_callback prc, plugin_search_entry_callback
psec, plugin_referral_entry_callback prec);
```

Parameters

This function takes the following parameters:

<i>pb</i>	A parameter block that has been initialized using <code>slapi_seq_internal_callback_set_pb()</code> .
<i>callback_data</i>	A pointer to arbitrary plug-in or operation-specific data that you would like to pass to your callback functions.
<i>prc</i>	Callback function that the server calls to send result codes. The function must have the prototype specified by plugin_result_callback .
<i>psec</i>	Callback function that the server calls when finding a matching entry in the directory. The function must have the prototype specified by plugin_search_entry_callback .
<i>prec</i>	Callback function that the server calls when finding an entry that contains LDAPv3 referrals. The function must have the prototype specified by plugin_referral_entry_callback .

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs. If -1 is returned, the **SLAPI_PLUGIN_INTOP_RESULT** field of the parameter block should be consulted to determine the precise LDAP result code.

Memory Concerns

The entries passed to the search entry callback function do not need to be freed. If you need to access an entry after returning from the callback function, call [slapi_entry_dup\(\)](#) to make a copy.

The referral URLs passed to the referral entry callback function do not need to be freed. If you need to access a referral string after returning from the callback function, call [slapi_ch_strdup\(\)](#) to make a copy.

You do not need to call [slapi_free_search_results_internal\(\)](#) after calling `slapi_search_internal_callback_pb`.

17.7. SLAPI_SEARCH_INTERNAL_GET_ENTRY()

Description

This function performs an internal search operation to read one entry; that is, it performs a

base object search. If an entry named by *dn* is found, the *ret_entry* pointer will be set to point to a copy of the entry that contains the attribute values specified by the *attrs* parameter.

Syntax

```
#include "slapi-plugin.h"
int slapi_search_internal_get_entry( Slapi_DN *dn, char ** attrs,
Slapi_Entry **ret_entry , void * component_identity)
```

Parameters

This function takes the following parameters:

<i>dn</i>	The DN of the entry to be read.
<i>attrs</i>	A NULL terminated array of attribute types to return from entries that match filter. If you specify a NULL , all attributes will be returned.
<i>ret_entry</i>	The address of a pointer to receive the entry if it is found.
<i>component_identity</i>	A plug-in or component identifier. This value can be obtained from the SLAPI_PLUGIN_IDENTITY field of the parameter block that is passed to your plug-in initialization function.

Returns

This function returns the LDAP result code for the search operation and the results of **slapi_search_internal_get_entry_ext**.

Memory Concerns

The returned entry (**ret_entry*) should be freed by calling [slapi_entry_free\(\)](#).

See Also

- [slapi_search_internal_pb\(\)](#)
- [slapi_entry_free\(\)](#)

17.8. SLAPI_SEARCH_INTERNAL_PB()

This function performs an internal LDAP search based on a parameter block to search the directory. The parameter block should be initialized by calling the **slapi_search_internal_set_pb()** function.

Syntax

```
#include "slapi-plugin.h"
int slapi_search_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

<i>pb</i>	A parameter block that has been initialized using slapi_search_internal_set_pb() .
-----------	---

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs. If -1 is returned, the **SLAPI_PLUGIN_INTOP_RESULT** field of the parameter block should be consulted to determine the precise LDAP result code.

Memory Concerns

[slapi_free_search_results_internal\(\)](#) should be called to dispose of any entires and other items that were allocated by a call to [slapi_search_internal_pb\(\)](#).

CHAPTER 18. FUNCTIONS FOR SETTING INTERNAL OPERATION FLAGS

This chapter contains reference information on routines for setting internal-operation flags.

Table 18.1. Internal Operation Flag Routines

Function	Description
slapi_add_entry_internal_set_pb()	Sets up a parameter block so that it can be used by slapi_add_internal_pb() for an internal add operation.
slapi_add_internal_set_pb()	Sets up a parameter block so that it can be used by slapi_add_internal_pb() for an internal add operation; the entry is constructed from a DN and a set of attributes.
slapi_delete_internal_set_pb()	Sets up a parameter block so that it can be used by slapi_delete_internal_pb() for an internal delete operation.
slapi_modify_internal_set_pb()	Sets up a parameter block so that it can be used by slapi_modify_internal_pb() for an internal modify operation.
slapi_rename_internal_set_pb()	Sets up a parameter block so that it can be used by slapi_modrdn_internal_pb() for an internal rename operation.
slapi_search_internal_set_pb()	Sets up a parameter block so that it can be used by slapi_search_internal_pb() for an internal search operation.
slapi_seq_internal_callback_pb()	Performs an internal sequential access operation.
slapi_seq_internal_set_pb()	Sets up a parameter block for use by slapi_seq_internal_callback_pb() for an internal sequential-access operation.

18.1. SLAPI_ADD_ENTRY_INTERNAL_SET_PB()

Description

This function populates parameters in the pblock structure so that it can be used by [slapi_add_internal_pb\(\)](#) for an internal add operation.

Syntax

```
#include "slapi-plugin.h"
void slapi_add_entry_internal_set_pb(Slapi_PBlock *pb, Slapi_Entry *e,
LDAPControl **controls, Slapi_ComponentId *plugin_identity, int
```

```
operation_flags);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block populated with add parameters.
<i>e</i>	Entry to be added.
<i>controls</i>	List of controls associated with the operation.
<i>plugin_identity</i>	Plug-in identity; a cookie that identifies the plug-in to the Directory Server during an internal operation. This cookie is used by the server to retrieve the plug-in configuration in order to determine whether to allow the operation and which actions to take during the operation processing. Plug-in identity is passed to the plug-in initialization function in the SLAPI_PLUGIN_IDENTITY <i>pblock</i> parameter. A plug-in must save this information and pass it to every internal operation issued by the plug-in.
<i>operation_flags</i>	Actions taken during operation processing.

18.2. SLAPI_ADD_INTERNAL_SET_PB()

Description

This function sets up a parameter block so that it can be used by [slapi_add_internal_pb\(\)](#) for an internal add operation. This function is similar to [slapi_add_entry_internal_set_pb\(\)](#) except that it constructs the entry from a DN and a set of attributes. The function sets pblock to contain the following data:

- **SLAPI_TARGET_DN** set to DN of the new entry.
- **SLAPI_CONTROLS_ARG** set to request controls, if present.
- **SLAPI_ADD_ENTRY** set to Slapi_Entry to add.

Syntax

```
#include "slapi-plugin.h"
int slapi_add_internal_set_pb(Slapi_PBlock *pb, const Slapi_DN *dn,
LDAPMod **attrs, LDAPControl **controls,
Slapi_ComponentId *plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block populated with add parameters.
<i>dn</i>	Entry DN.
<i>attrs</i>	Entry attributes.
<i>controls</i>	List of controls associated with the operation.
<i>plugin_identity</i>	Plug-in identity; a cookie that identifies the plug-in to the Directory Server during an internal operation. This cookie is used by the server to retrieve the plug-in configuration in order to determine whether to allow the operation and which actions to take during the operation processing. Plug-in identity is passed to the plug-in initialization function in the SLAPI_PLUGIN_IDENTITY pblock parameter. A plug-in must save this information and pass it to every internal operation issued by the plug-in.
<i>operation_flags</i>	Actions taken during operation processing.

Returns

This function returns **LDAP_SUCCESS** or one of the LDAP error codes if the entry cannot be constructed from the specified attributes due to constraint violation.

18.3. SLAPI_DELETE_INTERNAL_SET_PB()

Description

This function populates pblock to contain data for use by [slapi_delete_internal_pb\(\)](#) for an internal delete operation.

For unique identifier-based operation:

- **SLAPI_TARGET_DN** set to the DN that allows to select the right backend.
- **SLAPI_TARGET_UNIQUEID** set to the unique ID of the entry.
- **SLAPI_CONTROLS_ARG** set request controls, if present.

For DN-based search:

- **SLAPI_TARGET_DN** set to the entry DN.
- **SLAPI_CONTROLS_ARG** set to request controls, if present.

Syntax

```
#include "slapi-plugin.h"
void slapi_delete_internal_set_pb (Slapi_PBlock *pb, const Slapi_DN *dn,
LDAPControl **controls,
    const char *uniqueid, Slapi_ComponentId *plugin_identity, int
operation_flags);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block populated with delete parameters.
<i>dn</i>	DN of the entry to be removed. For unique ID operation, this parameter is used to select the correct backend.
<i>controls</i>	List of controls associated with the operation.
<i>uniqueid</i>	Unique identifier of the entry to be removed. All directory entries contain a unique identifier. Unlike the distinguished name (DN), the unique identifier of an entry never changes, providing a good way to refer unambiguously to an entry in a distributed/replicated environment.
<i>plugin_identity</i>	Plug-in identity; a cookie that identifies the plug-in to the Directory Server during an internal operation. This cookie is used by the server to retrieve the plug-in configuration in order to determine whether to allow the operation and which actions to take during the operation processing. Plug-in identity is passed to the plug-in initialization function in the SLAPI_PLUGIN_IDENTITY pblock parameter. A plug-in must save this information and pass it to every internal operation issued by the plug-in.
<i>operation_flags</i>	Actions taken during operation processing.

18.4. SLAPI_MODIFY_INTERNAL_SET_PB()

Description

This function populates pblock to contain data for use by [slapi_modify_internal_pb\(\)](#) for an internal modify operation.

For unique ID-based operation:

- **SLAPI_TARGET_DN** set to the DN that allows to select the right backend.
- **SLAPI_TARGET_UNIQUEID** set to the unique ID of the entry.
- **SLAPI_MODIFY_MODS** set to the mods.
- **SLAPI_CONTROLS_ARG** set to request controls, if present.

For DN-based search:

- **SLAPI_TARGET_DN** set to the entry DN.
- **SLAPI_MODIFY_MODS** set to the mods.
- **SLAPI_CONTROLS_ARG** set to request controls, if present.

Syntax

```
#include "slapi-plugin.h"
void slapi_modify_internal_set_pb(Slapi_PBlock *pb, const Slapi_DN *dn,
LDAPMod **mods,
LDAPControl **controls, const char *uniqueid, Slapi_ComponentId
*plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block populated with modify parameters.
<i>dn</i>	DN of the entry to be modified. For unique ID operation, this parameter is used to select the correct backend.
<i>mods</i>	Modifications to be applied to the entry.
<i>controls</i>	List of controls associated with the operation.
<i>uniqueid</i>	Unique identifier of the entry to be modified. All directory entries contain a unique identifier. Unlike the distinguished name (DN), the unique identifier of an entry never changes, providing a good way to refer unambiguously to an entry in a distributed/replicated environment.

<i>plugin_identity</i>	Plug-in identity; a cookie that identifies the plug-in to the Directory Server during an internal operation. This cookie is used by the server to retrieve the plug-in configuration in order to determine whether to allow the operation and which actions to take during the operation processing. Plug-in identity is passed to the plug-in initialization function in the SLAPI_PLUGIN_IDENTITY pblock parameter. A plug-in must save this information and pass it to every internal operation issued by the plug-in.
<i>operation_flags</i>	Actions taken during operation processing.

18.5. SLAPI_RENAME_INTERNAL_SET_PB()

Description

This function populates pblock with parameters for use by [slapi_modrdn_internal_pb\(\)](#) for an internal rename operation. The function sets the parameter block to contain the following data.

Syntax

```
#include "slapi-plugin.h"
void slapi_rename_internal_set_pb(Slapi_PBlock *pb, const char *olddn,
const char *newrdn, const char *newsuperior, int deloldrdn,
LDAPControl **controls, const char *uniqueid, Slapi_ComponentId
*plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block populated with rename parameters.
<i>olddn</i>	DN of the entry to be renamed. For unique ID operation, this parameter is used to select the correct backend.
<i>newrdn</i>	New RDN of the entry.
<i>newsuperior</i>	New entry superior, moddn operation only.
<i>deloldrdn</i>	Specifies whether the old RDN should be removed or left as a non-DN attribute.
<i>controls</i>	List of controls associated with the operation.

<i>uniqueid</i>	Unique identifier of the entry to be renamed. All directory entries contain a unique identifier. Unlike the distinguished name (DN), the unique identifier of an entry never changes, providing a good way to refer unambiguously to an entry in a distributed/replicated environment.
<i>plugin_identity</i>	Plug-in identity; a cookie that identifies the plug-in to the Directory Server during an internal operation. This cookie is used by the server to retrieve the plug-in configuration in order to determine whether to allow the operation and which actions to take during the operation processing. Plug-in identity is passed to the plug-in initialization function in the SLAPI_PLUGIN_IDENTITY pblock parameter. A plug-in must save this information and pass it to every internal operation issued by the plug-in.
<i>operation_flags</i>	Actions taken during operation processing.

For unique ID-based operation:

- **SLAPI_TARGET_DN** set to the DN that allows to select the right backend.
- **SLAPI_TARGET_UNIQUEID** set to the uniqueid of the entry.
- **SLAPI_MODRDN_NEWRDN** set to the new RDN of the entry.
- **SLAPI_MODRDN_DELOLDRDN** indicates whether the old RDN should be kept in the entry.
- **SLAPI_CONTROLS_ARG** set to request controls, if present.

For DN-based search:

- **SLAPI_TARGET_DN** set to the entry DN.
- **SLAPI_MODRDN_NEWRDN** set to the new RDN of the entry.
- **SLAPI_MODRDN_DELOLDRDN** indicates whether the old RDN should be kept in the entry.
- **SLAPI_CONTROLS_ARG** set to request controls, if present.

18.6. SLAPI_SEARCH_INTERNAL_SET_PB()

This function sets up the parameter block, for subsequent use by [slapi_search_internal_pb\(\)](#), to contain the following data for an internal search operation.

For unique ID-based search:

- **SLAPI_TARGET_DN** set to the DN that allows to select the right backend.
- **SLAPI_TARGET_UNIQUEID** set to the unique ID of the entry.

For DN-based search:

- **SLAPI_TARGET_DN** set to the search base.
- **SLAPI_SEARCH_SCOPE** set to the search scope.
- **SLAPI_SEARCH_STRFILTER** set to the search filter.
- **SLAPI_CONTROLS_ARG** set to request controls, if present.
- **SLAPI_SEARCH_ATTRS** set to the list of attributes to return.
- **SLAPI_SEARCH_ATTRSONLY** indicates whether attribute values should be returned.

Syntax

```
#include "slapi-plugin.h"
void slapi_search_internal_set_pb(Slapi_PBlock *pb, const char *base, int
scope, const char *filter, char **attrs,
int attrsonly, LDAPControl **controls, const char *uniqueid,
Slapi_ComponentId *plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block that is populated with search parameters.
<i>base</i>	Search base.
<i>scope</i>	Search scope (LDAP_SCOPE_SUBTREE , etc.).
<i>filter</i>	Search filter.
<i>attrs</i>	Attributes to be returned.
<i>attrsonly</i>	Flag specifying whether to return just attribute names or names and values.
<i>controls</i>	List of controls associated with the operation.

<i>uniqueid</i>	Unique identifier of the entry. Non-NULL value indicates unique ID-based search. In this case, scope and filter are ignored; however, base is still required and is used to select the correct backend. All directory entries contain a unique identifier. Unlike the distinguished name (DN), the unique identifier of an entry never changes, providing a good way to refer unambiguously to an entry in a distributed/replicated environment.
<i>plugin_identity</i>	Plug-in identity; a cookie that identifies the plug-in to the Directory Server during an internal operation. This cookie is used by the server to retrieve the plug-in configuration in order to determine whether to allow the operation and which actions to take during the operation processing. Plug-in identity is passed to the plug-in initialization function in the SLAPI_PLUGIN_IDENTITY pblock parameter. A plug-in must save this information and pass it to every internal operation issued by the plug-in.
<i>operation_flags</i>	Actions taken during operation processing.

Memory Concerns

The controls passed with `slapi_search_internal_set_pb()` *must* be an allocated array. Additionally, this array *must* be freed by `slapi_pblock_destroy()`.

If the user passes memory allocated on the stack or frees the controls himself, then when `slapi_pblock_destroy()` is called, the function can double-free the memory or corrupt the memory structures. This potentially leads to segfaults or other problems when the allocated memory is taken by any `Slapi_*` function.

18.7. SLAPI_SEQ_INTERNAL_CALLBACK_PB()

Description

This function performs internal sequential access operation.

Syntax

```
#include "slapi-plugin.h"
int slapi_seq_internal_callback_pb(Slapi_PBlock *pb, void *callback_data,
    plugin_result_callback res_callback, plugin_search_entry_callback
    srch_callback, plugin_referral_entry_callback ref_callback);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block initialized with operation parameters. The easiest way to provide required parameters is by calling slapi_seq_internal_callback_pb() function. Parameters can also be set directly.
<i>callback_data</i>	Data passed to the callback functions.
<i>res_callback</i>	Function called once the search is complete.
<i>srch_callback</i>	Function called for each entry returned.
<i>ref_callback</i>	Function called for each referral returned.

Returns

This function returns 0 on success, -1 on error.

18.8. SLAPI_SEQ_INTERNAL_SET_PB()

Description

This function sets up pblock for use by [slapi_seq_internal_callback_pb\(\)](#) for an internal, sequential-access operation; the function sets up the parameter block contain the following data:

- **SLAPI_SEARCH_TARGET** set to the search base.
- **SLAPI_ORIGINAL_TARGET_DN** preserves the DN that was sent from the client (this DN is normalized as it is processed by **SLAPI_SEARCH_TARGET**); this value is read-only.
- **SAPI_SEQ_TYPE** set to the sequential-access type (**SLAPI_SEQ_FIRST**, **SLAPI_SEQ_NEXT**, and so on.)
- **SLAPI_SEQ_ATTRNAME** defines attribute value assertion relative to which access is performed.
- **SLAPI_SEQ_VAL** defines attribute value assertion relative to which access is performed.
- **SLAPI_CONTROLS_ARG** set to request controls, if present.
- **SLAPI_SEARCH_ATTRS** set to the list of attributes to return.
- **SLAPI_SEARCH_ATTRSONLY** indicates whether attribute values should be returned.

Syntax

```
#include "slapi-plugin.h"
void slapi_seq_internal_set_pb(Slapi_PBlock *pb, char *ibase, int type,
char *attrname, char *val,
char **attrs, int attrsonly, LDAPControl **controls, Slapi_ComponentId
*plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block initialized with operation parameters. The easiest way to provide required parameters is by calling slapi_seq_internal_callback_pb() function. Parameters can also be set directly.
<i>callback_data</i>	Data passed to the callback functions.
<i>res_callback</i>	Function called once the search is complete.
<i>srch_callback</i>	Function called for each entry returned.
<i>ref_callback</i>	Function called for each referral returned.

CHAPTER 19. FUNCTIONS FOR HANDLING ATTRIBUTES

This chapter contains reference information on attribute routines.

Table 19.1. Attribute Routines

Function	Description
<code>slapi_attr_add_value()</code>	Adds a value to an attribute.
<code>slapi_attr_basetype()</code>	Returns the base type of an attribute.
<code>slapi_attr_dup()</code>	Duplicates an attribute.
<code>slapi_attr_first_value()</code>	Gets the first value of an attribute.
<code>slapi_attr_flag_is_set()</code>	Determines if certain flags are set.
<code>slapi_attr_free()</code>	Frees an attribute.
<code>slapi_attr_get_berval_copy()</code>	Puts the values contained in an attribute into an array of berval structures.
<code>slapi_attr_get_flags()</code>	Gets the flags associated with an attribute.
<code>slapi_attr_get_numvalues()</code>	Puts the count of values of an attribute into an integer.
<code>slapi_attr_get_oid_copy()</code>	Searches for an attribute type and gives its OID string.
<code>slapi_attr_get_type()</code>	Gets the name of the attribute type.
<code>slapi_attr_get_valueset()</code>	Copies attribute values into a valueset.
<code>slapi_attr_init()</code>	Initializes an empty attribute.
<code>slapi_attr_new()</code>	Creates a new attribute.
<code>slapi_attr_next_value()</code>	Gets the next value of an attribute.
<code>slapi_attr_set_valueset()</code>	Initializes a valueset in a <code>Slapi_Attr</code> structure from a specified <code>Slapi_ValueSet</code> structure.
<code>slapi_attr_syntax_normalize()</code>	Returns a copy of the normalized attribute types.

Function	Description
slapi_attr_type2plugin()	Gets information about the plug-in responsible for handling an attribute type.
slapi_attr_type_cmp()	Compares two attributes.
slapi_attr_types_equivalent()	Compares two attribute names to determine if they represent the same attribute.
slapi_attr_value_cmp()	Compares two attribute values.
slapi_attr_value_find()	Determines if an attribute contains a given value.
Section 19.23, "slapi_valueset_set_from_smod()"	Adds the changes in a modification to a valueset.
slapi_valueset_set_valueset()	Initializes a Slapi_ValueSet structure from another Slapi_ValueSet structure.

19.1. SLAPI_ATTR_ADD_VALUE()

Description

Adds a value to an attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_add_value(Slapi_Attr *a, const Slapi_Value *v);
```

Parameters

This function takes the following parameters:

<i>a</i>	The attribute that will contain the values.
<i>v</i>	Values to be added to the attribute.

Returns

This function always returns 0.

See Also

- [slapi_attr_first_value\(\)](#)
- [slapi_attr_next_value\(\)](#)
- [slapi_attr_get_numvalues\(\)](#)

- [slapi_attr_value_cmp\(\)](#)
- [slapi_attr_value_find\(\)](#)

19.2. SLAPI_ATTR_BASETYPE()

Description

This function returns the base type of an attribute (for example, if given **cn;lang-jp**, returns **cn**).

Syntax

```
#include "slapi-plugin.h"
char *slapi_attr_basetype( char *type, char *buf, size_t bufsiz );
```

Parameters

This function takes the following parameters:

<i>type</i>	Attribute type from which you want to get the base type.
<i>buf</i>	Buffer to hold the returned base type.
<i>bufsiz</i>	Size of the buffer.

Returns

This function returns **NULL** if the base type fits in the buffer. If the base type is longer than the buffer, the function allocates memory for the base type and returns a pointer to it.

Memory Concerns

You should free the returned base type when done by calling [slapi_attr_basetype\(\)](#).

See Also

- [slapi_attr_type2plugin\(\)](#)
- [slapi_attr_get_type\(\)](#)
- [slapi_attr_type_cmp\(\)](#)
- [slapi_attr_types_equivalent\(\)](#)

19.3. SLAPI_ATTR_DUP()

Description

Use this function to make a copy of an attribute.

Syntax


```
#include "slapi-plugin.h"
Slapi_Attr *slapi_attr_dup(const Slapi_Attr *attr);
```

Parameters

This function takes the following parameters:

<i>attr</i>	The attribute to be duplicated.
-------------	---------------------------------

Returns

This function returns the newly created copy of the attribute.

Memory Concerns

You must free the returned attribute using [slapi_attr_free\(\)](#).

See Also

- [slapi_attr_new\(\)](#)
- [slapi_attr_init\(\)](#)
- [slapi_attr_free\(\)](#)

19.4. SLAPI_ATTR_FIRST_VALUE()

Description

Use this function to get the first value of an attribute. This is part of a set of functions to enumerate over an `Slapi_Attr` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_first_value( Slapi_Attr *a, Slapi_Value **v );
```

Parameters

This function takes the following parameters:

<i>a</i>	Attribute containing the desired value.
<i>v</i>	Holds the first value of the attribute.

Returns

This function returns one of the following values:

- 0, which is the index of the first value.
- -1 if the attribute (*a*) is NULL or if the value (*v*) parameter has no value.

See Also

[slapi_attr_next_value\(\)](#) [slapi_attr_get_numvalues\(\)](#)

19.5. SLAPI_ATTR_FLAG_IS_SET()

Description

This function determines if certain flags are set for a particular attribute. These flags can identify an attribute as a single-valued attribute, an operational attribute, or as a read-only attribute.

Syntax

```
#include "slapi-plugin.h"int slapi_attr_flag_is_set( const Slapi_Attr
*attr, unsigned long flag );
```

Parameters

This function takes the following parameters:

<i>attr</i>	Attribute that you want to check.
<i>flag</i>	Flag to check in the attribute.

The value of the **flag** argument can be one of the following:

SLAPI_ATTR_FLAG_SINGLE	Flag that determines if the attribute is single-valued.
SLAPI_ATTR_FLAG_OPATTR	Flag that determines if the attribute is an operational attribute.
SLAPI_ATTR_FLAG_READONLY	Flag that determines if the attribute is read-only.

Returns

This function returns one of the following values:

- 1 if the specified flag is set.
- 0 if the specified flag is not set.

See Also

[slapi_attr_get_flags\(\)](#)

19.6. SLAPI_ATTR_FREE()

Description

Use this function to free an attribute when you are finished with it.

Syntax

```
#include "slapi-plugin.h"
void slapi_attr_free( Slapi_Attr **a );
```

Parameters

This function takes the following parameters:

<i>a</i>	Attribute to be freed.
----------	------------------------

See Also

- [slapi_attr_new\(\)](#)
- [slapi_attr_init\(\)](#)
- [slapi_attr_dup\(\)](#)

19.7. SLAPI_ATTR_GET_BERVALS_COPY()

Description

This function copies the values from an attribute into an array of berval structure pointers.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_berval_copy( Slapi_Attr *a, struct berval ***vals );
```

Parameters

This function takes the following parameters:

<i>a</i>	Attribute that contains the desired values.
<i>vals</i>	Pointer to an array of berval structure pointers to hold the desired values.

Returns

This function returns one of the following values:

- 0 if values are found.
- -1 if null.

Memory Concerns

You should free this array using **ber_bvecfree** from the LDAP SDK for C.

19.8. SLAPI_ATTR_GET_FLAGS()

Description

This function gets the flags associated with the specified attribute. These flags can identify an attribute as a single-valued attribute, an operational attribute, or as a read-only attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_flags( const Slapi_Attr *attr, unsigned long *flags );
```

Parameters

This function takes the following parameters:

<i>attr</i>	Attribute for which you want to get the flags.
<i>flags</i>	When you call slapi_attr_get_flags() , this parameter is set to a pointer to the flags of the specified attribute. Do not free the flags; the flags are part of the actual data in the attribute, not a copy of the data.

To determine which flags have been set, you can bitwise AND the value of the *flags* argument with one or more of the following:

SLAPI_ATTR_FLAG_SINGLE	Flag that determines if the attribute is single-valued.
SLAPI_ATTR_FLAG_OPATTR	Flag that determines if the attribute is an operational attribute.
SLAPI_ATTR_FLAG_READONLY	Flag that determines if the attribute is read-only.

Returns

This function returns 0 if successful.

See Also

[slapi_attr_flag_is_set\(\)](#)

19.9. SLAPI_ATTR_GET_NUMVALUES()

Description

This function counts the number of values in an attribute and places that count in an integer.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_numvalues( const Slapi_Attr *a, int *numValues);
```

Parameters

This function takes the following parameters:

<i>a</i>	Attribute containing the values to be counted.
<i>numValues</i>	Integer to hold the counted values.

Returns

This function always returns 0.

See Also

- [slapi_attr_first_value\(\)](#)
- [slapi_attr_next_value\(\)](#)

19.10. SLAPI_ATTR_GET_OID_COPY()

Description

This function replaces the deprecated function, **slapi_attr_get_oid**. Use this function to search the syntaxes for an attribute type's OID and return a copy of it's OID string.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_oid_copy( const Slapi_Attr *attr, char **oidp );
```

Parameters

This function takes the following parameters:

<i>attr</i>	Attribute that contains the desired type.
<i>oidp</i>	Destination string of the copied attribute type OID.

Returns

This function returns one of the following values:

- 0 if the attribute type is found.
- -1 if it is not.

Memory Concerns

You should free this string using [slapi_ch_free\(\)](#).

19.11. SLAPI_ATTR_GET_TYPE()

Description

Gets the name of the attribute type from a specified attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_type( Slapi_Attr *attr, char **type );
```

Parameters

This function takes the following parameters:

<i>attr</i>	Attribute for which you want to get the type.
<i>type</i>	When you call slapi_attr_get_type() , this parameter is set to a pointer to the type of the specified attribute. Do not free this attribute type; the type is part of the actual data in the attribute, not a copy of the data.

Returns

This function returns 0 if successful.

See Also

- [slapi_attr_type2plugin\(\)](#)
- [slapi_attr_type_cmp\(\)](#)
- [slapi_attr_types_equivalent\(\)](#)
- [slapi_attr_basetype\(\)](#)

19.12. SLAPI_ATTR_GET_VALUESSET()

Description

Copies existing values contained in an attribute into a valueset.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_valueset(const Slapi_Attr *a, Slapi_ValueSet **vs);
```

Parameters

This function takes the following parameters:

<i>a</i>	Attribute containing the values to be placed into a valueset.
<i>vs</i>	Receives values from the first parameter.

Returns

This function always returns 0.

See Also

- [slapi_entry_add_valueset\(\)](#)
- [slapi_valueset_new\(\)](#)
- [slapi_valueset_free\(\)](#)
- [slapi_valueset_init\(\)](#)
- [slapi_valueset_done\(\)](#)
- [slapi_valueset_add_value\(\)](#)
- [slapi_valueset_first_value\(\)](#)
- [slapi_valueset_next_value\(\)](#)
- [slapi_valueset_count\(\)](#)

19.13. SLAPI_ATTR_INIT()

Description

Use this function to initialize an empty attribute with an attribute type.

Syntax

```
#include "slapi-plugin.h"
Slapi_Attr *slapi_attr_init(Slapi_Attr *a, const char *type);
```

Parameters

This function takes the following parameters:

<i>a</i>	The empty attribute to be initialized.
<i>type</i>	Attribute type to be initialized.

Returns

This function returns the newly-initialized attribute.

See Also

- [slapi_attr_new\(\)](#)
- [slapi_attr_free\(\)](#)
- [slapi_attr_dup\(\)](#)

19.14. SLAPI_ATTR_NEW()

Description

Use this function to create an empty attribute.

Syntax

```
#include "slapi-plugin.h"
Slapi_Attr *slapi_attr_new( void );
```

Parameters

This function takes no parameters.

Returns

This function returns the newly-created attribute.

See Also

- [slapi_attr_free\(\)](#)
- [slapi_attr_dup\(\)](#)

19.15. SLAPI_ATTR_NEXT_VALUE()

Description

Use this function to get the next value of an attribute. The value of an attribute associated with an index is placed into a value. This is part of a set of functions to enumerate over a [Slapi_Attr](#) structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_next_value( Slapi_Attr *a, int hint, Slapi_Value **v );
```

Parameters

This function takes the following parameters:

<i>a</i>	Attribute contained the desired value.
<i>hint</i>	Index of the value to be returned.
<i>v</i>	Holds the value of the attribute.

Returns

This function returns one of the following values:

- **hint** plus **1** if the value is found.
- **-1** if null or if a value at **hint** is not found.

See Also

- [slapi_attr_first_value\(\)](#)
- [slapi_attr_get_numvalues\(\)](#)

19.16. SLAPI_ATTR_SET_VALUESSET()**Description**

This function initializes a valueset in a Slapi_Attr structure from a specified Slapi_ValueSet structure; the valueset in Slapi_Attr will be **vs*, not a copy.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_set_valueset(Slapi_Attr *a, const Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameters:

<i>a</i>	Pointer to the Slapi_Attr structure, the valueset of which you wish to set.
<i>vs</i>	Pointer to the Slapi_ValueSet structure from which you want to extract the values.

Returns

This function returns 0 unconditionally.

See Also

[slapi_valueset_set_valueset\(\)](#)

19.17. SLAPI_ATTR_SYNTAX_NORMALIZE()**Description**

Use this function to search the syntaxes for an attribute type and return its normalized form.

Syntax

```
#include "slapi-plugin.h"
char * slapi_attr_syntax_normalize( const char *s );
```

Parameters

This function takes the following parameters:

<i>s</i>	Attribute type for which you wish to search.
----------	--

Returns

This function returns the copy of the desired normalized attribute or a normalized copy of what was passed in.

Memory Concerns

You should free the returned string using [slapi_ch_free\(\)](#)

See Also

[slapi_ch_free\(\)](#)

19.18. SLAPI_ATTR_TYPE2PLUGIN()

Description

Gets a pointer to information about the syntax plug-in responsible for handling the specified attribute type. Syntax plug-ins are plug-ins that you can write to index and search for specific attribute types.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_type2plugin( const char *type, void **pi );
```

Parameters

This function takes the following parameters:

<i>type</i>	Type of attribute for which you want to get the plug-in.
<i>pi</i>	Pointer to the plug-in structure.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if the corresponding plug-in is not found.

See Also

- [slapi_attr_get_type\(\)](#)
- [slapi_attr_type_cmp\(\)](#)
- [slapi_attr_types_equivalent\(\)](#)
- [slapi_attr_basetype\(\)](#)

19.19. SLAPI_ATTR_TYPE_CMP()

Description

Compares two attribute types to determine if they are the same.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_type_cmp( const char *t1, const char *t2, int opt );
```

Parameters

This function takes the following parameters:

<i>t1</i>	Name of the first attribute type that you want to compare.
<i>t2</i>	Name of the second attribute type that you want to compare.
<i>opt</i>	One of the following values: <ul style="list-style-type: none"> • 0- Compare the types as-is. • 1- Compare only the base names of the types (for example, if the type is cn;lang-en, the function compares only the cn part of the type). • 2- Ignore any options in the second type that are not in the first type. For example, if the first type is cn and the second type is cn;lang-en, the lang-en option in the second type is not part of the first type. In this case, the function considers the two types to be the same.

Returns

This function returns one of the following values:

- 0 if the type names are equal.
- A non-zero value if the type names are not equal.

See Also

- [slapi_attr_type2plugin\(\)](#)
- [slapi_attr_get_type\(\)](#)
- [slapi_attr_types_equivalent\(\)](#)
- [slapi_attr_basetype\(\)](#)

19.20. SLAPI_ATTR_TYPES_EQUIVALENT()

Description

Compares two attribute names to determine if they represent the same attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_types_equivalent( const char *t1, const char *t2 );
```

Parameters

This function takes the following parameters:

<i>t1</i>	Pointer to the first attribute type that you want to compare.
<i>t2</i>	Pointer to the second attributed type that you want to compare.

Returns

This function returns one of the following values:

- 1 if **t1** and **t2** represent the same attribute.
- 0 if **t1** and **t2** do not represent the same attribute.

See Also

- [slapi_attr_add_value\(\)](#)
- [slapi_attr_first_value\(\)](#)
- [slapi_attr_next_value\(\)](#)
- [slapi_attr_get_numvalues\(\)](#)
- [slapi_attr_value_find\(\)](#)

19.21. SLAPI_ATTR_VALUE_CMP()

Description

Compares two values for a given attribute to determine if they are equal.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_value_cmp( const Slapi_Attr *attr, const struct berval *v1,
const struct berval *v2 );
```

Parameters

This function takes the following parameters:

<i>attr</i>	Attribute used to determine how these values are compared; for example, if the attribute contains case-insensitive strings, the strings are compared without regard to case.
<i>v1</i>	Pointer to the structure containing the first value that you want to compare.
<i>v2</i>	Pointer to the structure containing the second value that you want to compare.

Returns

This function returns one of the following values:

- 0 if the values are equal.
- -1 if the values are not equal.

See Also

- [slapi_attr_add_value\(\)](#)
- [slapi_attr_first_value\(\)](#)
- [slapi_attr_next_value\(\)](#)
- [slapi_attr_get_numvalues\(\)](#)
- [slapi_attr_value_find\(\)](#)

19.22. SLAPI_ATTR_VALUE_FIND()

Description

Determines if an attribute contains a given value.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_value_find( const Slapi_Attr *a, const struct berval *v );
```

Parameters

This function takes the following parameters:

<i>a</i>	Attribute that you want to check.
<i>v</i>	Pointer to the berval structure containing the value for which you want to search.

Returns

This function returns one of the following values:

- 0 if the attribute contains the specified value.
- -1 if the attribute does not contain the specified value.

See Also

- [slapi_attr_add_value\(\)](#)
- [slapi_attr_first_value\(\)](#)
- [slapi_attr_next_value\(\)](#)
- [slapi_attr_get_numvalues\(\)](#)
- [slapi_attr_value_cmp\(\)](#)

19.23. SLAPI_VALUESSET_SET_FROM_SMOD()

Adds the changes in a modification to a valueset. Use this function to create a value set that contains the changes from *smod*.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_set_from_smod(Slapi_ValueSet *vs, Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

<i>vs</i>	The valueset that will receive changes.
<i>smod</i>	Holds the changes to an attribute.

See Also

- [slapi_mods_init\(\)](#)
- [slapi_mods_free\(\)](#)
- [slapi_mods_done\(\)](#)

CHAPTER 20. FUNCTIONS FOR MANAGING BACKEND OPERATIONS

This chapter contains reference information on routines that help you deal with backends.

Table 20.1. Backend Routines

Function	Description
<code>slapi_be_addsuffix()</code>	Adds the specified suffix to the given backend and increments the backend's suffix count.
<code>slapi_be_delete_onexit()</code>	Sets the flag to denote that the backend will be deleted on exiting.
<code>slapi_be_exist()</code>	Checks if the backend that contains the specified DN exists.
<code>slapi_be_free()</code>	Frees memory and linked resources from the backend structure.
<code>slapi_be_get_instance_info()</code>	Gets the instance information of the specified backend.
<code>slapi_be_get_name()</code>	Returns the name of the specified backend.
<code>slapi_be_get_readonly()</code>	Indicates if the database associated with the backend is in read-only mode.
<code>slapi_be_getentrypoint()</code>	Sets pointer to a callback function that corresponds to the specified entry point into a given backend.
<code>slapi_be_getsuffix()</code>	Returns the $n+1$ suffix associated with the specified backend.
<code>slapi_be_gettype()</code>	Returns the type of the backend.
<code>slapi_be_is_flag_set()</code>	Checks if a flag is set in the backend configuration.
<code>slapi_be_issuffix()</code>	Verifies that the specified suffix matches a registered backend suffix.
<code>slapi_be_logchanges()</code>	Indicates if the changes applied to the backend should be logged in the changelog.
<code>slapi_be_new()</code>	Creates a new backend structure, allocates memory for it, and initializes values for relevant parameters.

Function	Description
<code>slapi_be_private()</code>	Verifies if the backend is private.
<code>slapi_be_select()</code>	Finds the backend that should be used to service the entry with the specified DN.
<code>slapi_be_select_by_instance_name()</code>	Find the backend used to service the database.
<code>slapi_be_set_flag()</code>	Sets the specified flag in the backend.
<code>slapi_be_set_instance_info()</code>	Sets the instance information of the specified backend with given data.
<code>slapi_be_set_readonly()</code>	Sets a flag to denote that the backend is meant to be read-only.
<code>slapi_be_setentrypoint()</code>	Sets the entry point in the backend to the specified function.
<code>slapi_get_first_backend()</code>	Returns a pointer of the backend structure of the first backend.
<code>slapi_get_first_suffix()</code>	Returns the first root suffix of the DIT.
<code>slapi_get_next_backend()</code>	Returns a pointer to the next backend.
<code>slapi_get_next_suffix()</code>	Returns the DN of the next root suffix of the DIT.
<code>slapi_is_root_suffix()</code>	Checks if a suffix is a root suffix of the DIT.
<code>slapi_register_backend_state_change()</code>	Registers for callbacks when a backend changes state.
<code>slapi_unregister_backend_state_change()</code>	Unregisters backend-state-change callbacks.

20.1. SLAPI_BE_ADDSUFFIX()

Description

Adds the specified suffix to the given backend and increments the backend's suffix count.

Syntax


```
#include "slapi-plugin.h"
void slapi_be_addsuffix(Slapi_Backend *be, const Slapi_DN *suffix);
```

Parameters

This function takes the following parameters:

<i>be</i>	Pointer to the structure containing the backend configuration.
<i>suffix</i>	Suffix that needs to be added to the backend.

20.2. SLAPI_BE_DELETE_ONEXIT()

Description

Sets the flag to denote that the backend will be deleted on exiting.

Syntax

```
#include "slapi-plugin.h"
void slapi_be_delete_onexit(Slapi_Backend *be);
```

Parameters

This function takes the following parameter:

<i>be</i>	Pointer to the structure containing the backend configuration.
-----------	--

20.3. SLAPI_BE_EXIST()

Description

Checks if the backend that contains the specified DN exists.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_exist(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

<i>sdn</i>	Pointer to the DN in the backends for which you are looking.
------------	--

Returns

This function returns one of the following values:

- 1 if the backend containing the specified DN exists.
- 0 if the backend does not exist.

See Also

[slapi_be_select\(\)](#)

20.4. SLAPI_BE_FREE()

Description

Frees memory and linked resources from the backend structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_be_free(Slapi_Backend **be);
```

Parameters

This function takes the following parameter:

<i>be</i>	Pointer to the structure containing the backend configuration.
-----------	--

20.5. SLAPI_BE_GET_INSTANCE_INFO()

Description

Gets the instance information of the specified backend.

Syntax

```
#include "slapi-plugin.h"
void * slapi_be_get_instance_info(Slapi_Backend * be);
```

Parameters

This function takes the following parameter:

<i>be</i>	Pointer to the structure containing the backend configuration.
-----------	--

Returns

This function returns an opaque pointer to the instance information.

20.6. SLAPI_BE_GET_NAME()

Description

Returns the name of the specified backend.

Syntax

```
#include "slapi-plugin.h"
char * slapi_be_get_name(Slapi_Backend * be);
```

Parameters

This function takes the following parameter:

<i>be</i>	Pointer to the structure containing the backend configuration.
-----------	--

Returns

This function returns the name associated to the specified backend.

Memory Concerns

You should not free the returned pointer.

20.7. SLAPI_BE_GET_READONLY()**Description**

Indicates if the database associated with the backend is in read-only mode.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_get_readonly(Slapi_Backend *be);
```

Parameters

This function takes the following parameter:

<i>be</i>	Pointer to the structure containing the backend configuration.
-----------	--

Returns

This function returns one of the following values:

- 0 if the database is not in read-only mode.
- 1 if the database is in read-only mode.

20.8. SLAPI_BE_GETENTRYPOINT()**Description**

Sets pointer to a callback function that corresponds to the specified entry point into a given backend.

Syntax

–

```
int slapi_be_getentrypoint(Slapi_Backend *be, int entrypoint,
void **ret_fnptr, Slapi_PBlock *pb);
```

Parameters

This function takes the following parameters:

<i>be</i>	Pointer to the structure containing the backend configuration.
<i>entrypoint</i>	Entry point in the backend.
<i>ret_fnptr</i>	Opaque pointer to store function address.
<i>pb</i>	Pointer to the parameter block.

Returns

This function returns 0 if successful, -1 otherwise.

20.9. SLAPI_BE_GETSUFFIX()

Description

This function returns the **n+1** suffix associated with the specified backend. This function is still present for compatibility purposes with previous versions of the Directory Server Plug-in API. Current versions of Directory Server do not support backends containing multiple suffixes; so, if **n** is not **0**, **NULL** will be returned.

Syntax

```
#include "slapi-plugin.h"
const Slapi_DN *slapi_be_getsuffix(Slapi_Backend *be, int n);
```

Parameters

This function takes the following parameters:

<i>be</i>	Pointer to the structure containing the backend configuration.
<i>n</i>	Index.

Returns

This function returns the DN of the suffix if it exists, or NULL if there is no **n+1** suffix in the backend.

Memory Concerns

You should not free the returned pointer.

20.10. SLAPI_BE_GETTYPE()

Description

Returns the type of the backend.

Syntax

```
#include "slapi-plugin.h"
const char * slapi_be_gettype(Slapi_Backend *be);;
```

Parameters

This function takes the following parameter:

<i>be</i>	Pointer to the structure containing the backend configuration.
-----------	--

Returns

This function returns the type of the backend.

Memory Concerns

You should not free the returned pointer.

20.11. SLAPI_BE_IS_FLAG_SET()

Description

Checks if a flag is set in the backend configuration.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_is_flag_set(Slapi_Backend * be, int flag);
```

Parameters

This function takes the following parameters:

<i>be</i>	Pointer to the structure containing the backend configuration.
<i>flag</i>	Flag to check; for example, SLAPI_BE_FLAG_REMOTE_DATA .

Returns

This function returns one of the following values:

- 0 if a flag is not set in the backend configuration.
- 1 if a flag is set in the backend configuration.

20.12. SLAPI_BE_ISSUFFIX()

Description

This function checks if the specified suffix exactly matches a registered suffix on a specified backend.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_issuffix(const Slapi_Backend *be, const Slapi_DN *suffix );
```

Parameters

This function takes the following parameters:

<i>be</i>	Pointer to the structure containing the backend configuration.
<i>suffix</i>	DN of the suffix for which you are looking.

Returns

This function returns one of the following values:

- 0 if the suffix is not part of the specified backend.
- 1 if the suffix is part of the specified backend.

20.13. SLAPI_BE_LOGCHANGES()

Description

Indicates whether the changes applied to the backend should be logged in the changelog.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_logchanges(Slapi_Backend *be);
```

Parameters

This function takes the following parameter:

<i>be</i>	Pointer to the structure containing the backend configuration.
-----------	--

Returns

This function returns one of the following values:

- 0 if the changes applied to the specific backend should not be logged in the changelog.

- 1 if the changes should be logged in the changelog.

20.14. SLAPI_BE_NEW()

Description

Creates a new backend structure, allocates memory for it, and initializes values for relevant parameters.

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend *slapi_be_new( const char *type, const char *name, int is
private, int logchanges );
```

Parameters

This function takes the following parameters:

<i>type</i>	Database type.
<i>name</i>	Database name.
<i>isprivate</i>	Flag to denote whether the database is private.
<i>logchanges</i>	Flag for indicating whether changes are to be logged.

Returns

This function returns a pointer to the newly-created backend.

20.15. SLAPI_BE_PRIVATE()

Description

Verifies if the backend is private.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_private( Slapi_Backend *be );
```

Parameters

This function takes the following parameter:

<i>be</i>	Pointer to the structure containing the backend configuration.
-----------	--

Returns

This function returns one of the following values:

- 0 if the backend is not hidden from the user.
- 1 if the backend is hidden from the user (for internal use only).

20.16. SLAPI_BE_SELECT()

Description

Finds the backend that should be used to service the entry with the specified DN.

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend *slapi_be_select( const Slapi_DN * sdn );
```

Parameters

This function takes the following parameter:

<i>sdn</i>	Pointer to the DN of which you wish to get the backend.
------------	---

Returns

This function returns one of the following values:

- A pointer to the default backend if no backend with the appropriate suffix is configured.
- A pointer to the backend structure.

Memory Concerns

You should not free the returned pointer.

See Also

[slapi_be_select_by_instance_name\(\)](#)

20.17. SLAPI_BE_SELECT_BY_INSTANCE_NAME()

Description

This function finds the backend that should be used to service the database named as the parameter.

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend *slapi_be_select_by_instance_name( const char *name );
```

Parameters

This function takes the following parameter:

<i>name</i>	Pointer to the name of the backend of which you wish to get the structure.
-------------	--

Returns

This function returns one of the following values:

- NULL if no backend with the appropriate name is configured.
- A pointer to the backend structure.

Memory Concerns

You should not free the returned pointer.

See Also

[slapi_be_select\(\)](#)

20.18. SLAPI_BE_SET_FLAG()

Description

Sets the specified flag in the backend.

Syntax

```
#include "slapi-plugin.h"
void slapi_be_set_flag(Slapi_Backend * be, int flag);
```

Parameters

This function takes the following parameters:

<i>be</i>	Pointer to the structure containing the backend configuration.
<i>flag</i>	Flag (bitmap) to set in the configuration.

20.19. SLAPI_BE_SET_INSTANCE_INFO()

Description

Sets the instance information of the specified backend with given data.

Syntax

```
#include "slapi-plugin.h"
void slapi_be_set_instance_info(Slapi_Backend * be, void * data);
```

Parameters

This function takes the following parameters:

<i>be</i>	Pointer to the structure containing the backend configuration.
<i>data</i>	Data for setting the instance information.

20.20. SLAPI_BE_SET_READONLY()

Description

Sets a flag to denote that the backend is meant to be read-only.

Syntax

```
#include "slapi-plugin.h"
void slapi_be_set_readonly(Slapi_Backend *be, int readonly);
```

Parameters

This function takes the following parameters:

<i>be</i>	Pointer to the structure containing the backend configuration.
<i>readonly</i>	Flag to specify the read-only status.

20.21. SLAPI_BE_SETEXTENTRYPOINT()

Description

Sets the entry point in the backend to the specified function.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_setentrypoint(Slapi_Backend *be, int entrypoint, void
*ret_fnptr, Slapi_PBlock *pb);
```

Parameters

This function takes the following parameters:

<i>be</i>	Pointer to the structure containing the backend configuration.
<i>entrypoint</i>	Entry point in the backend.
<i>ret_fnptr</i>	Opaque pointer to store function address.
<i>pb</i>	Pointer to the parameter block.

Returns

This function returns 0 if successful, -1 otherwise.

20.22. SLAPI_GET_FIRST_BACKEND()

Description

This function returns a pointer to the backend structure of the first backend. If you wish to iterate through all of the backends, use this function in conjunction with [slapi_get_next_backend\(\)](#). For example:

```

Slapi_Backend *be = NULL;
char *cookie = NULL;
be = slapi_get_first_backend (&cookie);
while (be )
{
    ...
    be = slapi_get_next_backend (cookie);
}
slapi_ch_free ((void*)&cookie);

```

Syntax

```

#include "slapi-plugin.h"
Slapi_Backend* slapi_get_first_backend(char **cookie);

```

Parameters

This function takes the following parameter:

<i>cookie</i>	Output parameter containing the index of the returned backend. This is useful for calls to slapi_get_next_backend() . Contains 0 in output if no backend is returned.
---------------	---

Returns

This function returns one of the following values:

- A pointer to the backend structure of the first backend and its index in the **cookie** parameter.
- **NULL** if there is no backend.

Memory Concerns

Free the cookie parameter after the iteration using **slapi_ch_free()**.

See Also

[slapi_get_next_backend\(\)](#)

20.23. SLAPI_GET_FIRST_SUFFIX()

Description

This function returns the first root suffix of the DIT. If you wish to iterate through all of the suffixes, use this function in conjunction with [slapi_get_next_suffix\(\)](#). For example:

```
void *node = NULL;
Slapi_DN * suffix = slapi_get_first_suffix (&node, 1);
while (suffix)
{
    ...
    suffix = slapi_get_next_suffix (&node, 1);
}
```

Syntax

```
#include "slapi-plugin.h"
Slapi_DN * slapi_get_first_suffix(void ** node, int show_private);
```

Parameters

This function takes the following parameter:

<i>node</i>	Contains the returned value, which is the DN of the first root suffix of the DIT.
<i>show_private</i>	0 checks only for non-private suffixes. 1 checks for both private and non-private suffixes.

Returns

This function returns the DN of the first root suffix.

Memory Concerns

You should not free the returned pointer.

See Also

[slapi_get_next_suffix\(\)](#)

20.24. SLAPI_GET_NEXT_BACKEND()

Description

This function returns a pointer to the next backend. If you wish to iterate through all of the backends, use this function in conjunction with [slapi_get_first_backend\(\)](#). For example:

```
Slapi_Backend *be = NULL;
char *cookie = NULL;
be = slapi_get_first_backend (&cookie);
while (be )
{
    ...
    be = slapi_get_next_backend (cookie);
}
slapi_ch_free ((void*)&cookie);
```

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend* slapi_get_next_backend(char *cookie);
```

Parameters

This function takes the following parameter:

<i>cookie</i>	Upon input, contains the index from which the search for the next backend is done. Upon output, contains the index of the returned backend.
---------------	---

Returns

This function returns one of the following values:

- A pointer to the next backend, if it exists, and updates the **cookie** parameter.
- **NULL**, and **cookie** is not changed.

Memory Concerns

Free the cookie parameter after the iteration using **slapi_ch_free()**.

See Also

- [slapi_get_first_backend\(\)](#)
- [slapi_ch_free\(\)](#)

20.25. SLAPI_GET_NEXT_SUFFIX()

Description

This function returns the DN of the next root suffix of the DIT. If you wish to iterate through all of the suffixes, use this function in conjunction with [slapi_get_first_suffix\(\)](#). For example:

```
void *node = NULL;
Slapi_DN * suffix = slapi_get_first_suffix (&node, 1);
while (suffix)
{
    ...
    suffix = slapi_get_next_suffix (&node, 1);
}
```

Syntax

```
#include "slapi-plugin.h"
Slapi_DN * slapi_get_next_suffix(void ** node, int show_private);
```

Parameters

This function takes the following parameter:

<i>show_private</i>	0 checks only for non-private suffixes. 1 checks for both private and non-private suffixes.
<i>node</i>	Contains the returned value, which is the DN of the next root suffix of the DIT.

Returns

This function returns one of the following values:

- The DN of the next root suffix of the DIT.
- NULL if there are more suffixes to parse.

Memory Concerns

You should not free the returned pointer.

See Also

[slapi_get_first_suffix\(\)](#)

20.26. SLAPI_IS_ROOT_SUFFIX()

Description

Checks if a suffix is a root suffix of the DIT.

Syntax

```
#include "slapi-plugin.h"
int slapi_is_root_suffix(Slapi_DN * dn);
```

Parameters

This function takes the following parameter:

<i>dn</i>	DN to check.
-----------	--------------

Returns

This function returns one of the following values:

- 0 if the DN is not a root suffix.
- 1 if the DN is a root suffix.

20.27. SLAPI_REGISTER_BACKEND_STATE_CHANGE()

Description

This function enables a plug-in to register for callback when the state of a backend changes. The function will come handy when developing custom plug-ins.

For example, if your plug-in stores any kind of state, such as a configuration cache, it will

become invalidated or incomplete whenever the state of a backend changes. Because the plug-in wouldn't be aware of these state changes, it would require restarting the server whenever a backend state changed.

By registering for callback whenever the backend changes its state, your plug-in can keep track of these changes and retain its functionality. You can use [slapi_unregister_backend_state_change\(\)](#) to unregister the callback.

Syntax

```
#include "slapi-plugin.h"
void slapi_register_backend_state_change(void * handle,
slapi_backend_state_change_fnptr funct);
```

Parameters

This function takes the following parameter:

handle	Pointer or reference to the address of the specified function.
--------	--

See Also

[slapi_unregister_backend_state_change\(\)](#)

20.28. SLAPI_UNREGISTER_BACKEND_STATE_CHANGE()

Description

This function enables a plug-in to unregister backend-state-change callback. Use this function to unregister the callback, which is registered using [slapi_register_backend_state_change\(\)](#).

Syntax

```
#include "slapi-plugin.h"
int slapi_unregister_backend_state_change(void * handle);
```

Parameters

This function takes the following parameter:

<i>handle</i>	Pointer or reference to the address of the specified function.
---------------	--

Returns

This function returns one of the following values:

- 0 if the specified callback was found and unregistered successfully.
- 1 if the specified callback wasn't unregistered successfully; for example, if it were not found.

See Also

[slapi_register_backend_state_change\(\)](#)

CHAPTER 21. FUNCTIONS FOR DEALING WITH CONTROLS

This chapter contains reference information on routines for dealing with controls.

Table 21.1. Routines for Dealing with Controls

Function	Description
slapi_add_control_ext()	Adds the specified control to the end of an array of controls.
slapi_add_controls()	Appends all of an array of controls to an existing array of controls.
slapi_build_control()	Creates an LDAPControl structure based on a BerElement, an OID, and a criticality flag.
slapi_build_control_from_berval()	Creates an LDAPControl structure based on a struct berval, an OID, and a criticality flag.
slapi_control_present()	Determines whether the specified object identification (OID) identifies a control that is present in a list of controls.
slapi_dup_control()	Makes an allocated copy of an LDAPControl.
slapi_get_supported_controls_copy()	Retrieves an allocated array of object identifiers (OIDs) representing the controls supported by the Directory Server.
slapi_register_supported_control()	Registers the specified control with the server. This function associates the control with an object identification (OID).

21.1. SLAPI_ADD_CONTROL_EXT()

This function specifies an LDAPControl, and then appends the control to the end of a specified array or creates a new array. For example:

```
slapi_add_control_ext(&ctrls, newctrl, 1);
```

An existing array is grown using [slapi_ch_realloc\(\)](#). Otherwise, a new array is created using [slapi_ch_malloc\(\)](#).

Syntax

```
#include "slapi-plugin.h"
void slapi_add_control_ext(LDAPControl ***ctrlsp, LDAPControl *newctrl,
int copy)
```

Parameters

This function takes the following parameters:

<i>ctrlsp</i>	Pointer that will receive the specified LDAPControl. If <i>ctrls</i> is NULL , then the array is created using slapi_ch_malloc() .
<i>newctrl</i>	Pointer to the specified LDAPControl.
<i>copy</i>	Sets whether the given control is copied. If the value is true (0), then the control in <i>newctrl</i> is copied. If the value is false (1), then the control in <i>newctrl</i> is used and owned by the array and must be freed using ldap_controls_free .

Returns

This function returns **LDAP_SUCCESS** (LDAP result code) if successful.

See Also

- [slapi_ch_realloc\(\)](#)
- [slapi_ch_malloc\(\)](#)
- `ldap_controls_free`

21.2. SLAPI_ADD_CONTROLS()

This function specifies an array of LDAPControls, and then either appends the array to the end of a specified array or uses it to create a new array.

An existing array is grown using [slapi_ch_realloc\(\)](#). Otherwise, a new array is created using [slapi_ch_malloc\(\)](#).

Syntax

```
#include "slapi-plugin.h"
void slapi_add_controls(LDAPControl ***ctrlsp, LDAPControl **newctrls, int
copy)
```

Parameters

This function takes the following parameters:

<i>ctrlsp</i>	Pointer that will receive the specified LDAPControl array. If <i>ctrls</i> is NULL , then a new array is created using slapi_ch_malloc() .
<i>newctrasl</i>	The specified LDAPControl array.

<i>copy</i>	Sets whether the given controls are copied. If the value is true (0), then all of the controls in <i>newctrls</i> are copied. If the value is false (1), then all of the controls in <i>newctrls</i> are used and owned by the array and must be freed using ldap_controls_free .
-------------	--

Returns

This function returns **LDAP_SUCCESS** (LDAP result code) if successful.

See Also

- [slapi_ch_realloc\(\)](#)
- [slapi_ch_malloc\(\)](#)
- [ldap_controls_free](#)

21.3. SLAPI_BUILD_CONTROL()

This function creates an LDAPControl structure based on a BerElement, an OID, and a criticality flag. The LDAPControl that is created can be used in LDAP client requests or internal operations.

Syntax

```
#include "slapi-plugin.h"
int slapi_build_control( char *oid, BerElement *ber, char iscritical,
LDAPControl **ctrlp );
```

Parameters

This function takes the following parameters:

<i>oid</i>	The OID (object identifier) for the control that is to be created.
<i>ber</i>	A BerElement that contains the control value. Pass NULL if the control has no value.
<i>iscritical</i>	The criticality flag. If non-zero, the control will be marked as critical. If 0 , it will not be marked as critical.
<i>ctrlp</i>	Pointer that will receive the allocated LDAPControl structure.

Returns

This function returns **LDAP_SUCCESS** (LDAP result code) if successful.

Memory Concerns

The contents of the *ber* parameter are consumed by this function. Because of this, the caller should not free the *BerElement* once a successful call has been made to **slapi_build_control()**.

The *LDAPControl* pointer that is returned in *ctrlp* should be freed by calling **ldap_control_free()**, which is an LDAP API function; see the *Mozilla LDAP SDK for C Programmer's Guide*.

See Also

- [slapi_build_control_from_berval\(\)](#)
- [ldap_control_free\(\)](#)

21.4. SLAPI_BUILD_CONTROL_FROM_BERVAL()

Description

This function creates an *LDAPControl* structure based on a struct *berval*, an *OID*, and a criticality flag. The *LDAPControl* that is created can be used in LDAP client requests or internal operations.

Syntax

```
#include "slapi-plugin.h"
int slapi_build_control_from_berval( char *oid, struct berval *bvp, char
iscritical, LDAPControl **ctrlp );
```

Parameters

This function takes the following parameters:

<i>oid</i>	The <i>OID</i> (object identifier) for the control that is to be created.
<i>bvp</i>	A struct <i>berval</i> that contains the control value. Pass NULL if the control has no value.
<i>iscritical</i>	The criticality flag. If non-zero, the control will be marked as critical. If 0 , it will not be marked as critical.
<i>ctrlp</i>	Pointer that will receive the allocated <i>LDAPControl</i> structure.

Returns

This function returns **LDAP_SUCCESS** (LDAP result code) if successful.

Memory Concerns

The contents of the *bvp* parameter are consumed by this function. Because of this, the caller should not free the *bvp->bv_val* pointer once a successful call to this function has been made.

The LDAPControl pointer that is returned in *ctrlp* should be freed by calling `ldap_control_free()`, which is an LDAP API function; see the *Mozilla LDAP SDK for C Programmer's Guide*.

See Also

- [slapi_build_control\(\)](#)
- [slapi_dup_control\(\)](#)
- `ldap_control_free()`

21.5. SLAPI_CONTROL_PRESENT()

Description

Determines whether the specified object identification (OID) identifies a control that is present in a list of controls.

Syntax

```
#include "slapi-plugin.h"
int slapi_control_present( LDAPControl **controls, char *oid, struct
berval **val, int *iscritical );
```

Parameters

This function takes the following parameters:

<i>controls</i>	List of controls that you want to check.
<i>oid</i>	OID of the control that you want to find.
<i>val</i>	If the control is present in the list of controls, specifies the pointer to the berval structure containing the value of the control.
<i>iscritical</i>	<p>If the control is present in the list of controls, specifies whether the control is critical to the operation of the server:</p> <ul style="list-style-type: none"> • 0 means that the control is not critical to the operation. • 1 means that the control is critical to the operation.

Returns

This function returns one of the following values:

- **1** if the specified control is present in the list of controls.
- **0** if the control is not present in the list of controls.

Memory Concerns

The *val* output parameter is set to point into the controls array. A copy of the control value is not made.

See Also

- [slapi_get_supported_controls_copy\(\)](#)
- [slapi_register_supported_control\(\)](#)

21.6. SLAPI_DUP_CONTROL()

Description

Makes an allocated copy of an LDAPControl. This function duplicates the contents of an LDAPControl structure. All fields within the LDAPControl are copied to a new, allocated structure, and a pointer to the new structure is returned.

Syntax

```
#include "slapi-plugin.h"
LDAPControl * slapi_dup_control( LDAPControl *ctrl )
```

Parameters

This function takes the following parameter:

<i>ctrl</i>	Pointer to an LDAPControl structure whose contents are to be duplicated.
-------------	--

Returns

This function returns one of the following values:

- A pointer to an allocated LDAPControl structure if successful.
- **NULL** if an error occurs.

Memory Concerns

The structure that is returned should be freed by calling **ldap_control_free()**, which is an LDAP API function; see the *Mozilla LDAP SDK for C Programmer's Guide*.

See Also

ldap_control_free()

21.7. SLAPI_GET_SUPPORTED_CONTROLS_COPY()

Description

This function replaces the deprecated **slapi_get_supported_controls()** function from previous releases, as it was not multi thread safe. It retrieves an allocated array of object identifiers (OIDs) representing the controls supported by the Directory Server. You can register new controls by calling [slapi_register_supported_control\(\)](#).

When you call `slapi_register_supported_control()` to register a control, you specify the OID of the control and the IDs of the operations that support the control. The server records this information in two arrays; an array of control OIDs and an array of operations that support the control. You can get copies of these arrays by calling `slapi_get_supported_controls_copy()`.

For each OID returned in the **ctrloidsp** array, the corresponding array element (with the same index) in the **ctrlovsp** array identifies the operations that support the control. For a list of the possible IDs for the operations, refer to `slapi_register_supported_control()`.

Syntax

```
#include "slapi-plugin.h"
int slapi_get_supported_controls_copy( char ***ctrloidsp, unsigned long
**ctrlovsp );
```

Parameters

This function takes the following parameters:

<i>ctrloidsp</i>	Pointer to a character array that will receive the set of supported control OIDs. Pass NULL for this parameter if you do not wish to receive the OIDs.
<i>ctrlovsp</i>	Pointer to an unsigned long array that will receive the supported operation values for each control in the ctrloidsp array. Pass NULL for this parameter if you do not wish to receive the supported operation values.

Returns

This function returns one of the following values:

- 0 if successful.
- A non-zero value if an error occurs.

Memory Concerns

The returned **ctrloidsp** array should be freed by calling `slapi_ch_array_free()`. The returned **ctrlovsp** array should be freed by calling `slapi_ch_free()`.

See Also

- [slapi_register_supported_control\(\)](#)
- [slapi_ch_array_free\(\)](#)

21.8. SLAPI_REGISTER_SUPPORTED_CONTROL()

Description

Registers the specified control with the server. This function associates the control with an object identification (OID). When the server receives a request that specifies this OID, the

server makes use of this information to determine if the control is supported by the server or its plug-ins.

Syntax

```
#include "slapi-plugin.h"
void slapi_register_supported_control( char *controloid, unsigned long
controlops );
```

Parameters

This function takes the following parameters:

<i>controloid</i>	OID of the control you want to register.
<i>controlops</i>	Operation to which the control is applicable.
<i>SLAPI_OPERATION_BIND</i>	The specified control applies to the LDAP bind operation.
<i>SLAPI_OPERATION_UNBIND</i>	The specified control applies to the LDAP unbind operation.
<i>SLAPI_OPERATION_SEARCH</i>	The specified control applies to the LDAP search operation.
<i>SLAPI_OPERATION_MODIFY</i>	The specified control applies to the LDAP modify operation.
<i>SLAPI_OPERATION_ADD</i>	The specified control applies to the LDAP add operation.
<i>SLAPI_OPERATION_DELETE</i>	The specified control applies to the LDAP delete operation.
<i>SLAPI_OPERATION_MODDN</i>	The specified control applies to the LDAP modify DN operation.
<i>SLAPI_OPERATION_MODRDN</i>	The specified control applies to the LDAPv3 modify RDN operation.
<i>SLAPI_OPERATION_COMPARE</i>	The specified control applies to the LDAP compare operation.
<i>SLAPI_OPERATION_ABANDON</i>	The specified control applies to the LDAP abandon operation.
<i>SLAPI_OPERATION_EXTENDED</i>	The specified control applies to the LDAPv3 extended operation.

<i>SLAPI_OPERATION_ANY</i>	The specified control applies to any LDAP operation.
<i>SLAPI_OPERATION_NONE</i>	The specified control applies to none of the LDAP operations.

You can specify a combination of values by bitwise OR-ing the values together. For example: **SLAPI_OPERATION_ADD | SLAPI_OPERATION_DELETE**

See Also

[slapi_get_supported_controls_copy\(\)](#) [slapi_control_present\(\)](#)

CHAPTER 22. FUNCTIONS FOR SYNTAX PLUG-INS

This chapter contains reference information on routines for syntax plug-ins.

Table 22.1. Syntax Plug-in Routines

Function	Description
<code>slapi_call_syntax_assertion2keys_ava_sv()</code>	Calls a function, specified in the syntax plug-in, to compare against directory entries.
<code>slapi_call_syntax_assertion2keys_sub_sv()</code>	Calls a function, specified in the syntax plug-in, to compare against directory entries.
<code>slapi_call_syntax_values2keys_sv()</code>	Manages index values when adding or removing values from an index.

22.1. SLAPI_CALL_SYNTAX_ASSERTION2KEYS_AVA_SV()

Description

When processing a search, calls the function (defined in the specified syntax plug-in) responsible for returning an array of values (specified by the search filter) to compare against the entries in the directory. This function applies to searches that use the filter types **LDAP_FILTER_EQUALITY** and **LDAP_FILTER_APPROX**.

When processing a search that uses an attribute-value assertion (AVA) filter (for example, **ou=Accounting** or **ou=~Accounting**), the backend needs to compare the value specified in the search filter against the value of the specified attribute in each entry.

The function invokes the syntax plug-in specified by the **vpi** argument. This is the plug-in associated with the type of attribute used in the search. You can get this handle by calling the `slapi_attr_type2plugin()` function.

The syntax plug-in function invoked by this function is responsible for comparing the value specified by *val* against the actual values of the attributes in the directory entries. The syntax plug-in function returns a list of matching entry keys (the *ivals* argument) to the backend.

Syntax

```
#include "slapi-plugin.h"
int slapi_call_syntax_assertion2keys_ava_sv( void *vpi, Slapi_Value *val,
Slapi_Value ***ivals, int ftype );
```

Parameters

This function takes the following parameters:

<i>vpi</i>	Handle to plug-in for this attribute type
------------	---

<i>val</i>	Pointer to the Slapi_Value arrays containing the value from the search filter; for example, if the filter is ou=Accounting , the argument <i>val</i> is Accounting .
<i>ivals</i>	Pointer to the Slapi_Value arrays containing the values returned by the plug-in function; these values can now be compared against entries in the directory.
<i>ftype</i>	Type of filter; for example, LDAP_FILTER_EQUALITY .

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs; for example, if the corresponding function for the specified plug-in is not found.

22.2. SLAPI_CALL_SYNTAX_ASSERTION2KEYS_SUB_SV()

Description

When processing a search, calls the function (defined in the specified syntax plug-in) responsible for returning an array of values (specified by the search filter) to compare against the entries in the directory. This function applies to searches that use the filter type **LDAP_FILTER_SUBSTRINGS**.

The **ldbm** backend (the default backend database) calls this function when processing searches in which the filter type is **LDAP_FILTER_SUBSTRINGS**.

The function invokes the syntax plug-in specified by the **vpi** argument. This is the plug-in associated with the type of attribute used in the search. You can get this handle by calling the [slapi_attr_type2plugin\(\)](#) function.

The syntax plug-in function invoked by this function is responsible for comparing the values (specified by *initial*, *any*, and *final*) against the actual values of the attributes in the directory entries. The syntax plug-in function returns a list of matching entry keys (the *ivals* argument) to the backend.

Syntax

```
#include "slapi-plugin.h"
int slapi_call_syntax_assertion2keys_sub_sv( void *vpi, char *initial,
char **any, char *final, Slapi_Value ***ivals );
```

Parameters

This function takes the following parameters:

<i>vpi</i>	Handle to plug-in for this attribute type
<i>initial</i>	"Starts with" value from the search filter; for example, if the filter is ou=Sales* , the argument initial is Sales .
<i>any</i>	Array of "contains" values from the search filter; for example, if the filter is ou=*Corporate*Sales* , the argument <i>any</i> is an array containing Corporate and Sales .
<i>final</i>	"Ends with" value from the search filter; for example, if the filter is ou=*Sales , the argument <i>final</i> is Sales .
<i>ivals</i>	Pointer to an array of Slapi_Value structures containing the values returned by the plug-in function; these values can now be compared against entries in the directory.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs; for example, if the corresponding function for the specified plug-in is not found.

22.3. SLAPI_CALL_SYNTAX_VALUES2KEYS_SV()

Description

When adding or removing values from an index, the Directory Server calls the function (defined in the specified syntax plug-in) responsible for returning an array of keys matching the specified values.

Syntax

```
#include "slapi-plugin.h"
int slapi_call_syntax_values2keys_sv( void *vpi, Slapi_Value **vals,
Slapi_Value ***ivals, int ftype );
```

Parameters

This function takes the following parameters:

<i>vpi</i>	Handle to plug-in for this attribute type.
<i>vals</i>	Pointer to the Slapi_Value structure containing the value to add or delete.

<i>ivals</i>	Pointer to an array of Slapi_Value structures containing the values returned by the plug-in function; these values can be compared against entries in the directory.
<i>ftype</i>	Type of filter; for example, LDAP_FILTER_EQUALITY .

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs; for example, if the corresponding function for the specified plug-in is not found.

CHAPTER 23. FUNCTIONS FOR MANAGING MEMORY

This chapter contains reference information on routines for managing memory, such as allocating and freeing memory.

Table 23.1. Memory Management Routines

Function	Description
slapi_ch_array_add()	Adds a new array.
slapi_ch_array_free()	Frees an existing array.
slapi_ch_bvdup()	Makes a copy of an existing berval structure.
slapi_ch_bvecdup()	Makes a copy of an array of existing berval structures.
slapi_ch_calloc()	Allocates space for an array of a number of elements of a specified size.
slapi_ch_free()	Frees space allocated by the slapi_ch_malloc() , slapi_ch_realloc() , and slapi_ch_calloc() functions.
slapi_ch_free_string()	Frees space previously allocated to a string.
slapi_ch_malloc()	Allocates space in memory.
slapi_ch_realloc()	Changes the size of a block of allocated memory.
slapi_ch_smprintf()	Creates, formats, and returns a given string.
slapi_ch_strdup()	Makes a copy of an existing string.

23.1. SLAPI_CH_ARRAY_ADD()

Adds the given string to the given null-terminated array.

If the *array* is **NULL**, then the function creates a new array.

string is not copied, so if you want to add a copy of *string* to the array, use [slapi_ch_strdup\(\)](#).

Then use [slapi_ch_array_free\(\)](#) to free the array.

For example:

```
char **array = NULL;
```

```

...
slapi_ch_array_add(&array, slapi_ch_strdup("some string"));
...
slapi_ch_array_free(array);

```

Syntax

```

#include "slapi-plugin.h"
void slapi_ch_array_add(char ***array, char *string);

```

Parameters

This function takes the following parameters:

<i>array</i>	Pointer to the array to be freed. If this parameter is NULL , then the function creates a new array.
<i>string</i>	The string to add to the array.

23.2. SLAPI_CH_ARRAY_FREE()

This function frees the **char **** pointed to by *arrayp*.

Syntax

```

#include "slapi-plugin.h"
void slapi_ch_array_free( char **arrayp );

```

Parameters

This function takes the following parameter:

<i>arrayp</i>	Pointer to the array to be freed. This parameter can be NULL .
---------------	---

See Also

- [slapi_ch_strdup\(\)](#)
- [slapi_ch_array_free\(\)](#)

23.3. SLAPI_CH_BVDUP()

Description

Makes a copy of an existing berval structure.

Syntax

```

#include "slapi-plugin.h"
extern struct berval* slapi_ch_bvdup( const struct berval *v );

```

Parameters

This function takes the following parameter:

<code>v</code>	Pointer to the berval structure that you want to copy.
----------------	--

Returns

This function returns a pointer to the new copy of the berval structure. If the structure cannot be duplicated, e.g., because of insufficient virtual memory, the **slapd** program terminates.

Memory Concerns

The contents of the `v` parameter are not altered by this function. The returned berval structure should be freed by calling **ber_bvfree()**, which is an LDAP API function; see the *Mozilla LDAP SDK for C Programmer's Guide*.

See Also

- [slapi_ch_bvdup\(\)](#)
- [ber_bvfree\(\)](#)

23.4. SLAPI_CH_BVECDUP()

Description

Makes a copy of an array of existing berval structures.

Syntax

```
#include "slapi-plugin.h"
extern struct berval** slapi_ch_bvecdup (const struct berval **v);
```

Parameters

This function takes the following parameter:

<code>v</code>	Pointer to the array of berval structures that you want to copy.
----------------	--

Returns

This function returns a pointer to an array of the new copy of the berval structures. If the structures cannot be duplicated, e.g., because of insufficient virtual memory, the **slapd** program terminates.

Memory Concerns

The contents of the `v` parameter are not altered by this function. The returned berval structure should be freed by calling **ber_bvfree()**, which is an LDAP API function; see the *Mozilla LDAP SDK for C Programmer's Guide*.

See Also

- [slapi_ch_bvdup\(\)](#)
- [ber_bvfree\(\)](#)

23.5. SLAPI_CH_CALLOC()

Description

Allocates space for an array of a number of elements of a specified size.

Syntax

```
#include "slapi-plugin.h"
char * slapi_ch_calloc( unsigned long nelem, unsigned long size );
```

Parameters

This function takes the following parameters:

<i>nelem</i>	Number of elements for which you want to allocate memory.
<i>size</i>	Size, in bytes, of the element for which you want to allocate memory.

Returns

This function returns a pointer to the newly allocated space of memory. If space cannot be allocated, e.g., no more virtual memory exists, the **slapd** program terminates.

Memory Concerns

This function terminates the **slapd** server with an "out of memory" error message if memory cannot be allocated. You should free the returned pointer by calling [slapi_build_control\(\)](#).

See Also

- [slapi_ch_free\(\)](#)
- [slapi_ch_malloc\(\)](#)
- [slapi_ch_realloc\(\)](#)
- [slapi_ch_strdup\(\)](#)

23.6. SLAPI_CH_FREE()

Description

Frees space allocated by the [slapi_ch_malloc\(\)](#), [slapi_ch_realloc\(\)](#), and [slapi_ch_calloc\(\)](#) functions and sets the pointer to **NULL**. Call this function instead of the standard **free()** C function.

Syntax

—

```
#include "slapi-plugin.h"
void slapi_ch_free( void **ptr );
```

Parameters

This function takes the following parameter:

<i>ptr</i>	Address of the pointer to the block of memory that you want to free. If NULL , no action occurs.
------------	---

Memory Concerns

The *ptr* passed to **slapi_ch_free()** should be the address of a pointer that was allocated using a **slapi** call such as [slapi_ch_malloc\(\)](#), [slapi_ch_calloc\(\)](#), [slapi_ch_realloc\(\)](#), or [slapi_ch_strdup\(\)](#).

See Also

- [slapi_ch_malloc\(\)](#)
- [slapi_ch_calloc\(\)](#)
- [slapi_ch_realloc\(\)](#)
- [slapi_ch_strdup\(\)](#)

23.7. SLAPI_CH_FREE_STRING()

Description

This function frees space previously allocated to a string. This function is similar to [slapi_ch_free\(\)](#), but the argument is the address of a string. This helps with compile time error checking.

Syntax

```
#include "slapi-plugin.h"
void slapi_ch_free_string( char **s );
```

Parameters

This function takes the following parameter:

<i>s</i>	Address of the string that you want to free. If NULL , no action occurs.
----------	---

See Also

[slapi_ch_free\(\)](#)

23.8. SLAPI_CH_MALLOC()

Allocates space in memory.

Syntax

```
#include "slapi-plugin.h"
char * slapi_ch_malloc( unsigned long size );
```

Parameters

This function takes the following parameter:

<i>size</i>	Size of the space for which you want to allocate memory.
-------------	--

Returns

This function returns a pointer to the newly allocated space of memory. If space cannot be allocated, e.g., no more virtual memory exists, the **slapd** program terminates.

Memory Concerns

This function terminates the **slapd** server with an "out of memory" error message if memory cannot be allocated.

The returned pointer should be freed by calling [slapi_ch_free\(\)](#).

See Also

- [slapi_ch_free\(\)](#)
- [slapi_ch_calloc\(\)](#)
- [slapi_ch_realloc\(\)](#)
- [slapi_ch_strdup\(\)](#)

23.9. SLAPI_CH_REALLOC()

Description

Changes the size of a block of allocated memory.

Syntax

```
#include "slapi-plugin.h"
char * slapi_ch_realloc( char *block, unsigned long size );
```

Parameters

This function takes the following parameters:

<i>block</i>	Pointer to an existing block of allocated memory.
<i>size</i>	New size of the block of memory you want allocated.

Returns

This function returns a pointer to the reallocated space of memory. If space cannot be allocated, e.g., no more virtual memory exists, the **slapd** program terminates.

Memory Concerns

This function terminates the **slapd** server with an "out of memory" error message if memory cannot be allocated.

The block parameter passed to **slapi_ch_realloc()** should be the address of a pointer that was allocated using a **slapi** call such as [slapi_ch_malloc\(\)](#), [slapi_ch_calloc\(\)](#), or [slapi_ch_strdup\(\)](#). The returned pointer should be freed by calling [slapi_ch_free\(\)](#).

See Also

- [slapi_ch_free\(\)](#)
- [slapi_ch_calloc\(\)](#)
- [slapi_ch_strdup\(\)](#)

23.10. SLAPI_CH_SMPRINTF()

Creates, writes to, and returns a given string.

Syntax

```
#include "slapi-plugin.h"
char *string = slapi_ch_sprintf (format, *arg, ...);
```

Parameters

This function takes the following parameter:

<i>string</i>	String that is printed.
<i>format</i>	A printf -style format string.
<i>arg</i>	Arguments to pass for the string.

Returns

- 0, if the string is successfully printed and returned.
- 1, if an error occurs.

This function the specified string, write to it using the specified formats and arguments, and returns the string. If space cannot be allocated, e.g., no more virtual memory exists, the **slapd** program terminates.

Memory Concerns

This function terminates the **slapd** server with an "out of memory" error message if memory cannot be allocated. The returned string should be freed by calling [slapi_ch_free\(\)](#) to avoid memory leaks.

See Also

- [slapi_ch_free\(\)](#)
- [slapi_ch_calloc\(\)](#)
- [slapi_ch_malloc\(\)](#)
- [slapi_ch_realloc\(\)](#)

23.11. SLAPI_CH_STRDUP()

Makes a copy of an existing string.

Syntax

```
#include "slapi-plugin.h"
char * slapi_ch_strdup( const char *s );
```

Parameters

This function takes the following parameter:

<i>s</i>	Pointer to the string you want to copy.
----------	---

Returns

This function returns a pointer to a copy of the string. If space cannot be allocated, e.g., no more virtual memory exists, the **slapd** program terminates.

Memory Concerns

This function terminates the **slapd** server with an "out of memory" error message if memory cannot be allocated.

The returned pointer should be freed by calling [slapi_ch_free\(\)](#).

See Also

- [slapi_ch_free\(\)](#)
- [slapi_ch_calloc\(\)](#)
- [slapi_ch_malloc\(\)](#)
- [slapi_ch_realloc\(\)](#)

CHAPTER 24. FUNCTIONS FOR MANAGING ENTRIES

This chapter contains reference information on routines for managing entries.

Table 24.1. Entry Routines

Function	Description
slapi_entry2str()	Generates an LDIF string description.
slapi_entry2str_with_options()	Generates an LDIF string descriptions with options.
slapi_entry_add_rdn_values()	Add components in an entry's RDN.
slapi_entry_add_string()	Adds a string value to an attribute in an entry.
slapi_entry_add_value()	Adds a data value to an attribute in an entry.
slapi_entry_add_values_sv()	Adds an array of data values to an attribute in an entry.
slapi_entry_add_valueset()	Adds a data value to an attribute in an entry.
slapi_entry_alloc()	Allocates memory for a new entry.
Section 24.9, “ slapi_entry_apply_mods() ”	Applies an array of LDAPMod to a Slapi_Entry .
slapi_entry_attr_delete()	Deletes an attribute from an entry.
slapi_entry_attr_find()	Checks if an entry contains a specific attribute.
slapi_entry_attr_get_bool()	Gets a given attribute value as a boolean.
slapi_entry_attr_get_charptr()	Gets the first value as a string.
slapi_entry_attr_get_charray()	Gets the values of a multi-valued attribute of an entry.
slapi_entry_attr_get_int()	Gets the first value as an integer.
slapi_entry_attr_get_long()	Gets the first value as a long.
slapi_entry_attr_get_uint()	Gets the first value as an unsigned integer.
slapi_entry_attr_get_ulong()	Gets the first value as an unsigned long.

Function	Description
<code>slapi_entry_attr_has_syntax_value()</code>	Checks if an attribute in an entry contains a value.
<code>slapi_entry_attr_merge_sv()</code>	Adds an array to the attribute values in an entry.
<code>slapi_entry_attr_replace_sv()</code>	Replaces the values of an attribute.
<code>slapi_entry_attr_set_charptr()</code>	Replaces the values of an attribute with a string.
<code>slapi_entry_attr_set_int()</code>	Replaces the value of an attribute with an integer.
<code>slapi_entry_attr_set_long()</code>	Replaces the value of an attribute with a long.
<code>slapi_entry_attr_set_uint()</code>	Replaces the value of an attribute with an unsigned integer.
<code>slapi_entry_attr_set_ulong()</code>	Replaces the value of an attribute with an unsigned long.
<code>slapi_entry_delete_string()</code>	Deletes a string from an attribute.
<code>slapi_entry_delete_values_sv()</code>	Removes a <code>Slapi_Value</code> array from an attribute.
<code>slapi_entry_dup()</code>	Copies an entry, its DN, and its attributes.
<code>slapi_entry_first_attr()</code>	Finds the first attribute in an entry.
<code>slapi_entry_free()</code>	Frees an entry from memory.
<code>slapi_entry_get_dn()</code>	Gets the DN from an entry.
<code>slapi_entry_get_dn_const()</code>	Returns the DN of an entry as a constant.
<code>slapi_entry_get_ndn()</code>	Returns the NDN of an entry.
<code>slapi_entry_get_sdn()</code>	Returns the <code>Slapi_DN</code> from an entry.
<code>slapi_entry_get_sdn_const()</code>	Returns a <code>Slapi_DN</code> from an entry as a constant.
<code>slapi_entry_get_uniqueid()</code>	Gets the unique ID from an entry.

Function	Description
<code>slapi_entry_has_children()</code>	Determines if the specified entry has child entries.
<code>slapi_entry_init()</code>	Initializes the values of an entry.
<code>slapi_entry_merge_values_sv()</code>	Adds an array of data values to an attribute in an entry.
<code>slapi_entry_next_attr()</code>	Finds the next attribute in an entry.
<code>slapi_entry_rdn_values_present()</code>	Checks if values present in an entry's RDN are also present as attribute values.
<code>slapi_entry_schema_check()</code>	Determines if an entry complies with the schema for its object class.
<code>slapi_entry_set_dn()</code>	Sets the DN of an entry.
<code>slapi_entry_set_sdn()</code>	Sets the Slapi_DN value in an entry.
<code>slapi_entry_set_uniqueid()</code>	Sets the unique ID in an entry.
<code>slapi_entry_size()</code>	Returns the size of an entry.
<code>slapi_is_rootdse()</code>	Determines if an entry is the root DSE.
<code>slapi_str2entry()</code>	Converts an LDIF description into an entry.

24.1. SLAPI_ENTRY2STR()

Generates an LDIF string description of an LDAP entry.

Description

This function generates an LDIF string value conforming to the following format:

```
dn: dn\n
[attr: value\n]*
```

For example:

```
dn: uid=jdoe, ou=People, dc=example,dc=com
cn: Jane Doe
sn: Doe
...
```


To convert a string description in LDIF format to an entry of the [Section 14.22, “Slapi_Entry”](#) data type, call the [slapi_str2entry\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry2str( Slapi_Entry *e, int *len );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry that you want to convert into an LDIF string.
<i>len</i>	Length of the returned LDIF string.

Returns

This function returns one of the following values:

- The LDIF string representation of the entry you specify.
- **NULL** if an error occurs.

Memory Concerns

When you no longer need to use the string, you should free it from memory by calling the [slapi_ch_free\(\)](#) function.

See Also

- [slapi_entry2str_with_options\(\)](#)
- [slapi_str2entry\(\)](#)

24.2. SLAPI_ENTRY2STR_WITH_OPTIONS()

Generates a description of an entry as an LDIF string. This function behaves much like [slapi_entry2str\(\)](#); however, you can specify output options with this function.

This function generates an LDIF string value conforming to the following syntax:

```
dn: dn\n
[attr: value\n]*
```

For example:

```
dn: uid=jdoe, ou=People, dc=example,dc=com
cn: Jane Doe
sn: Doe
...
```

To convert an entry described in LDIF string format to an LDAP entry using the [Section 14.22, “Slapi_Entry”](#) data type, call the `slapi_str2entry()` function.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry2str_with_options( Slapi_Entry *e, int *len, int options
);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry that you want to convert into an LDIF string.
<i>len</i>	Length of the LDIF string returned by this function.
<i>options</i>	An option set that specifies how you want the string converted.

The Options Parameter

You can **OR** together any of the following options when you call this function:

Flag Value	Description
<i>SLAPI_DUMP_STATEINFO</i>	This is only used internally by replication. This allows access to the internal data used by multi-master replication.
<i>SLAPI_DUMP_UNIQUEID</i>	This option is used when creating an LDIF file to be used to initialize a replica. Each entry will contain the nsuniqueID operational attribute.
<i>SLAPI_DUMP_NOOPATTRS</i>	By default, certain operational attributes (such as creatorName , modifiersName , createTimestamp , modifyTimestamp) may be included in the output. With this option, no operational attributes will be included.
<i>SLAPI_DUMP_NOWRAP</i>	By default, lines will be wrapped as defined in the LDIF specification. With this option, line wrapping is disabled.

Returns

This function returns one of the following values:

- The LDIF string representation of the entry you specify.

- **NULL** if an error occurs.

Memory Concerns

When you no longer need to use the string, you should free it from memory by calling the [slapi_ch_free\(\)](#) function.

See Also

- [slapi_entry2str\(\)](#)
- [slapi_str2entry\(\)](#)

24.3. SLAPI_ENTRY_ADD_RDN_VALUES()

Adds the components in an entry's relative distinguished name (RDN) to the entry as attribute values. (For example, if the entry's RDN is **uid=bjensen**, the function adds **uid=bjensen** to the entry as an attribute value.)

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_add_rdn_values( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

e	Entry to which you want to add the RDN attributes.
---	--

Returns

This function returns one of the following values:

- **LDAP_SUCCESS** if the values were successfully added to the entry. The function also returns **LDAP_SUCCESS** if the entry is **NULL**, if the entry's DN is **NULL**, or if the entry's RDN is **NULL**.
- **LDAP_INVALID_DN_SYNTAX** if the DN of the entry cannot be parsed.

Memory Concerns

Free the entry from memory by using the [slapi_ch_free\(\)](#) function, if the entry was allocated by the user.

See Also

[slapi_entry_free\(\)](#)

24.4. SLAPI_ENTRY_ADD_STRING()

Description

This function adds a string value to the existing attribute values in an entry. If the specified attribute does not exist in the entry, the attribute is created with the string value specified. The function doesn't check for duplicate values; it does not check if the string value being

added is already there.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_add_string (Slapi_Entry *e, const char *type, const char
*value);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to which you want to add a string value.
<i>type</i>	Attribute to which you want to add a string value.
<i>value</i>	String value you want to add.

Returns

This function returns 0 when successful; any other value returned signals failure.

Memory Concerns

This routine makes a copy of the parameter *value*. *value* can be **NULL**.

24.5. SLAPI_ENTRY_ADD_VALUE()

Description

This function adds a `Slapi_Value` data value to the existing attribute values in an entry. If the specified attribute does not exist in the entry, the attribute is created with the `Slapi_Value` specified. The function doesn't check for duplicate values, meaning it does not check if the value being added is already there.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_add_value (Slapi_Entry *e, const char *type, const
Slapi_Value *value);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to which you want to add a value.
<i>type</i>	Attribute to which you want to add a value.
<i>value</i>	The Slapi_value data value you want to add to the entry.

Returns

This function returns 0 when successful; any other value returned signals failure.

Memory Concerns

This routine makes a copy of the parameter *value*. *value* can be **NULL**.

24.6. SLAPI_ENTRY_ADD_VALUES_SV()**Description**

This function adds an array of Slapi_Value data values to an attribute. If the attribute does not exist, it is created and given the value contained in the Slapi_Value array.

This function replaces the deprecated **slapi_entry_add_values()** function. This function uses Slapi_Value attribute values instead of the now obsolete berval attribute values.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_add_values_sv( Slapi_Entry *e, const char *type,
    Slapi_Value **vals );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to which you want to add values.
<i>type</i>	Attribute type to which you want to add values.
<i>vals</i>	Array of Slapi_Value data values that you want to add.

Returns

This function returns one of the following values:

- **LDAP_SUCCESS** if the Slapi_Value array is successfully added to the attribute.
- **LDAP_TYPE_OR_VALUE_EXISTS** if any values you are trying to add duplicate an existing value in the attribute.
- **LDAP_OPERATIONS_ERROR** if there are pre-existing duplicate values in the attribute.

Memory Concerns

This routine makes a copy of the parameter *vals*. *vals* can be **NULL**.

See Also

[slapi_entry_add_values_sv\(\)](#)

24.7. SLAPI_ENTRY_ADD_VALUESSET()

Description

This function adds a set of values to an attribute in an entry. The values added are in the form of a `Slapi_ValueSet` data type. If the entry does not contain the attribute specified, it is created with the specified `Slapi_ValueSet` value.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_add_valueset(Slapi_Entry *e, const char *type,
Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to which you want to add values.
<i>type</i>	Attribute type to which you want to add values.
<i>vs</i>	<code>Slapi_ValueSet</code> data value that you want to add to the entry.

Returns

This function returns 0 when successful; any other value returned signals failure.

Memory Concerns

This routine makes a copy of the parameter `vs`. `vs` can be **NULL**.

24.8. SLAPI_ENTRY_ALLOC()

Allocates memory for a new entry of the [Section 14.22, “Slapi_Entry”](#) data type.

Description

This function returns an empty `Slapi_Entry` structure. You can call other frontend functions to set the DN and attributes of this entry.

Syntax

```
#include "slapi-plugin.h"
Slapi_Entry *slapi_entry_alloc();
```

Returns

This function returns a pointer to the newly allocated entry of the data type `Slapi_Entry`. If space cannot be allocated, e.g., no more virtual memory exists, the **slapd** program terminates.

When you are no longer using the entry, you should free it from memory by calling the [slapi_entry_free\(\)](#) function.

Memory Concerns

When you no longer use the entry, free it from memory by calling the [slapi_entry_free\(\)](#) function.

See Also

- [slapi_entry_dup\(\)](#)
- [slapi_entry_free\(\)](#)

24.9. SLAPI_ENTRY_APPLY_MODS()

Applies an array of LDAPMod to a new entry of the [Section 14.22, “Slapi_Entry”](#) data type.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_apply_mods(Slapi_Entry *e, LDAPMod **mods);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to which to apply the modifications.
<i>mods</i>	Pointer to an initialized Slapi_Mods.

Returns

If the modifications are applied cleanly to the entry, then the function returns an **LDAP_SUCCESS** message (0).

If an error occurs, then the function returns the appropriate LDAP error message.

See Also

- [Chapter 33, Functions for LDAPMod Manipulation](#)

24.10. SLAPI_ENTRY_ATTR_DELETE()

Deletes an attribute (and all its associated values) from an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_delete( Slapi_Entry *e, const char *type );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to delete the attribute.
----------	--

<i>type</i>	Attribute type that you want to delete.
-------------	---

Returns

This function returns one of the following values:

- 0 if successful.
- 1 if the specified attribute is not part of the entry.
- -1 if an error occurred.

24.11. SLAPI_ENTRY_ATTR_FIND()

Determines if an entry contains the specified attribute. If the entry contains the attribute, the function returns a pointer to the attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_find( const Slapi_Entry *e, const char *type,
Slapi_Attr **attr );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry that you want to check.
<i>type</i>	Name of the attribute that you want to check.
<i>attr</i>	Pointer to the attribute, if the attribute is in the entry.

Returns

This function returns one of the following values:

- 0 if the entry contains the specified attribute.
- -1 if the entry does not contain the specified attribute.

Memory Concerns

Do not free the returned *attr*. It is a pointer to the internal entry data structure. It is usually wise to make a copy of the returned *attr*, using [slapi_attr_dup\(\)](#), to avoid dangling pointers if the entry is freed while the pointer to *attr* is still being used.

See Also

[slapi_attr_dup\(\)](#)

24.12. SLAPI_ENTRY_ATTR_GET_BOOL()

Gets the value of a given attribute of a given entry as a boolean value.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry_attr_get_bool(const Slapi_Entry* e, const char *type);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to get the boolean value.
<i>type</i>	Attribute type from which you want to get the value.

Returns

This function returns one of the following values:

- true | false
- yes | no
- An integer

Comparisons are case-insensitive (**TRUE**, **true**, and **true** are all the same), and unique substrings can be matched (**t** and **tr** will be interpreted as **true**). If the attribute value is a number, then non-zero numbers are interpreted as **true**, and **0** is interpreted as **false**.

24.13. SLAPI_ENTRY_ATTR_GET_CHARPTR()

Gets the first value of an attribute of an entry as a string.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry_attr_get_charptr(const Slapi_Entry* e, const char
*type);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to get the string value.
<i>type</i>	Attribute type from which you want to get the value.

Returns

This function returns one of the following values:

- A copy of the first value in the attribute.
- **NULL** if the entry does not contain the attribute.

Memory Concerns

When you are done working with this value, free it from memory by calling the [slapi_ch_free\(\)](#) function.

24.14. SLAPI_ENTRY_ATTR_GET_CHARRAY()

Gets the values of a multi-valued attribute of an entry.

Description

This function is very similar to [slapi_entry_attr_get_charptr\(\)](#), except that it returns a **char**** array for multi-valued attributes. The array and all values are copies. Even if the attribute values are not strings, they will still be null terminated so that they can be used safely in a string context. If there are no values, NULL will be returned. Because the array is NULL terminated, the usage should be similar to the sample shown below:

```
char **ary = slapi_entry_attr_get_charray(e, someattr);
int ii;
for (ii = 0; ary && ary[ii]; ++ii) {
    char *strval = ary[ii];
    ...
}
slapi_ch_array_free(ary);
```

Syntax

```
#include "slapi-plugin.h"
char ** slapi_entry_attr_get_charray( const Slapi_Entry* e, const char
*type);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to get the values.
<i>type</i>	Attribute type from which you want to get the values.

Returns

This function returns one of the following values:

- A copy of all the values of the attribute.
- **NULL** if the entry does not contain the attribute or if the attribute has no values.

Memory Concerns

When you are done working with the values, free them from memory by calling the `slapi_ch_array_free()` function.

See Also

[slapi_entry_attr_get_charptr\(\)](#)

24.15. SLAPI_ENTRY_ATTR_GET_INT()

Gets the first value of an attribute in an entry as an integer.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_get_int(const Slapi_Entry* e, const char *type);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to get the integer value.
<i>type</i>	Attribute type from which you want to get the value.

Returns

This function returns one of the following values:

- The first value in the attribute converted to an integer.
- 0 if the entry does not contain the attribute.

24.16. SLAPI_ENTRY_ATTR_GET_LONG()

Gets the first value of an attribute in an entry as a long data type.

Syntax

```
#include "slapi-plugin.h"
long slapi_entry_attr_get_long( const Slapi_Entry* e, const char *type);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to get the long value.
<i>type</i>	Attribute type from which you want to get the value.

Returns

This function returns one of the following values:

- The first value in the attribute converted to a **long** type.
- 0 if the entry does not contain the attribute specified.

24.17. SLAPI_ENTRY_ATTR_GET_UINT()

Gets the first value of an attribute in an entry as an unsigned integer data type.

Syntax

```
#include "slapi-plugin.h"
unsigned int slapi_entry_attr_get_uint( const Slapi_Entry* e, const char
*type);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to get the value.
<i>type</i>	Attribute type from which you want to get the value.

Returns

This function returns one of the following values:

- The first value in the attribute converted to an **unsigned integer**.
- 0 if the entry does not contain the attribute specified.

24.18. SLAPI_ENTRY_ATTR_GET_ULONG()

Gets the first value of an attribute in an entry as a unsigned long data type.

Syntax

```
#include "slapi-plugin.h"
unsigned long slapi_entry_attr_get_ulong( const Slapi_Entry* e, const char
*type);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to get the value.
<i>type</i>	Attribute type from which you want to get the value.

Returns

This function returns one of the following values:

- The first value in the attribute converted to an **unsigned long**.
- 0 if the entry does not contain the attribute specified.

24.19. SLAPI_ENTRY_ATTR_HAS_SYNTAX_VALUE()

Description

This function replaces the deprecated `slapi_entry_attr_hasvalue()` function and takes into consideration the syntax of the attribute type. It determines if an attribute in an entry contains a specified value.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_has_syntax_value(const Slapi_Entry *e, const char
*type, const Slapi_Value *value);
```

Parameters

<i>e</i>	Entry that you want to check.
<i>type</i>	Attribute type that you want to test for the value specified.
<i>value</i>	Value that you want to find in the attribute.

Returns

This function returns one of the following values:

- 1 if the attribute contains the specified value.
- 0 if the attribute does not contain the specified value.

Memory Concerns

value must not be **NULL**.

24.20. SLAPI_ENTRY_ATTR_MERGE_SV()

Description

Adds an array of `Slapi_Value` data values to the existing attribute values in an entry. If the attribute does not exist, it is created with the `Slapi_Value` specified. This function replaces the deprecated `slapi_entry_attr_merge()` function. This function uses `Slapi_Value` attribute values instead of the now obsolete `berval` attribute values.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_merge_sv( Slapi_Entry *e, const char *type,
Slapi_Value **vals );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to which you want to add values.
<i>type</i>	Attribute to which you want to add values.
<i>vals</i>	Array of Slapi_Value data values you want to add.

Returns

This function returns 0 if successful; any other value returned signals failure.

Memory Concerns

This function makes a copy of the parameter *vals*. *vals* can be **NULL**.

24.21. SLAPI_ENTRY_ATTR_REPLACE_SV()

Replaces the values of an attribute with the Slapi_Value data value you specify.

Description

This function replaces existing attribute values in a specified entry with a single Slapi_Value data value. The function first deletes the existing attribute from the entry, then replaces it with the new value specified. This function replaces the deprecated **slapi_entry_attr_replace()** function. This function uses Slapi_Value attribute values instead of the now obsolete berval attribute values.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_replace_sv( Slapi_Entry *e, const char *type,
Slapi_Value **vals );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry in which you want to replace values.
<i>type</i>	Attribute type which will receive the replaced values.
<i>vals</i>	Array containing the Slapi_Value values that should replace the existing values of the attribute.

Returns

This function returns 0 when successful; any other value returned signals failure.

Memory Concerns

This function makes a copy of the parameter *vals*. *vals* can be **NULL**.

See Also

[slapi_entry_attr_replace\(\)](#)

24.22. SLAPI_ENTRY_ATTR_SET_CHARPTR()

Replaces the value or values of an attribute in an entry with a specified string value.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_charptr(Slapi_Entry* e, const char *type, const
char *value);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry in which you want to set the value.
<i>type</i>	Attribute type in which you want to set the value.
<i>value</i>	String value that you want to assign to the attribute.

Memory Concerns

This function makes a copy of the parameter values. *values* can be **NULL**; if so, this function is roughly equivalent to [slapi_entry_attr_delete\(\)](#).

See Also

[slapi_entry_attr_delete\(\)](#)

24.23. SLAPI_ENTRY_ATTR_SET_INT()**Description**

This function will replace the value or values of an attribute with the integer value that you specify. If the attribute does not exist, it is created with the integer value that you specify.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_int(Slapi_Entry* e, const char *type, int l);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry in which you want to set the value.
<i>type</i>	Attribute type in which you want to set the value.
<i>l</i>	Integer value that you want assigned to the attribute.

24.24. SLAPI_ENTRY_ATTR_SET_LONG()

Replaces the value or values of an attribute in an entry with a specified longdata type value.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_long(Slapi_Entry* e, const char *type, long l);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry in which you want to set the value.
<i>type</i>	Attribute type in which you want to set the value.
<i>l</i>	Long integer value that you want assigned to the attribute.

24.25. SLAPI_ENTRY_ATTR_SET_UINT()

Description

This function will replace the value or values of an attribute with the **unsigned integer** value that you specify. If the attribute does not exist, it is created with the unsigned integer value you specify.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_uint(Slapi_Entry* e, const char *type, unsigned
int l);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry in which you want to set the value.
<i>type</i>	Attribute type in which you want to set the value.
<i>l</i>	Unsigned integer value that you want assigned to the attribute.

24.26. SLAPI_ENTRY_ATTR_SET_ULONG()

Description

This function will replace the value or values of an attribute with the unsigned long value that you specify. If the attribute does not exist, it is created with the unsigned long value that you specify.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_ulong(Slapi_Entry* e, const char *type, unsigned long l);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry in which you want to set the value.
<i>type</i>	Attribute type in which you want to set the value.
<i>l</i>	Unsigned long value that you want assigned to the attribute.

24.27. SLAPI_ENTRY_DELETE_STRING()

Deletes a string value from an attribute in an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_delete_string(Slapi_Entry *e, const char *type, const char *value);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want the string deleted.
----------	---

<i>type</i>	Attribute type from which you want the string deleted.
<i>value</i>	Value of string to delete.

Returns

This function returns 0 when successful; any other value returned signals failure.

24.28. SLAPI_ENTRY_DELETE_VALUES_SV()

Removes an array of Slapi_Value data values from an attribute in an entry.

Description

This function removes an attribute/valueset from an entry. Both the attribute and its Slapi_Value data values are removed from the entry. If you supply aSlapi_Value whose value is **NULL**, the function will delete the specified attribute from the entry. In either case, the function returns **LDAP_SUCCESS**.

This function replaces the deprecated **slapi_entry_delete_values()** function. This function uses Slapi_Value attribute values instead of the now obsolete berval attribute values.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_delete_values_sv( Slapi_Entry *e, const char *type,
    Slapi_Value **vals );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to delete values.
<i>type</i>	Attribute from which you want to delete values.
<i>vals</i>	Array of Slapi_Value data values that you want to delete.

Returns

This function returns **LDAP_SUCCESS** if the specified attribute and the array ofSlapi_Value data values are deleted from the entry.

If the specified attribute contains a **NULL** value, the attribute is deleted from the attribute list, and the function returns **LDAP_NO_SUCH_ATTRIBUTE**. As well, if the attribute is not found in the list of attributes for the specified entry, the function returns **LDAP_NO_SUCH_ATTRIBUTE**.

If there is an operational error during the processing of this call such as a duplicate value found, the function will return **LDAP_OPERATIONS_ERROR**. If this occurs, please report the problem to the Red Hat technical support.

Memory Concerns

The *vals* parameter can be **NULL**, in which case this function does nothing.

See Also

[slapi_entry_delete_values\(\)](#)

24.29. SLAPI_ENTRY_DUP()

Makes a copy of an entry, its DN, and its attributes.

Description

This function returns a copy of an existing [Section 14.22, “Slapi_Entry”](#) structure. You can call other frontend functions to change the DN and attributes of this entry.

Syntax

```
#include "slapi-plugin.h"
Slapi_Entry *slapi_entry_dup( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

<i>e</i>	Entry that you want to copy.
----------	------------------------------

Returns

This function returns the new copy of the entry. If the structure cannot be duplicated, for example, if no more virtual memory exists, the **slapd** program terminates.

Memory Concerns

When you are no longer using the entry, free it from memory by calling the [slapi_entry_free\(\)](#) function.

See Also

[slapi_entry_alloc\(\)](#) [slapi_entry_free\(\)](#)

24.30. SLAPI_ENTRY_FIRST_ATTR()

Finds the first attribute in an entry. If you want to iterate through the attributes in an entry, use this function in conjunction with the [slapi_entry_next_attr\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_first_attr( const Slapi_Entry *e, Slapi_Attr **attr );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to get the attribute.
<i>attr</i>	Pointer to the first attribute in the entry.

Returns

Returns 0 when successful; any other value returned signals failure.

Memory Concerns

Do not free the returned *attr*. This is a pointer into the internal entry data structure. If you need a copy, use [slapi_attr_dup\(\)](#).

See Also

[slapi_attr_dup\(\)](#)

24.31. SLAPI_ENTRY_FREE()

Frees an entry, its DN, and its attributes from memory.

Description

Call this function to free an entry that you have allocated by using the [slapi_entry_alloc\(\)](#) function or the [slapi_entry_dup\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_free( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

<i>e</i>	Entry that you want to free. If NULL , no action occurs.
----------	---

Memory Concerns

To free entries, always use this function instead of using **slapi_ch_free()** or **free()**.

See Also

- [slapi_entry_alloc\(\)](#)
- [slapi_entry_dup\(\)](#)

24.32. SLAPI_ENTRY_GET_DN()

Gets the distinguished name (DN) of the specified entry.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry_get_dn( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

e	Entry from which you want to get the DN.
---	--

Returns

This function returns the DN of the entry. This returns a pointer to the actual DN in the entry, not a copy of the DN. You should not free the DN unless you plan to replace it by calling [slapi_entry_set_dn\(\)](#).

Memory Concerns

Use [slapi_ch_free\(\)](#) if you are replacing the DN with [slapi_entry_set_dn\(\)](#).

See Also

- [slapi_ch_free\(\)](#)
- [slapi_entry_set_dn\(\)](#)

24.33. SLAPI_ENTRY_GET_DN_CONST()

Returns as a **const** the **DN** value of the entry that you specify.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_entry_get_dn_const( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

e	Entry from which you want to get the DN as a constant.
---	---

Returns

This function returns one of the following values:

- The **DN** of the entry that you specify. The **DN** is returned as a **const**; you are not able to modify the **DN** value.
- The **NDN** value of `Slapi_DN` if the **DN** of the `Slapi_DN` object is **NULL**.

Memory Concerns

Never free this value.

24.34. SLAPI_ENTRY_GET_NDN()

Returns the normalized **DN** from the entry that you specify.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry_get_ndn( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

e	Entry from which you want to obtain the normalized DN .
---	--

Returns

This function returns the normalized **DN** from the entry that you specify. If the entry you specify does not contain a normalized **DN**, one is created through the processing of this function.

Memory Concerns

Never free this value.

24.35. SLAPI_ENTRY_GET_SDN()

Returns the Slapi_DN object from the entry that you specify.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_entry_get_sdn( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

e	Entry from which you want to get the Slapi_DN object.
---	---

Returns

Returns the Slapi_DN object from the entry that you specify.

Memory Concerns

Never free this value. If you need a copy, use **slapi_sdn_dup()**.

See Also

[slapi_sdn_dup\(\)](#)

24.36. SLAPI_ENTRY_GET_SDN_CONST()

Returns as a **const** the value of the `Slapi_DN` object from the entry that you specify.

Syntax

```
#include "slapi-plugin.h"
const Slapi_DN *slapi_entry_get_sdn_const ( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

<code>e</code>	Entry from which you want to get the <code>Slapi_DN</code> object.
----------------	--

Returns

This function returns as a **const** the value of the `Slapi_DN` object from the entry that you specify.

Memory Concerns

Never free this value. If you need a copy, use [slapi_sdn_dup\(\)](#).

See Also

[slapi_sdn_dup\(\)](#)

24.37. SLAPI_ENTRY_GET_UNIQUEID()

Gets the unique ID value of the entry.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_entry_get_uniqueid( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

<code>e</code>	Entry from which you want obtain the unique ID.
----------------	---

Returns

This function returns the unique ID value of the entry specified.

Memory Concerns

Never free this value. If you need a copy, use [slapi_ch_strdup\(\)](#).

See Also

[slapi_ch_strdup\(\)](#)

24.38. SLAPI_ENTRY_HAS_CHILDREN()

This function determines if the specified entry has child entries.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_has_children(const Slapi_Entry *e);
```

Parameters

This function takes the following parameter:

<i>e</i>	Entry that you want to test for child entries.
----------	--

Returns

This function returns one of the following values:

- 1 if the entry you supply has children entries.
- 0 if the entry you supply has no children entries.

24.39. SLAPI_ENTRY_INIT()

Initializes the values of an entry with the DN and attribute value pairs you supply.

Description

This function initializes the attributes and the corresponding attribute values of an entry. Also, during the course of processing, the unique ID of the entry is set to **NULL**, and the flag value is set to **0**.

Use this function to initialize a Slapi_Entry pointer.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_init(Slapi_Entry *e, Slapi_DN *dn, Slapi_Attr *a);
```

Parameters

This function takes the following parameters:

<i>e</i>	The entry you want to initialize.
<i>dn</i>	The DN of the entry to initialize.
<i>a</i>	Initialization list of attribute value pairs, supplied as a Slapi_Attr data value.

Memory Concerns

This function should always be used after [slapi_entry_alloc\(\)](#) and never otherwise. For

example:

```
Slapi_Entry *e = slapi_entry_alloc();
slapi_entry_init(e, NULL, NULL);
```

To set the DN in the entry:

```
slapi_sdn_set_dn_passin(slapi_entry_get_sdn(e), dn);
```

In this case, the **dn** argument is not copied but is consumed by the function. To copy the argument, see the following example:

```
Slapi_DN *dn = slapi_ch_strdup(some_dn);
Slapi_Entry *e = slapi_entry_alloc();
slapi_entry_init(e, dn, NULL);
```

dn is not freed in this context but will eventually be freed when [slapi_entry_free\(\)](#) is called.

See Also

- [slapi_entry_free\(\)](#)
- [slapi_entry_alloc\(\)](#)
- [slapi_ch_strdup\(\)](#)

24.40. SLAPI_ENTRY_MERGE_VALUES_SV()

Merges (adds) and array of Slapi_Value data values to a specified attribute in an entry. If the entry does not contain the attribute specified, the attribute is created with the value supplied.

Description

This function adds additional Slapi_Value data values to the existing values contained in an attribute. If the attribute type does not exist, it is created.

If the specified attribute exists in the entry, the function merges the value specified and returns **LDAP_SUCCESS**. If the attribute is not found in the entry, the function creates it with the Slapi_Value specified and returns **LDAP_NO_SUCH_ATTRIBUTE**.

If this function fails, it leaves the values for *type* within a pointer to *e* in an indeterminate state. The present valueset may be truncated.

```
rc = delete_values_sv_internal( e, type, vals, 1 /* Ignore Errors */
);
```

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_merge_values_sv( Slapi_Entry *e, const char *type,
Slapi_Value **vals );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry into which you want to merge values.
<i>type</i>	Attribute type that contains the values you want to merge.
<i>vals</i>	Values that you want to merge into the entry. Values are of type <code>Slapi_Value</code> .

Returns

This function returns one of the following values:

- `LDAP_SUCCESS`.
- `LDAP_NO_SUCH_ATTRIBUTE`.

Memory Concerns

This function makes a copy of *vals*. *vals* can be `NULL`.

24.41. SLAPI_ENTRY_NEXT_ATTR()

Finds the next attribute after *prevattr* in an entry. To iterate through the attributes in an entry, use this function in conjunction with the [slapi_entry_first_attr\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_next_attr( const Slapi_Entry *e, Slapi_Attr *prevattr,
Slapi_Attr **attr );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which you want to get the attribute.
<i>prevattr</i>	Previous attribute in the entry.
<i>attr</i>	Pointer to the next attribute after <i>prevattr</i> in the entry.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if *prevattr* was the last attribute in the entry.

Memory Concerns

Never free the returned *attr*. Use [slapi_attr_dup\(\)](#) to make a copy if a copy is needed.

See Also

[slapi_attr_dup\(\)](#)

24.42. SLAPI_ENTRY_RDN_VALUES_PRESENT()

Determines whether the values in an entry's relative distinguished name (RDN) are also present as attribute values. For example, if the entry's RDN is **cn=Barbara Jensen**, the function determines if the entry has the **cn** attribute with the value **Barbara Jensen**.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_rdn_values_present( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

<i>e</i>	Entry from which you want to get the attribute.
----------	---

Returns

The function returns one of the following values:

- 1 if the values in the RDN are present in attributes of the entry.
- 0 if the values are not present.

24.43. SLAPI_ENTRY_SCHEMA_CHECK()

Determines whether the specified entry complies with the schema for its object class.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_schema_check( Slapi_PBlock *pb, Slapi_Entry *e );
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block.
<i>e</i>	Entry of which you want to check the schema.

Returns

The function returns one of the following values:

- 0 if the entry complies with the schema or if schema checking is turned off. The function also returns 0 if the entry has additional attributes not allowed by the schema and has the object class **extensibleObject**.
- 1 if the entry is missing the **objectclass** attribute, if it is missing any required attributes, if it has any attributes not allowed by the schema but does not have the object class **extensibleObject**, or if the entry has multiple values for a single-valued attribute.

Memory Concerns

The *pb* argument can be **NULL**. It is only used to get the **SLAPI_IS_REPLACED_OPERATION** flag. If that flag is present, no schema checking is done.

24.44. SLAPI_ENTRY_SET_DN()

Sets the distinguished name (DN) of an entry.

Description

This function sets the DN pointer in the specified entry to the DN that you supply.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_set_dn( Slapi_Entry *e, Slapi_DN *dn );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to which you want to assign the DN.
<i>dn</i>	Distinguished name to assign to the entry.

Memory Concerns

The *dn* will be freed eventually when [slapi_entry_free\(\)](#) is called.

A copy of *dn* should be passed. For example:

```
Slapi_DN *dn = slapi_ch_strdup(some_dn);
slapi_entry_set_dn(e, dn);
```

The old *dn* will be freed as a result of this call. Do not pass in **NULL** value.

See Also

- [slapi_entry_free\(\)](#)
- [slapi_entry_get_dn\(\)](#)

24.45. SLAPI_ENTRY_SET_SDN()

Description

This function sets the value for the `Slapi_DN` object in the entry you specify.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_set_sdn( Slapi_Entry *e, const Slapi_DN *sdn );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to which you want to set the value of the <code>Slapi_DN</code> .
<i>sdn</i>	The specified <code>Slapi_DN</code> value that you want to set.

Memory Concerns

This function makes a copy of the *sdn* argument.

24.46. SLAPI_ENTRY_SET_UNIQUEID()

Description

This function replaces the unique ID value of the entry with the *uniqueid* value that you specify. In addition, the function adds **SLAPI_ATTR_UNIQUEID** to the attribute list and gives it the unique ID value supplied. If the entry already contains a **SLAPI_ATTR_UNIQUEID** attribute, its value is updated with the new value supplied.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_set_uniqueid( Slapi_Entry *e, char *uniqueid );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry for which you want to generate a description.
<i>uniqueid</i>	The unique ID value to which you want to assign the entry.

Memory Concerns

Do not free the *uniqueid* after calling this function. The value will eventually be freed when [slapi_entry_free\(\)](#) is called.

You should pass in a copy of the value because this function will consume the value passed in. For example:

```
char *uniqueid = slapi_ch_strdup(some_uniqueid);
slapi_entry_set_uniqueid(e, uniqueid);
```

Do not pass in a **NULL** for *uniqueid*.

See Also

[slapi_entry_free\(\)](#)

24.47. SLAPI_ENTRY_SIZE()

This function returns the approximate size of an entry, rounded to the nearest 1k. This can be useful for checking cache sizes, estimating storage needs, and so on.

Description

When determining the size of an entry, only the sizes of the attribute values are counted; the size of other entry values (such as the size of attribute names, variously-normalized DNS, or any metadata) are not included in the size returned. It is assumed that the size of the metadata, **et al.**, is well enough accounted for by the rounding of the size to the next largest 1k. This holds true especially in larger entries, where the actual size of the attribute values far outweighs the size of the metadata.

Syntax

```
#include "slapi-plugin.h"
size_t slapi_entry_size(Slapi_Entry *e);
```

Parameters

This function takes the following parameter:

e	Entry from which you want the size returned.
---	--

Returns

This function returns one of the following values:

- The size of the entry, rounded to the nearest 1k. The value returned is a **size_t** data type with a **u_long** value.
- A size of 1k if the entry is empty.

When determining the size of the entry, both deleted values and deleted attributes are included in the count.

24.48. SLAPI_IS_ROOTDSE()

This function determines if an entry is the root DSE. The root DSE is a special entry that contains information about the Directory Server, including its capabilities and configuration.

Syntax

```
#include "slapi-plugin.h"
int slapi_is_rootdse ( const Slapi_DN *dn );
```

■

Parameters

This function takes the following parameter:

<i>dn</i>	The DN that you want to test to see if it is the root DSE entry.
-----------	--

Returns

This function returns one of the following values:

- 1 if *dn* is the root DSE.
- 0 if *dn* is not the root DSE.

24.49. SLAPI_STR2ENTRY()

Converts an LDIF description of a directory entry (a string value) into an entry of the `Slapi_Entry` type.

Description

A directory entry can be described by a string in LDIF format; for details, see [Section 5.5.2, “Converting Between Entries and Strings”](#).

Calling the `slapi_str2entry()` function converts a string description in this format to a `Slapi_Entry` structure, which you can pass to other API functions.

**NOTE**

This function modifies the string argument *s*. If you still need to use this string value, you should make a copy of this string before calling `slapi_str2entry()`.

If an error occurred during the conversion process, the function returns `NULL` instead of the entry.

When you are done working with the entry, you should call the [slapi_entry_free\(\)](#) function.

To convert an entry to a string description, call the [slapi_filter_free\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
Slapi_Entry *slapi_str2entry( char *s, int flags );
```

Parameters

This function takes the following parameters:

<i>s</i>	Description of an entry that you want to convert to Section 14.22, “Slapi_Entry” .
----------	--

<i>flags</i>	One or more flags specifying how the entry should be generated
--------------	--

The value of the *flags* argument can be one of the following values:

<i>SLAPI_STR2ENTRY_REMOVEDUPVALS</i>	Removes any duplicate values in the attributes of the entry.
<i>SLAPI_STR2ENTRY_ADDDRDNVALS</i>	Adds the relative distinguished name (RDN) components (for example, uid=bjensen) as attributes of the entry.

Returns

This function returns one of the following values:

- A pointer to the `Slapi_Entry` structure representing the entry.
- **NULL** if the string cannot be converted; for example, if no DN is specified in the string.

See Also

[slapi_filter_free\(\)](#)

CHAPTER 25. FUNCTIONS RELATED TO ENTRY FLAGS

This chapter contains reference information on functions that are specific to entry flags.

Table 25.1. Entry Flags

Function	Description
slapi_entry_clear_flag()	Clears a flag for a specified entry.
slapi_entry_flag_is_set()	Checks if certain flags are set in an entry.
slapi_entry_set_flag()	Sets a flag for an entry.

25.1. SLAPI_ENTRY_CLEAR_FLAG()

Clears a flag for a specified entry.

Description

In this release of Directory Server, the only external flag that can be set is **SLAPI_ENTRY_FLAG_TOMBSTONE**. This flag means that the entry is a **tombstone** entry. More flags may be exposed in future releases. Do not use your own flags.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_clear_flag( Slapi_Entry *e, unsigned char flag);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry in which you want to clear the flag settings.
<i>flag</i>	Flag that you want to clear.

See Also

- [slapi_entry_flag_is_set\(\)](#)
- [slapi_entry_set_flag\(\)](#)

25.2. SLAPI_ENTRY_FLAG_IS_SET()

Determines if certain flags are set for a specified entry.

Description

In this release of Directory Server, the only external flag that can be set is **SLAPI_ENTRY_FLAG_TOMBSTONE**. This flag means that the entry is a tombstone entry. More flags may be exposed in future releases. You should not use your own flags.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_flag_is_set( const Slapi_Entry *e, unsigned char flag );
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry in which you want to check the flag settings.
<i>flag</i>	Flag of which you want to check for presence.

Returns

This function returns one of the following values:

- 0 if the flag is not set.
- The value of the flag if it is set.

See Also

- [slapi_entry_clear_flag\(\)](#)
- [slapi_entry_set_flag\(\)](#)

25.3. SLAPI_ENTRY_SET_FLAG()

Sets a flag for a specified entry.

Description

In current versions of Directory Server, the only external flag that can be set is **SLAPI_ENTRY_FLAG_TOMBSTONE**. This flag means that the entry is a tombstone entry. More flags may be exposed in future releases. Do not use your own flags.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_set_flag( Slapi_Entry *e, unsigned char flag);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry for which you want to set the flags.
<i>flag</i>	Flag that you want to set.

See Also

- [slapi_entry_clear_flag\(\)](#)
- [slapi_entry_flag_is_set\(\)](#)

CHAPTER 26. FUNCTIONS FOR DEALING WITH FILTERS

This chapter contains reference information on filter routines.

Table 26.1. Filter Routines

Function	Description
<code>slapi_filter_apply()</code>	Applies a function to each simple filter component within a complex filter.
<code>slapi_filter_compare()</code>	Determines if two filters are identical.
<code>slapi_filter_dup()</code>	Duplicates the specified filter.
<code>slapi_filter_free()</code>	Frees the specified filter.
<code>slapi_filter_get_attribute_type()</code>	Gets the attribute type for all simple filter choices.
<code>slapi_filter_get_ava()</code>	Gets the attribute type and the value from the filter.
<code>slapi_filter_get_choice()</code>	Gets the type of the specified filter.
<code>slapi_filter_get_subfilt()</code>	Gets the substring values from the filter.
<code>slapi_filter_get_type()</code>	Gets the attribute type specified in the filter.
<code>slapi_filter_join()</code>	Joins two specified filters.
<code>slapi_filter_join_ex()</code>	Recursively joins two specified filters.
<code>slapi_filter_list_first()</code>	Gets the first filter that makes up the specified filter.
<code>slapi_filter_list_next()</code>	Gets the next filter.
<code>slapi_filter_test()</code>	Determines if the specified entry matches a particular filter.
<code>slapi_filter_test_ext()</code>	Determines if an entry matches a given filter.
<code>slapi_filter_test_simple()</code>	Determines if an entry matches a filter.

Function	Description
<code>slapi_find_matching_paren()</code>	Finds the matching right parentheses in a string, corresponding to the left parenthesis to which the string currently points.
<code>slapi_str2filter()</code>	Converts a string description of a search filter into a filter of the <code>Slapi_Filter</code> type.
<code>slapi_vattr_filter_test()</code>	Tests a filter against a single entry.

26.1. SLAPI_FILTER_APPLY()

Applies a function to each simple filter component within a complex filter; a simple filter is anything other than AND, OR, or NOT.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_apply( struct slapi_filter *f, FILTER_APPLY_FN fn, void
*arg, int *error_code );
```

Parameters

This function takes the following parameters:

<i>f</i>	Filter on which the function is to be applied.
<i>fn</i>	Function to apply.
<i>arg</i>	Argument to the function (<i>fn</i>).
<i>error_code</i>	Pointer to error code of <i>fn</i> , which can be accessed by calling function. Possible values <code>slapi_filter_apply()</code> may set in <i>error_code</i> include <code>SLAPI_FILTER_UNKNOWN_FILTER_TYPE</code> . A <code>FILTER_APPLY_FN</code> should return <code>_STOP</code> or <code>_CONTINUE</code> only.

Returns

This function returns an integer. Possible return values for **`slapi_filter_apply()`** include:

- **`SLAPI_FILTER_SCAN_NOMORE`** indicates success in traversing the entire filter.
- **`SLAPI_FILTER_SCAN_STOP`** indicates premature abort.
- **`SLAPI_FILTER_SCAN_CONTINUE`** indicates continue scanning.

- **SLAPI_FILTER_SCAN_ERROR** indicates an occurred during the traverse and the scan is aborted. In this case, *error_code* can be checked for more details; currently, the only error is **SLAPI_FILTER_UNKNOWN_FILTER_TYPE**.

26.2. SLAPI_FILTER_COMPARE()

Description

This function allows you to determine if two filters are identical and/or are allowed to be in a different order.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_compare(struct slapi_filter *f1, struct slapi_filter
*f2);
```

Parameters

This function takes the following parameters:

<i>f1</i>	First filter to compare.
<i>f2</i>	Second filter to compare.

Returns

This function returns one of the following values:

- 0 if the two filters are identical.
- A value other than 0 if the two filters are not identical.

26.3. SLAPI_FILTER_DUP()

Creates a duplicate of the specified filter.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_dup(Slapi_Filter *f);
```

Parameters

This function takes the following parameter:

<i>f</i>	Filter to duplicate.
----------	----------------------

Returns

This function returns a pointer to the duplicated filter if successful; otherwise, it returns NULL.

26.4. SLAPI_FILTER_FREE()

Frees the specified filter and (optionally) the set of filters that comprise it. For example, the set of filters in an **LDAP_FILTER_AND** type filter.

Description

This function frees the filter in parameter **f**.

Syntax

```
#include "slapi-plugin.h"
void slapi_filter_free( Slapi_Filter *f, int recurse );
```

Parameters

This function takes the following parameters:

<i>f</i>	Filter that you want to free.
<i>recurse</i>	If 1 , recursively frees all filters that comprise this filter. If 0 , only frees the filter specified by f .

Memory Concerns

Filters created using **slapi_str2filter()** must be freed after using this function. Filters extracted from a pblock using **slapi_pblock_get(pb,SLAPI_SEARCH_FILTER, &filter)** must not be freed.

See Also

- [slapi_str2filter\(\)](#)
- [slapi_pblock_get\(\)](#)

26.5. SLAPI_FILTER_GET_ATTRIBUTE_TYPE()

Gets the attribute type for all simple filter choices.

Description

This function gets the attribute type for all simple filter choices:

- **LDAP_FILTER_GE**
- **LDAP_FILTER_LE**
- **LDAP_FILTER_APPROX**
- **LDAP_FILTER_EQUALITY**
- **LDAP_FILTER_SUBSTRINGS**
- **LDAP_FILTER_PRESENT**

- **LDAP_FILTER_EXTENDED**
- **LDAP_FILTER_AND**
- **LDAP_FILTER_OR**
- **LDAP_FILTER_NOT**

A filter such as (**mail=foo**), will return the type **mail**.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_attribute_type( Slapi_Filter *f, char **type );
```

Parameters

This function takes the following parameters:

<i>f</i>	Filter from which you wish to get the substring values.
<i>type</i>	Pointer to the attribute type of the filter.

Returns

This function returns the attribute type of the filter.

Memory Concerns

The attribute type is returned in *type* and should not be freed after calling this function. It will be freed at the same time as the *Slapi_Filter* structure when **slapi_filter_free()** is called.

See Also

- [slapi_filter_get_choice\(\)](#)
- [slapi_filter_get_ava\(\)](#)
- [slapi_filter_get_type\(\)](#)
- [slapi_filter_free\(\)](#)

26.6. SLAPI_FILTER_GET_AVA()

Gets the attribute type and the value from the filter.

Description

Filters of the type **LDAP_FILTER_EQUALITY**, **LDAP_FILTER_GE**, **LDAP_FILTER_LE**, and **LDAP_FILTER_APPROX** generally compare a value against an attribute. For example:

```
(cn=Barbara Jensen)
```

This filter finds entries in which the value of the **cn** attribute is equal to **Barbara Jensen**.

The attribute **type** is returned in the parameter **type**, and the value is returned in the parameter **bval**.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_ava( Slapi_Filter *f, char **type, struct berval
**bval );
```

Parameters

This function takes the following parameters:

<i>f</i>	Filter from which you want to get the attribute and value.
<i>type</i>	Pointer to the attribute type of the filter.
<i>bval</i>	Pointer to the address of the berval structure containing the value of the filter.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if the filter is not one of the types listed above.

Memory Concerns

The strings within the parameters *type* and *bval* are direct pointers to memory inside the *Slapi_Filter* and therefore should not be freed after usage. They will be freed when a server entity calls **slapi_filter_free()** after usage of the *Slapi_Filter* structure.

See Also

- [slapi_filter_get_choice\(\)](#)
- [slapi_filter_get_type\(\)](#)
- [slapi_filter_get_attribute_type\(\)](#)

26.7. SLAPI_FILTER_GET_CHOICE()

Gets the type of the specified filter; for example, **LDAP_FILTER_EQUALITY**.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_choice( Slapi_Filter *f );
```

Parameters

This function takes the following parameters:

<i>f</i>	Filter of which you want to get type.
----------	---------------------------------------

Returns

This function returns one of the following values:

- **LDAP_FILTER_AND** (AND filter)

For example: **(*&(ou=Accounting)(l=Sunnyvale)*)**

- **LDAP_FILTER_OR** (OR filter)

For example: **(*|(ou=Accounting)(l=Sunnyvale)*)**

- **LDAP_FILTER_NOT** (NOT filter)

For example: **(*!(l=Sunnyvale)*)**

- **LDAP_FILTER_EQUALITY** (equals filter)

For example: **(*ou=Accounting*)**

- **LDAP_FILTER_SUBSTRINGS** (substring filter)

For example: **(*ou=Account*Department*)**

- **LDAP_FILTER_GE** ("greater than or equal to" filter)

For example: **(*supportedLDAPVersion>=3*)**

- **LDAP_FILTER_LE** ("less than or equal to" filter)

For example: **(*supportedLDAPVersion<=2*)**

- **LDAP_FILTER_PRESENT** (presence filter)

For example: **(*mail=**)**

- **LDAP_FILTER_APPROX** (approximation filter)

For example: **(*ou~=Sales*)**

- **LDAP_FILTER_EXTENDED** (extensible filter)

For example: **(*o:dn:=Example*)**

See Also

- [slapi_filter_get_type\(\)](#)
- [slapi_filter_get_attribute_type\(\)](#)
- [slapi_filter_get_ava\(\)](#)

26.8. SLAPI_FILTER_GET_SUBFILT()

Applies only to filters of the type **LDAP_FILTER_SUBSTRINGS**. Gets the substring values from the filter.

Description

Filters of the type **LDAP_FILTER_SUBSTRINGS** generally compare a set of substrings against an attribute. For example:

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_subfilt( Slapi_Filter *f, char **type, char
**initial, char ***any, char **final );
```

Parameters

This function takes the following parameters:

<i>f</i>	Filter that you want to get the substring values from.
<i>type</i>	Pointer to the attribute type of the filter.
<i>initial</i>	Pointer to the initial substring ("starts with") of the filter.
<i>any</i>	Pointer to an array of the substrings ("contains") for the filter.
<i>final</i>	Pointer to the final substring ("ends with") of the filter.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if the filter is not one of the types listed above.

```
(cn=John*Q*Public)
```

This filter finds entries in which the value of the **cn** attribute starts with **John**, contains **Q**, and ends with **Public**.

Call this function to get these substring values as well as the attribute type from this filter. In the case of the example above, calling this function gets the *initial* substring **John**, the *any* substring **Q**, and the *final* substring **Public** in addition to the attribute type **cn**.

See Also

- [slapi_filter_get_attribute_type\(\)](#)

- [slapi_filter_get_ava\(\)](#)
- [slapi_filter_get_choice\(\)](#)

26.9. SLAPI_FILTER_GET_TYPE()

Applies only to filters of the type **LDAP_FILTER_PRESENT**. Gets the attribute type specified in the filter.

Description

Filters of the type **LDAP_FILTER_PRESENT** generally determine if a specified attribute is assigned a value. For example:

```
(mail=*)
```

This filter finds entries that have a value assigned to the **mail** attribute.

Call this function to get the attribute type from this filter. In the case of the example above, calling this function gets the attribute type **mail**.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_type( Slapi_Filter *f, char **type );
```

Parameters

This function takes the following parameters:

<i>f</i>	Filter from which you want to get the substring values.
<i>type</i>	Pointer to the attribute type of the filter.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if the filter is not one of the types listed above.

Memory Concerns

The string returned in the parameter type must not be freed after calling this function. It will be freed when the structure **Slapi_Filter** is freed by calling **slapi_filter_free()**.

See Also

- [slapi_filter_get_attribute_type\(\)](#)
- [slapi_filter_get_ava\(\)](#)
- [slapi_filter_get_choice\(\)](#)

26.10. SLAPI_FILTER_JOIN()

Joins the two specified filters using one of the following filter types: **LDAP_FILTER_AND**, **LDAP_FILTER_OR**, or **LDAP_FILTER_NOT**. When specifying the filter type **LDAP_FILTER_NOT**, the second filter should be **NULL**.

Description

Filters of the type **LDAP_FILTER_AND**, **LDAP_FILTER_OR**, and **LDAP_FILTER_NOT** generally consist of one or more other filters. For example:

```
(&(ou=Accounting)(l=Sunnyvale))
(|(ou=Accounting)(l=Sunnyvale))
(!(l=Sunnyvale))
```

Each of these examples contain one or more **LDAP_FILTER_EQUALITY** filters.

Call the **slapi_filter_join()** function to create a new filter of the type **LDAP_FILTER_AND**, **LDAP_FILTER_OR**, or **LDAP_FILTER_NOT**.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_join( int ftype, Slapi_Filter *f1, Slapi_Filter
*f2 );
```

Parameters

This function takes the following parameters:

<i>ftype</i>	Type of composite filter you want to create.
<i>f1</i>	First filter that you want to join.
<i>f2</i>	Second filter that you want to join. If <i>ftype</i> is LDAP_FILTER_NOT , specify NULL for this argument.

Returns

This function returns the new filter constructed from the other two filters.

Memory Concerns

The *f1* and *f2* filters are neither copied nor freed during the join process, but the resulting filter will have references pointing to these two filters.

See Also

[slapi_filter_join\(\)](#) uses [slapi_filter_join_ex\(\)](#) with the **recurse_always** argument being **1**.

26.11. SLAPI_FILTER_JOIN_EX()

Recursively joins the two specified filters using one of the following filter types: **LDAP_FILTER_AND**, **LDAP_FILTER_OR**, or **LDAP_FILTER_NOT**. When specifying the filter type **LDAP_FILTER_NOT**, the second filter should be **NULL**.

Description

Filters of the type **LDAP_FILTER_AND**, **LDAP_FILTER_OR**, and **LDAP_FILTER_NOT** generally consist of one or more other filters. For example:

```
(&(ou=Accounting)(l=Sunnyvale))
(|(ou=Accounting)(l=Sunnyvale))
(!(l=Sunnyvale))
```

Each of these examples contain one or more **LDAP_FILTER_EQUALITY** filters.

Call the **slapi_filter_join()** function to create a new filter of the type **LDAP_FILTER_AND**, **LDAP_FILTER_OR**, or **LDAP_FILTER_NOT**.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_join_ex( int ftype, Slapi_Filter *f1,
Slapi_Filter *f2, int recurse_always );
```

Parameters

This function takes the following parameters:

<i>ftype</i>	Type of composite filter you want to create.
<i>f1</i>	First filter that you want to join.
<i>f2</i>	Second filter that you want to join. If <i>ftype</i> is LDAP_FILTER_NOT , specify NULL for this argument.
<i>recurse_always</i>	Recursively joins filters <i>f1</i> and <i>f2</i> .

Returns

This function returns the new filter constructed from the other two filters.

Memory Concerns

The *f1* and *f2* filters are neither copied nor freed during the join process, but the resulting filter will have references pointing to these two filters.

See Also

[slapi_filter_join\(\)](#) uses [slapi_filter_join_ex\(\)](#) with **recurse_always** argument set to **1**.

26.12. SLAPI_FILTER_LIST_FIRST()

Applies only to filters of the types **LDAP_FILTER_EQUALITY**, **LDAP_FILTER_GE**, **LDAP_FILTER_LE**, and **LDAP_FILTER_APPROX**. Gets the first filter that makes up the specified filter.

Description

To iterate through all filters that make up a specified filter, use this function in conjunction with the [slapi_filter_list_next\(\)](#) function.

Filters of the type **LDAP_FILTER_AND**, **LDAP_FILTER_OR**, and **LDAP_FILTER_NOT** generally consist of one or more other filters. For example, if the filter is:

```
(&(ou=Accounting)(l=Sunnyvale))
```

the first filter in this list is:

```
(ou=Accounting)
```

Call the **slapi_filter_list_first()** function to get the first filter in the list.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_list_first( Slapi_Filter *f );
```

Parameters

This function takes the following parameter:

<i>f</i>	Filter of which you want to get the first component.
----------	--

Returns

This function returns the first filter that makes up the specified filter *f*.

Memory Concerns

No duplication of the filter is done, so this filter should not be freed independently of the original filter.

See Also

[slapi_filter_list_next\(\)](#)

26.13. SLAPI_FILTER_LIST_NEXT()

Applies only to filters of the types **LDAP_FILTER_EQUALITY**, **LDAP_FILTER_GE**, **LDAP_FILTER_LE**, and **LDAP_FILTER_APPROX**. Gets the next filter (following *fprev*) that makes up the specified filter *f*.

Description

To iterate through all filters that make up a specified filter, use this function in conjunction with the [slapi_filter_list_first\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_list_next( Slapi_Filter *f, Slapi_Filter *fprev
);
```

Parameters

This function takes the following parameters:

<i>f</i>	Filter from which you want to get the next component (after <i>fprev</i>).
<i>fprev</i>	Filter within the specified filter <i>f</i> .

Returns

This function returns the next filter (after *fprev*) that makes up the specified filter *f*.

Filters of the type **LDAP_FILTER_AND**, **LDAP_FILTER_OR**, and **LDAP_FILTER_NOT** generally consist of one or more other filters. For example, if the filter is:

```
(&(ou=Accounting)(l=Sunnyvale))
```

the next filter after (**ou=Accounting**) in this list is:

```
(l=Sunnyvale)
```

Call the **slapi_filter_list_next()** function to get the filters from this list.

Memory Concerns

No duplication of the filter is done, so this filter should not be freed independently of the original filter.

See Also

[slapi_filter_list_first\(\)](#)

26.14. SLAPI_FILTER_TEST()

Determines if the specified entry matches a particular filter.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_test( Slapi_PBlock *pb, Slapi_Entry *e, Slapi_Filter *f,
int verify_access );
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block.
<i>e</i>	Entry that you want to test.
<i>f</i>	Filter that you want to test the entry against.
<i>verify_access</i>	If 1 , verifies that the current user has access rights to search the specified entry. If 0 , bypasses any access control.

Returns

This function returns one of the following values:

- 0 if the entry matched the filter or if the specified filter is NULL.
- -1 if the filter type is unknown.
- A positive value (an LDAP error code) if an error occurred.

See Also

- [slapi_filter_test_simple\(\)](#)
- [slapi_filter_test_ext\(\)](#)

26.15. SLAPI_FILTER_TEST_EXT()

Determines if an entry matches a given filter.

Description

This function allows you to determine if an entry matches a given filter and/or that the current user has the permission to access the entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_test_ext( Slapi_PBlock *pb, Slapi_Entry *e, Slapi_Filter
*f,int verify_access, int only_test_access)
```

Parameters

This function takes the following parameters:

<i>pb</i>	pblock from which the user is extracted.
<i>e</i>	The entry on which filter matching must be verified.
<i>f</i>	The filter used for filter matching.

<i>verify_access</i>	0 when access checking is not to be done. 1 when access checking must be done.
<i>only_test_access</i>	0 when filter matching must be done 1 when filter matching must not be done.

Returns

This function returns one of the following values:

- 0 if the entry matched the filter or if the specified filter is **NULL**.
- -1 if the filter type is unknown or if the entry does not match the filter.
- A positive value (an LDAP error code) if an error occurred or if the current user does not have access rights to search the specified entry.

See Also

- [slapi_filter_test_simple\(\)](#)
- [slapi_filter_test\(\)](#)

26.16. SLAPI_FILTER_TEST_SIMPLE()

Description

This function allows you to check if entry *e* matches filter *f*.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_test_simple( Slapi_Entry *e, Slapi_Filter *f);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry that you wish to test.
<i>f</i>	Filter to match the entry against.

Returns

This function returns one of the following values:

- 0 if the entry matched the filter or if the specified filter is **NULL**.
- -1 if the filter type is unknown or if the entry does not match the filter.
- A positive value (an LDAP error code) if an error occurred.

See Also

- [slapi_filter_test\(\)](#)
- [slapi_filter_test_ext\(\)](#)

26.17. SLAPI_FIND_MATCHING_PAREN()

Finds the matching right parentheses in a string, corresponding to the left parenthesis to which the string currently points.

Syntax

```
#include "slapi-plugin.h"
char *slapi_find_matching_paren( const char *str )
```

Parameters

This function takes the following parameter:

<i>str</i>	String containing the parentheses.
------------	------------------------------------

Returns

This function returns a pointer to the matching right parenthesis in the specified string.

26.18. SLAPI_STR2FILTER()

Converts a string description of a search filter into a filter of the `Slapi_Filter` type.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_str2filter( char *str );
```

Parameters

This function takes the following parameter:

<i>str</i>	String description of a search filter.
------------	--

Returns

This function returns one of the following values:

- A pointer to the `Slapi_Filter` structure representing the search filter.
- **NULL** if the string cannot be converted; for example, if an empty string is specified or if the filter syntax is incorrect.

When you are done working with this filter, you should free the `Slapi_Filter` structure by calling [slapi_filter_free\(\)](#).

26.19. SLAPI_VATTR_FILTER_TEST()

Tests a filter against a single entry.

Description

This function supports the case where the filter specifies virtual attributes. Performance for a real-attribute-only filter is the same as that for [slapi_filter_test\(\)](#).

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_filter_test( Slapi_PBlock *pb, Slapi_Entry *e, struct
slapi_filter *f, int verify_access);
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block containing information about the filter.
<i>e</i>	Entry against which the filter is to be tested.
<i>f</i>	Filter against which the entry is to be tested.
<i>verify_access</i>	Access control: <ul style="list-style-type: none">• 0 if access checking is not to be done; that is, bypass any access control.• 1 if access checking must be done; that is, verify that the current user has access rights to search the specified entry.

Returns

This function returns one of the following values:

- 0 if the filter matched.
- -1 if the filter did not match.
- An LDAP error code (an integer greater than zero) if an error occurs.

See Also

[slapi_filter_test\(\)](#)

CHAPTER 27. FUNCTIONS SPECIFIC TO EXTENDED OPERATION

This chapter contains reference information on routines for dealing with extended operations.

Table 27.1. Extended Operation Routines

Function	Description
slapi_get_supported_extended_ops_copy()	Gets a copy of the object IDs (OIDs) of the extended operations.

27.1. SLAPI_GET_SUPPORTED_EXTENDED_OPS_COPY()

Gets a copy of the object IDs (OIDs) of the extended operations.

Description

This function replaces the deprecated `slapi_get_supported_extended_ops()` function from earlier releases as `slapi_get_supported_extended_ops()` was not multi-thread safe.

This function gets a copy of the object IDs (OIDs) of the extended operations supported by the server. You can register new extended operations by putting the OID in the `SLAPI_PLUGIN_EXT_OP_OIDLIST` parameter and calling `slapi_block_set()`.

Syntax

```
#include "slapi-plugin.h"
char **slapi_get_supported_extended_ops_copy ( void );
```

Parameters

This function takes no parameters.

Returns

This function returns a pointer to an array of the OIDs of the extended operations supported by the server.

Memory Concerns

The array returned by this function should be freed by calling the `slapi_ch_array_free()` function.

See Also

- [slapi_pblock_set\(\)](#)
- [slapi_ch_array_free\(\)](#)

CHAPTER 28. FUNCTIONS SPECIFIC TO BIND METHODS

This chapter contains reference information on bind routines, including SASL.

Table 28.1. Bind Routines

Function	Description
<code>slapi_add_auth_response_control()</code>	Supplies authentication information from an LDAP bind operation.
<code>slapi_get_supported_saslmechanisms_copy()</code>	Gets an array of the names of the supported Simple Authentication and Security Layer (SASL) methods.
<code>slapi_get_supported_saslmechanisms_copy()</code>	Gets an array of the names of the supported Simple Authentication and Security Layer (SASL) methods.
<code>slapi_register_supported_saslmechanism()</code>	Registers the specified Simple Authentication and Security Layer (SASL) method with the server.

28.1. SLAPI_ADD_AUTH_RESPONSE_CONTROL()

Supplies authentication information from an LDAP bind operation based on the bind DN and passes back the actual bind identity.

Syntax

```
#include "slapi-plugin.h"
int slapi_add_auth_response_control ( Slapi_PBlock *pb, const char *binddn
);
```

Parameters

This function takes the following parameter:

<i>pb</i>	Parameter block.
<i>binddn</i>	The identity of the user specified in the bind operation.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs.

28.2. SLAPI_GET_SUPPORTED_SASLMECHANISMS_COPY()

Gets an array of the names of the supported Simple Authentication and Security Layer (SASL) mechanisms. You can register new SASL mechanisms by calling the `slapi_vattr_values_free()` function.

Syntax

```
#include "slapi-plugin.h"
char ** slapi_get_supported_saslmechanisms_copy( void );
```

Returns

This function returns a pointer to an array of the names of SASL mechanisms supported by the server.

28.3. SLAPI_REGISTER_SUPPORTED_SASLMECHANISM()

Registers the specified Simple Authentication and Security Layer (SASL) mechanism with the server.

Syntax

```
#include "slapi-plugin.h"
void slapi_register_supported_saslmechanism( char *mechanism );
```

Parameters

This function takes the following parameter:

<i>mechanism</i>	Name of the SASL mechanism.
------------------	-----------------------------

See Also

[Chapter 39, Functions for Managing DNs](#).

CHAPTER 29. FUNCTIONS FOR THREAD-SAFE LDAP CONNECTIONS

This chapter contains reference information on functions for thread-safe LDAP connections.

Table 29.1. Thread-Safe LDAP Connection Routines

Function	Description
<code>slapi_ldap_init()</code>	Initializes an LDAP session with another LDAP server.
<code>slapi_ldap_unbind()</code>	Unbinds from another LDAP server and frees the resources contained in the LDAP structure.

29.1. SLAPI_LDAP_INIT()

Initializes an LDAP session with another LDAP server.

Description

This function initializes an LDAP session with another LDAP server. If you want to connect to another LDAP server over TLS or if you want to allow multiple threads to use the same connection, call this function instead of the `ldap_init()` function provided with the **Red Hat Directory SDK**.

This function allocates an LDAP structure containing information about the session, including the hostname 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.

You can specify a list of LDAP servers that you want to attempt to connect to. 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.

If you specify a non-zero value for the *secure* argument, this function initializes the plug-in for TLS and installs the I/O routines for TLS.

If you specify a non-zero value for the *shared* argument, this function installs the server's threading functions and allows multiple threads to share this session (the returned LDAP structure). The Directory Server processes each request in a separate thread. When handling multiple requests, it is possible for the server to call your plug-in function concurrently for different threads.

If you initialize a session by calling this function, make sure to call the `slapi_ldap_unbind()` function (not the `ldap_unbind()` or `ldap_unbind_s()` functions provided with the Directory Server SDK) when you are done with the session.

As the `slapi_ldap_init()` function returns a regular **LDAP ***, you can use the **LDAP C SDK** connect timeout feature for plug-ins. That is, when connecting to an external LDAP server from a plug-in, you can specify a time limit for establishing the connection. To specify the timeout, call `ldap_set_option()` with the `LDAP_X_OPT_CONNECT_TIMEOUT` option after calling `slapi_ldap_init()`, as illustrated in the sample code below:


```

void my_ldap_function( void ) {

    LDAP *ld;
    int to = 5000; /* 5000 milliseconds == 5 second timeout */
    if (( ld = slapi_ldap_init( host, port, 0, 1 )) == NULL ) {
        /* error trying to create an LDAP session */
        return -1;
    }
    if ( ldap_set_option( ld, LDAP_X_OPT_CONNECT_TIMEOUT, &to ) != 0 ) {
        /* error setting timeout */
        slapi_ldap_unbind( ld );
        return -1;
    }
    /* use the handle, e.g., call ldap_search_ext() */
    slapi_ldap_unbind( ld );
    return 0;
}

```

Syntax

```

#include "slapi-plugin.h"
LDAP *slapi_ldap_init( char *ldaphost, int ldapport, int secure, int
shared );

```

Parameters

This function takes the following parameters:

<i>ldaphost</i>	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 to which you want to connect. The names can be in hostname:portnumber format, in which case portnumber overrides the port number specified by the ldapport argument.
<i>ldapport</i>	Port number of the LDAP server.
<i>secure</i>	Determines whether to establish the connection over TLS. Set this to a non-zero value to establish the connection over TLS.
<i>shared</i>	Determines whether the LDAP session (the LDAP structure) can be shared by different threads. Set this to a non-zero value to allow multiple threads to share the session.

Returns

This function 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 **NULL**.

29.2. SLAPI_LDAP_UNBIND()

Unbinds from another LDAP server and frees the resources contained in the LDAP structure.

Description

This function unbinds from another LDAP server. Call this function if you initialized the LDAP session with the [slapi_ldap_init\(\)](#) function. Do not call the `ldap_unbind()` or `ldap_unbind_s()` functions provided with the Red Hat Directory SDK.

Syntax

```
#include "slapi-plugin.h"
void slapi_ldap_unbind( LDAP *ld );
```

Parameters

This function takes the following parameter:

<i>ld</i>	Connection handle, which is a pointer to an LDAP structure containing information about the connection to the LDAP server.
-----------	--

CHAPTER 30. FUNCTIONS FOR LOGGING

This chapter contains reference information on logging routines.

Table 30.1. Logging Routines

Function	Description
<code>slapi_log_error()</code>	Writes a message to the error log for the Directory Server.
<code>slapi_is_loglevel_set()</code>	Checks if loglevel is selected as a log level.

30.1. SLAPI_LOG_ERROR()

Writes a message to the error log for the Directory Server. By default, the error log file is `/var/log/dirsrv/slapd-instance/errors`.

Syntax

```
#include "slapi-plugin.h"
int slapi_log_error( int severity, char *subsystem, char *fmt, ... );
```

Parameters

This function takes the following parameters:

<i>severity</i>	Level of severity of the message. In combination with the severity level specified by the administrator, this determines whether the message is written to the log.
<i>subsystem</i>	Name of the subsystem in which this function is called. The string that you specify here appears in the error log in the following format: subsystem_name:message
<i>fmt, ...</i>	Message that you want written. This message can be in printf() -style format. For example: ..., %s\n,myString);

The *severity* argument corresponds to the Log Level setting selected by in the Server Manager under Server Preferences | LDAP. If a Log Level setting is selected, messages with that severity level are written to the error log. The *severity* argument can have one of the following values:

<code>SLAPI_LOG_FATAL</code>	Always written to the error log. This severity level indicates that a fatal error has occurred in the server.
------------------------------	---

<i>SLAPI_LOG_TRACE</i>	Written to the error log if the Log Level setting "Trace function calls" is selected. This severity level is typically used to indicate what function is being called.
<i>SLAPI_LOG_PACKETS</i>	Written to the error log if the Log Level setting "Packet handling" is selected.
<i>SLAPI_LOG_ARGS</i>	Written to the error log if the Log Level setting "Heavy trace output" is selected.
<i>SLAPI_LOG_CONNS</i>	Written to the error log if the Log Level setting "Connection management" is selected.
<i>SLAPI_LOG_BER</i>	Written to the error log if the Log Level setting "Packets sent/received" is selected.
<i>SLAPI_LOG_FILTER</i>	Written to the error log if the Log Level setting "Search filter processing" is selected.
<i>SLAPI_LOG_CONFIG</i>	Written to the error log if the Log Level setting "Config file processing" is selected.
<i>SLAPI_LOG_ACL</i>	Written to the error log if the Log Level setting "Access control list processing" is selected.
<i>SLAPI_LOG_SHELL</i>	Written to the error log if the Log Level setting "Log communications with shell backends" is selected.
<i>SLAPI_LOG_PARSE</i>	Written to the error log if the Log Level setting "Log entry parsing" is selected.
<i>SLAPI_LOG_HOUSE</i>	Written to the error log if the Log Level setting "Housekeeping" is selected.
<i>SLAPI_LOG_REPL</i>	Written to the error log if the Log Level setting "Replication" is selected.
<i>SLAPI_LOG_CACHE</i>	Written to the error log if the Log Level setting "Entry cache" is selected.
<i>SLAPI_LOG_PLUGIN</i>	Written to the error log if the Log Level setting "Plug-ins" is selected. This severity level is typically used to identify messages from server plug-ins.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an unknown severity level is specified.

30.2. SLAPI_IS_LOGLEVEL_SET()

Checks if *loglevel* is selected as a log level.

Description

To specify the level of logging used by the Directory Server, the administrator can use the Server Console or set the **nsslapd-errorlog-level** attribute. For more information, see *Red Hat Directory Server Configuration, Command, and File Reference*.

Syntax

```
#include "slapi-plugin.h"
int slapi_is_loglevel_set( const int loglevel );
```

Parameters

This function takes the following parameter:

<i>loglevel</i>	Log level setting to check.
-----------------	-----------------------------

Returns

The function returns one of the following values:

- 0 if *loglevel* is not selected as log level settings.
- 1 if *loglevel* is selected as log level setting.

CHAPTER 31. FUNCTIONS FOR COUNTERS

This chapter contains reference information on routines for configuring values for Directory Server database and server statistics counters.

Table 31.1. Counter Routines

Function	Description
<code>slapi_counter_add()</code>	Adds a certain amount to the counter value.
<code>slapi_counter_decrement()</code>	Decrements the counter and returns the new value.
<code>slapi_counter_destroy()</code>	Destroys an existing counter.
<code>slapi_counter_get_value()</code>	Gets the current value of the counter.
<code>slapi_counter_increment()</code>	Increments the counter value and returns the new value.
<code>slapi_counter_init()</code>	Initializes a new counter.
<code>slapi_counter_new()</code>	Creates a new counter.
<code>slapi_counter_set_value()</code>	Sets the counter to a new, specified value and returns the updated value.
<code>slapi_counter_subtract()</code>	Subtracts a certain amount from the counter value.

31.1. SLAPI_COUNTER_ADD()

Atomically adds a specified amount to the counter value.

By default, the counter increments up **1**; using `slapi_counter_add()` allows the counter to increment by some other specified unit.

Syntax

```
#include "slapi-plugin.h"
PRUint64 slapi_counter_add(Slapi_Counter *counter, PRUint64 addvalue);
```

Parameters

This function takes the following parameters:

<i>counter</i>	The counter to which to add the specified value.
----------------	--

<i>addvalue</i>	The amount to add to the current counter value.
-----------------	---

Returns

The function returns the value of the counter (the current count) after the counter has been incremented by the amount set in `slapi_counter_add()`.

See Also

- [slapi_counter_increment\(\)](#)
- [slapi_counter_set_value\(\)](#)
- [slapi_counter_subtract\(\)](#)

31.2. SLAPI_COUNTER_DECREMENT()

Atomically decrements the counter value and returns the new value.

Both this function and [slapi_counter_increment\(\)](#) set the behavior of the counter, moving up or down. Other functions, such as [slapi_counter_set_value\(\)](#), manipulate the actual value of the counter.

Syntax

```
#include "slapi-plugin.h"
PRUint64 slapi_counter_decrement(Slapi_Counter *counter);
```

Parameters

This function takes the following parameter:

<i>counter</i>	The counter to decrement.
----------------	---------------------------

Returns

The function returns the value of the counter (the current count) after the counter has been decremented.

See Also

- [slapi_counter_subtract\(\)](#)
- [slapi_counter_increment\(\)](#)

31.3. SLAPI_COUNTER_DESTROY()

Frees a `Slapi_Counter` structure from memory. Use this with a dynamically allocated `Slapi_Counter` structure that was obtained by calling [slapi_counter_new\(\)](#).

Before calling this function, make sure that the specified counter is no longer in use.

Do not call `slapi_counter_destroy()` to destroy a counter which was not dynamically allocated.

Syntax

```
#include "slapi-plugin.h"
void slapi_counter_destroy(Slapi_Counter **counter);
```

Parameters

This function takes the following parameter:

<i>counter</i>	The address of the counter being destroyed.
----------------	---

See Also

- [slapi_counter_new\(\)](#)

31.4. SLAPI_COUNTER_GET_VALUE()

Atomically gets the current value of the counter.

Syntax

```
#include "slapi-plugin.h"
PRUint64 slapi_counter_get_value(Slapi_Counter *counter);
```

Parameters

This function takes the following parameter:

<i>counter</i>	The name of the counter for which the value is checked.
----------------	---

Returns

The function returns the value of the counter (the current count).

31.5. SLAPI_COUNTER_INCREMENT()

Atomically increments the counter value up by one (1) and returns the new value.

Syntax

```
#include "slapi-plugin.h"
PRUint64 slapi_counter_increment(Slapi_Counter *counter);
```

Parameters

This function takes the following parameter:

<i>counter</i>	The counter to increment.
----------------	---------------------------

Returns

The function returns the value of the counter (the current count) after the counter has been incremented.

See Also

- [slapi_counter_add\(\)](#)
- [slapi_counter_decrement\(\)](#)

31.6. SLAPI_COUNTER_INIT()

Initializes a new [Slapi_Counter](#) structure, sets the initial value for the new counter to 0.

This function is useful when the [Slapi_Counter](#) structure is static, similar to:

```
static Slapi_Counter operation_counter;
...
slapi_counter_init(&operation_counter);
```

Syntax

```
#include "slapi-plugin.h"
void slapi_counter_init(Slapi_Counter *counter);
```

Parameters

This function takes the following parameter:

<i>counter</i>	The name of the counter being initialized.
----------------	--

See Also

- [slapi_counter_new\(\)](#)

31.7. SLAPI_COUNTER_NEW()

Allocates and initializes a new [Slapi_Counter](#) structure. The value of this counter is initially set to zero (0).

When you are finished with this counter, call [slapi_counter_destroy\(\)](#) to free the function.

Syntax

```
#include "slapi-plugin.h"
Slapi_Counter *slapi_counter_new();
```

See Also

- [slapi_counter_destroy\(\)](#)

31.8. SLAPI_COUNTER_SET_VALUE()

Atomically sets the current value of the counter structure.

By default, counters start at zero (0). This function can be called to set a counter to a specific value.

Syntax

```
#include "slapi-plugin.h"
PRUint64 slapi_counter_set_value(Slapi_Counter *counter, PRUint64
newvalue);
```

Parameters

This function takes the following parameters:

<i>counter</i>	The counter for which to set the value.
<i>newvalue</i>	The new value for the counter.

Returns

The function returns the value of the counter (the current count) after the counter has been set to the value specified in **slapi_counter_set_value()**.

See Also

- [slapi_counter_add\(\)](#)
- [slapi_counter_subtract\(\)](#)
- [slapi_counter_get_value\(\)](#)

31.9. SLAPI_COUNTER_SUBTRACT()

Subtracts the specified amount from the counter value.

Syntax

```
#include "slapi-plugin.h"
PRUint64 slapi_counter_subtract(Slapi_Counter *counter, PRUint64
subvalue);
```

Parameters

This function takes the following parameters:

<i>counter</i>	The counter for which to set the value.
----------------	---

<i>subvalue</i>	The amount which should be subtracted from the current counter value to give the new value.
-----------------	---

Returns

The function returns the value of the counter (the current count) after the amount specified in `slapi_counter_subtract()` has been subtracted.

See Also

- [slapi_counter_add\(\)](#)
- [slapi_counter_decrement\(\)](#)

CHAPTER 32. FUNCTIONS FOR HANDLING MATCHING RULES

This chapter contains reference information on matching rule routines.

Table 32.1. Matching Rule Routines

Function	Description
<code>slapi_berval_cmp()</code>	Compares two berval structures to determine if they are equal.
<code>slapi_matchingrule_free()</code>	Frees the specified matching rule structure (and, optionally, its members) from memory.
<code>slapi_matchingrule_get()</code>	Gets information about a matching rule.
<code>slapi_matchingrule_is_ordering()</code>	Determines if a matching rule is a valid ordering matching rule for the given syntax.
<code>slapi_matchingrule_new()</code>	Allocates memory for a new <code>Slapi_MatchingRuleEntry</code> structure.
<code>slapi_matchingrule_register()</code>	Registers the specified matching rule with the server.
<code>slapi_matchingrule_set()</code>	Sets information about the matching rule.
<code>slapi_matchingrule_unregister()</code>	Placeholder for future function. Currently, this function does nothing.
<code>slapi_mr_filter_index()</code>	Calls the indexer function associated with an extensible match filter.
<code>slapi_mr_indexer_create()</code>	Calls the indexer factory function for the plug-in responsible for a specified matching rule.

32.1. SLAPI_BERVAL_CMP()

Compares two berval structures to determine if they are equal.

Syntax

```
#include "slapi-plugin.h"
int slapi_berval_cmp (const struct berval *L, const struct berval *R);
```

Parameters

This function takes the following parameters:

<i>L</i>	Pointer to the first berval structure that you want to compare.
<i>R</i>	Pointer to the second structure that you want to compare.

Returns

This function returns one of the following values:

- A negative value if *L* is less than *R*.
- 0 if *L* is equal to *R*.
- A positive value if *L* is greater than *R*.

32.2. SLAPI_MATCHINGRULE_FREE()

This function frees a `Slapi_MatchingRuleEntry` structure (and, optionally, its members) from memory. Call this function when you are done working with the structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_matchingrule_free(Slapi_MatchingRuleEntry **mrEntry, int
freeMembers);
```

Parameters

This function takes the following parameters:

<i>mrEntry</i>	The <code>Slapi_MatchingRuleEntry</code> structure that you want to free from memory.
<i>freeMembers</i>	If 1 , the function also frees the members of the structure from memory.

32.3. SLAPI_MATCHINGRULE_GET()

Gets information about a matching rule.

This function gets information about a matching rule from the `Slapi_MatchingRuleEntry` structure. To set information in this structure, call the [slapi_matchingrule_set\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
int slapi_matchingrule_get(Slapi_MatchingRuleEntry *mr, int arg, void
*value);
```

Parameters

This function takes the following parameters:

<i>mr</i>	Slapi_MatchingRuleEntry structure from which you want to get data.
<i>arg</i>	ID specifying the type of information you want to get.
<i>value</i>	Pointer to a variable to hold the retrieved data.

The *arg* argument can have one of the following values:

ID	Data Type of the value Argument	Description
<i>SLAPI_MATCHINGRULE_NAME</i>	<i>char *</i>	Name of the matching rule.
<i>SLAPI_MATCHINGRULE_OID</i>	<i>char *</i>	OID of the matching rule.
<i>SLAPI_MATCHINGRULE_DESC</i>	<i>char *</i>	Description of the matching rule.
<i>SLAPI_MATCHINGRULE_SYNTAX</i>	<i>char *</i>	Syntax supported by the matching rule.
<i>SLAPI_MATCHINGRULE_OBSOLETE</i>	<i>int</i>	If 1 , the matching rule is obsolete.

Returns

This function returns one of the following values:

- 0 if the information was successfully retrieved.
- -1 if an error occurred; for example, if an invalid argument was specified.

32.4. SLAPI_MATCHINGRULE_IS_ORDERING()

Determines if the matching rule is a valid ordering matching rule for the given syntax.

Syntax

This function takes the following syntax:

```
int slapi_matchingrule_is_ordering(const char *oid_or_name, const char *syntax_oid)
```

slapi_matchingrule_is_ordering Parameters

This function has the following parameters:

Parameter	Description
<i>oid_or_name</i>	The OID or name of the matching rule being applied.
<i>syntax_oid</i>	The OID of the entry syntax. The function compares the type of matching rule to the allowed syntax.

Return Values

This function returns either of two values:

- **1** means that the matching rule is a valid ordering matching rule for the given syntax.
- **0** means that the matching rule could not be found, does not apply to the given syntax, or is not an ordering matching rule.

32.5. SLAPI_MATCHINGRULE_NEW()

Description

This function allocates memory for a new `Slapi_MatchingRuleEntry` structure, which represents a matching rule. After you call this function, you can call the [slapi_matchingrule_set\(\)](#) function to set the values in this structure and call the [slapi_matchingrule_register\(\)](#) function to register the matching rule.

Syntax

```
#include "slapi-plugin.h"
Slapi_MatchingRuleEntry *slapi_matchingrule_new();
```

Returns

This function returns one of the following values:

- A new `Slapi_MatchingRuleEntry` structure.
- **NULL** if memory could not be allocated.

32.6. SLAPI_MATCHINGRULE_REGISTER()

Description

This function registers the specified matching rule with the server. To create the matching rule and set its values, call the [slapi_matchingrule_new\(\)](#) function and the [slapi_matchingrule_set\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
int slapi_matchingrule_register(Slapi_MatchingRuleEntry *mrEntry);
```

Parameters

This function takes the following parameter:

<i>mrEntry</i>	Slapi_MatchingRuleEntry structure representing the matching rule that you want to register.
----------------	---

Returns

This function returns one of the following values:

- 0 if the matching rule was successfully registered.
- -1 if an error occurred.

32.7. SLAPI_MATCHINGRULE_SET()

Sets information about the matching rule.

Description

This function sets information in an Slapi_MatchingRuleEntry structure. To get information from this structure, call the [slapi_matchingrule_get\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
int slapi_matchingrule_set(Slapi_MatchingRuleEntry *mr, int arg, void
*value);
```

Parameters

This function takes the following parameters:

<i>mr</i>	Slapi_MatchingRuleEntry structure in which you want to set data.
<i>arg</i>	ID specifying the type of information to set.
<i>value</i>	The value that you want to set.

The *arg* argument can have one of the following values:

ID	Data Type of the value Argument	Description
<i>SLAPI_MATCHINGRULE_NAME</i>	<i>char *</i>	Name of the matching rule.
<i>SLAPI_MATCHINGRULE_OID</i>	<i>char *</i>	OID of the matching rule.

<code>SLAPI_MATCHINGRULE_DESC</code>	<code>char *</code>	Description of the matching rule.
<code>SLAPI_MATCHINGRULE_SYNTAX</code>	<code>char *</code>	Syntax supported by the matching rule.
<code>SLAPI_MATCHINGRULE_OBSOLETE</code>	<code>int</code>	If 1 , the matching rule is obsolete.

Returns

This function returns one of the following values:

- 0 if the information was successfully set.
- -1 if an error occurred; for example, if an invalid argument was specified.

32.8. SLAPI_MATCHINGRULE_UNREGISTER()

Placeholder for future function. Currently, this function does nothing.

Syntax

```
#include "slapi-plugin.h"
int slapi_matchingrule_unregister(char *oid);
```

32.9. SLAPI_MR_FILTER_INDEX()

Calls the indexer function associated with an extensible match filter.

If the filter specified by the *f* argument is an extensible match filter, this function calls the indexer function associated with the filter.

Before calling this function, make sure that the parameter block *pb* contains the information needed by the indexer function. You can pass information to the indexer function by using the following parameters:

- **SLAPI_PLUGIN_MR_VALUES** should contain a NULL-terminated list of values from the extensible match filter.
- **SLAPI_PLUGIN_OBJECT** should contain information that you want to pass to the indexer function.

The indexer function should set the **SLAPI_PLUGIN_MR_KEYS** parameter of the parameter block *pb* to an array of the keys that correspond to the values in the **SLAPI_PLUGIN_MR_VALUES** parameter.

For more information on filter index functions and indexer functions, see [Chapter 11, Writing Matching Rule Plug-ins](#).

Syntax

```
#include "slapi-plugin.h"
int slapi_mr_filter_index (Slapi_Filter *f, Slapi_PBlock *pb);
```

Parameters

This function takes the following parameters:

<i>f</i>	Pointer to a Section 14.23 , “Slapi_Filter” structure, representing the extensible match filter for which you want to find the indexer function.
<i>pb</i>	Parameter block containing information about the extensible match filter.

Returns

This function returns the result code returned by the indexer function.

32.10. SLAPI_MR_INDEXER_CREATE()

Calls the indexer factory function for the plug-in responsible for a specified matching rule.

Description

This function calls the indexer factory function for the plug-in responsible for handing a specified matching rule. The matching rule is identified by the OID in the **SLAPI_PLUGIN_MR_OID** parameter.

If no plug-ins are associated with this matching rule, the function calls the indexer factory function for each matching rule plug-in until the **SLAPI_PLUGIN_MR_INDEX_FN** parameter is set to an indexer function.

Before calling this function, make sure that the parameter block *pb* contains the information needed by the indexer factory function. You can pass information to the indexer factory function by using the following parameters:

- **SLAPI_PLUGIN_MR_OID** should contain the OID of the matching rule that you want used for indexing or sorting.
- **SLAPI_PLUGIN_MR_TYPE** should contain the attribute type that you want used for indexing or sorting.
- **SLAPI_PLUGIN_MR_USAGE** should specify if the indexer will be used for indexing (**SLAPI_PLUGIN_MR_USAGE_INDEX**) or for sorting (**SLAPI_PLUGIN_MR_USAGE_SORT**).

The indexer factory function should set the following parameters:

- **SLAPI_PLUGIN_MR_OID** should contain the official OID of the matching rule that you want used for indexing or sorting.
- **SLAPI_PLUGIN_MR_INDEX_FN**, if specified, contains an indexer pointer function. The function is responsible for indexing or sorting, based on the matching rule OID or attribute type. The values to index or sort are provided in the *slapi_pblock*

function using the **SLAPI_PLUGIN_MR_VALUES** parameter. The values are stored in a NULL-terminated **berval** array.

- **SLAPI_PLUGIN_MR_INDEX_SV_FN**, if specified, contains an indexer pointer function. The function is responsible for indexing or sorting, based on the matching rule OID or attribute type. The values to index or sort are provided in the **slapi_pblock_get()** function using the **SLAPI_PLUGIN_MR_VALUES** parameter. The values are stored in a NULL-terminated **Slapi_Value** array.
- **SLAPI_PLUGIN_OBJECT** should contain any information that you want passed to the indexer function.
- **SLAPI_PLUGIN_DESTROY_FN** should specify the name of the function responsible for freeing any memory allocated by this indexer factory function. For example, memory allocated for a structure that you pass to the indexer function using **SLAPI_PLUGIN_OBJECT**.

For more information on filter index functions and indexer functions, see [Chapter 11, Writing Matching Rule Plug-ins](#).

Syntax

```
#include "slapi-plugin.h"
int slapi_mr_indexer_create (Slapi_PBlock *opb);
```

Parameters

This function takes the following parameter:

<i>pb</i>	Parameter block containing information about the matching rule and attribute type to be used in indexing or sorting.
-----------	--

Returns

This function returns the result code returned by the indexer factory function.

CHAPTER 33. FUNCTIONS FOR LDAPMOD MANIPULATION

This chapter contains reference information on LDAPMod manipulation routines.

Table 33.1. LDAPMod Manipulation Routines

Function	Description
<code>slapi_entry2mods()</code>	Creates an array of LDAPMod from a Slapi_Entry.
<code>slapi_mod_add_value()</code>	Adds a value to a Slapi_Mod structure.
<code>slapi_mod_done()</code>	Frees internals of Slapi_Mod structure.
<code>slapi_mod_dump()</code>	Dumps the contents of an LDAPMod to the server log.
<code>slapi_mod_free()</code>	Frees a Slapi_Mod structure.
<code>slapi_mod_get_first_value()</code>	Initializes a Slapi_Mod iterator and returns the first attribute value.
<code>slapi_mod_get_ldapmod_byref()</code>	Gets a reference to the LDAPMod in a Slapi_Mod structure.
<code>slapi_mod_get_ldapmod_passout()</code>	Retrieves the LDAPMod contained in a Slapi_Mod structure.
<code>slapi_mod_get_next_value()</code>	Increments the Slapi_Mod iterator and returns the next attribute value.
<code>slapi_mod_get_num_values()</code>	Gets the number of values in a Slapi_Mod structure.
<code>slapi_mod_get_operation()</code>	Gets the operation type of Slapi_Mod structure.
<code>slapi_mod_get_type()</code>	Gets the attribute type of a Slapi_Mod structure.
<code>slapi_mod_init()</code>	Initializes a Slapi_Mod structure.
<code>slapi_mod_init_byref()</code>	Initializes a Slapi_Mod structure that is a wrapper for an existing LDAPMod.
<code>slapi_mod_init_byval()</code>	Initializes a Slapi_Mod structure with a copy of an LDAPMod.

Function	Description
<code>slapi_mod_init_passin()</code>	Initializes a <code>Slapi_Mod</code> from an <code>LDAPMod</code> .
<code>slapi_mod_init_valueset_byval()</code>	Initializes the given <i>smod</i> with the given LDAP operation and attribute type.
<code>slapi_mod_isvalid()</code>	Determines whether a <code>Slapi_Mod</code> structure is valid.
<code>slapi_mod_new()</code>	Allocates a new <code>Slapi_Mod</code> structure.
<code>slapi_mod_remove_value()</code>	Removes the value at the current <code>Slapi_Mod</code> iterator position.
<code>slapi_mod_set_operation()</code>	Sets the operation type of a <code>Slapi_Mod</code> structure.
<code>slapi_mod_set_type()</code>	Sets the attribute type of a <code>Slapi_Mod</code> .
<code>slapi_mods2entry()</code>	Creates a <code>Slapi_Entry</code> from an array of <code>LDAPMod</code> .
<code>slapi_mods_add()</code>	Appends a new mod with a single attribute value to <code>Slapi_Mods</code> structure.
<code>slapi_mods_add_ldapmod()</code>	Appends an <code>LDAPMod</code> to a <code>Slapi_Mods</code> structure.
<code>slapi_mods_add_mod_values()</code>	Appends a new mod to a <code>Slapi_Mods</code> structure with attribute values provided as an array of <code>Slapi_Value</code> .
<code>slapi_mods_add_modbvps()</code>	Appends a new mod to a <code>Slapi_Mods</code> structure, with attribute values provided as an array of <code>berval</code> .
<code>slapi_mods_add_smod()</code>	Appends a new <i>smod</i> to a <code>Slapi_Mods</code> structure.
<code>slapi_mods_add_string()</code>	Appends a new mod to <code>Slapi_Mods</code> structure with a single attribute value provided as a string.
<code>slapi_mods_done()</code>	Frees internals of a <code>Slapi_Mods</code> structure.
<code>slapi_mods_dump()</code>	Dumps the contents of a <code>Slapi_Mods</code> structure to the server log.

Function	Description
<code>slapi_mods_free()</code>	Frees a <code>Slapi_Mods</code> structure.
<code>slapi_mods_get_first_mod()</code>	Initializes a <code>Slapi_Mods</code> iterator and returns the first <code>LDAPMod</code> .
<code>slapi_mods_get_first_smod()</code>	Initializes a <code>Slapi_Mods</code> iterator and returns the first mod wrapped in a <code>Slapi_Mods</code> structure.
<code>slapi_mods_get_ldapmods_byref()</code>	Gets a reference to the array of <code>LDAPMod</code> in a <code>Slapi_Mods</code> structure.
<code>slapi_mods_get_ldapmods_passout()</code>	Retrieves the array of <code>LDAPMod</code> contained in a <code>Slapi_Mods</code> structure.
<code>slapi_mods_get_next_mod()</code>	Increments the <code>Slapi_Mods</code> iterator and returns the next <code>LDAPMod</code> .
<code>slapi_mods_get_next_smod()</code>	Increments the <code>Slapi_Mods</code> iterator and returns the next mod wrapped in a <code>Slapi_Mods</code> .
<code>slapi_mods_get_num_mods()</code>	Gets the number of mods in a <code>Slapi_Mods</code> structure.
Section 33.40, “ <code>slapi_mods_init()</code> ”	Initializes a <code>Slapi_Mods</code> .
<code>slapi_mods_init_byref()</code>	Initializes a <code>Slapi_Mods</code> that is a wrapper for an existing array of <code>LDAPMod</code> .
<code>slapi_mods_init_passin()</code>	Initializes a <code>Slapi_Mods</code> structure from an array of <code>LDAPMod</code> .
<code>slapi_mods_insert_after()</code>	Inserts an <code>LDAPMod</code> into a <code>Slapi_Mods</code> structure after the current iterator position.
<code>slapi_mods_insert_at()</code>	Inserts an <code>LDAPMod</code> anywhere in a <code>Slapi_Mods</code> .
<code>slapi_mods_insert_before()</code>	Inserts an <code>LDAPMod</code> into a <code>Slapi_Mods</code> structure before the current iterator position.
<code>slapi_mods_insert_smod_at()</code>	Inserts an <i>smod</i> anywhere in a <code>Slapi_Mods</code> structure.
<code>slapi_mods_insert_smod_before()</code>	Inserts an <i>smod</i> before a <code>Slapi_Mods</code> structure.
<code>slapi_mods_iterator_backone()</code>	Decrements the <code>Slapi_Mods</code> current iterator position.

Function	Description
slapi_mods_new()	Allocates a new uninitialized Slapi_Mods structure.
slapi_mods_remove()	Removes the mod at the current Slapi_Mods iterator position.

33.1. SLAPI_ENTRY2MODS()

Description

This function creates an array of LDAPMod of type **LDAP_MOD_ADD** from a Slapi_Entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry2mods(const Slapi_Entry *e, char **dn, LDAPMod ***attrs);
```

Parameters

This function takes the following parameters:

<i>e</i>	Pointer to a Slapi_Entry.
<i>dn</i>	Address of a char* that will be set on return to the entry DN.
<i>attrs</i>	Address of an array of LDAPMod that will be set on return to a copy of the entry attributes.

Returns

This function returns one of the following values:

- 0 if successful.
- A non-zero value if not successful.

See Also

[slapi_mods2entry\(\)](#)

33.2. SLAPI_MOD_ADD_VALUE()

Description

Adds a copy of a given attribute to the Slapi_Mod structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_add_value(Slapi_Mod *smod, const struct berval *val);
```

■

Parameters

This function takes the following parameters:

<i>smod</i>	Pointer to an initialized Slapi_Mod.
<i>val</i>	Pointer to a berval representing the attribute value.

33.3. SLAPI_MOD_DONE()

Description

This function frees the internals of a Slapi_Mod, leaving it in the uninitialized state.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_done(Slapi_Mod *mod);
```

Parameters

This function takes the following parameter:

<i>mod</i>	Pointer to a Slapi_Mod.
------------	-------------------------

Memory Concerns

Use this function on a stack-allocated Slapi_Mod when you have finished with it or want to reuse it.

See Also

- [slapi_mod_init\(\)](#)
- [slapi_mod_init_byval\(\)](#)
- [slapi_mod_init_byref\(\)](#)
- [slapi_mod_init_passin\(\)](#)

33.4. SLAPI_MOD_DUMP()

Description

This function uses the **LDAP_DEBUG_ANY** log level to dump the contents of anLDAPMod to the server log for debugging.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_dump(LDAPMod *mod, int n);
```


Parameters

This function takes the following parameters:

<i>mod</i>	Pointer to an LDAPMod.
<i>n</i>	Numeric label that will be included in the log.

33.5. SLAPI_MOD_FREE()

Description

This function frees a Slapi_Mod structure that was allocated by [slapi_mod_new\(\)](#).

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_free(Slapi_Mod **smod);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to an initialized Slapi_Mod.
-------------	--------------------------------------

See Also

[slapi_mod_new\(\)](#)

33.6. SLAPI_MOD_GET_FIRST_VALUE()

Description

Use this function with [slapi_mod_get_next_value\(\)](#) to iterate through the attribute values in a Slapi_Mod structure. The function initializes a Slapi_Mod iterator and returns the first attribute value.

Syntax

```
#include "slapi-plugin.h"
struct berval *slapi_mod_get_first_value(Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to an initialized Slapi_Mod.
-------------	--------------------------------------

Returns

This function returns a pointer to the first attribute value in the Slapi_Mod or **NULL** if no values exist.

See Also

[slapi_mod_get_next_value\(\)](#)

33.7. SLAPI_MOD_GET_LDAPMOD_BYREF()

Description

This function gets a reference to the LDAPMod in a Slapi_Mod structure. Use this function to get direct access to the LDAPMod contained in a Slapi_Mod.

Syntax

```
#include "slapi-plugin.h"
const LDAPMod *slapi_mod_get_ldapmod_byref(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to an initialized Slapi_Mod.
-------------	--------------------------------------

Returns

This function returns a pointer to a read-only LDAPMod owned by the Slapi_Mod.

Memory Concerns

Responsibility for the LDAPMod remains with the Slapi_Mod.

See Also

[slapi_mod_get_ldapmod_passout\(\)](#)

33.8. SLAPI_MOD_GET_LDAPMOD_PASSOUT()

Description

Use this function to get the LDAPMod out of a Slapi_Mod.

Syntax

```
#include "slapi-plugin.h"
LDAPMod *slapi_mod_get_ldapmod_passout(Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to an initialized Slapi_Mod.
-------------	--------------------------------------

Returns

This function returns a pointer to an LDAPMod owned by the caller.

Memory Concerns

Responsibility for the LDAPMod transfers to the caller. The `Slapi_Mod` is left in the uninitialized state.

See Also

[slapi_mod_get_ldapmod_byref\(\)](#)

33.9. SLAPI_MOD_GET_NEXT_VALUE()

Description

This function increments the `Slapi_Mod` iterator and return the next attribute value. Use this function with `slapi_mods_get_first_mod()` to iterate through the attribute values in a `Slapi_Mod`.

Syntax

```
#include "slapi-plugin.h"
struct berval *slapi_mod_get_next_value(Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to an initialized <code>Slapi_Mod</code> .
-------------	--

Returns

This function returns a pointer to the next attribute value in the `Slapi_Mod` or **NULL** if there are no more.

See Also

[slapi_mods_get_first_mod\(\)](#)

33.10. SLAPI_MOD_GET_NUM_VALUES()

Gets the number of values in a `Slapi_Mod` structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_mod_get_num_values(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to an initialized <code>Slapi_Mod</code> .
-------------	--

Returns

This function returns the number of attribute values in the `Slapi_Mod`.

33.11. SLAPI_MOD_GET_OPERATION()

Gets the operation type of Slapi_Mod structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_mod_get_operation(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to an initialized Slapi_Mod.
-------------	--------------------------------------

Returns

This function returns one of **LDAP_MOD_ADD**, **LDAP_MOD_DELETE**, or **LDAP_MOD_REPLACE**, combined using the bitwise or operator with **LDAP_MOD_BYVALUES**.

See Also

[slapi_mod_set_operation\(\)](#)

33.12. SLAPI_MOD_GET_TYPE()

Description

Gets the LDAP attribute type of a Slapi_Mod.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_mod_get_type(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to an initialized Slapi_Mod.
-------------	--------------------------------------

Returns

This function returns a read-only pointer to the attribute type in the Slapi_Mod.

See Also

[slapi_mod_set_type\(\)](#)

33.13. SLAPI_MOD_INIT()

Description

This function initializes a Slapi_Mod so that it is empty but initially has room for the given number of attribute values.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init(Slapi_Mod *smod, int initCount);
```

Parameters

This function takes the following parameters:

<i>smod</i>	Pointer to an uninitialized Slapi_Mod.
<i>initCount</i>	Suggested number of attribute values for which to make room. Minimum value is 0 .

Memory Concerns

If you are unsure of the room you will need, you may use an *initCount* of **0**. The Slapi_Mod expands as necessary.

See Also

- [slapi_mod_done\(\)](#)
- [slapi_mod_init_byval\(\)](#)
- [slapi_mod_init_byref\(\)](#)
- [slapi_mod_init_passin\(\)](#)

33.14. SLAPI_MOD_INIT_BYREF()

Description

This function initializes a Slapi_Mod containing a reference to an LDAPMod. Use this function when you have an LDAPMod and would like the convenience of the Slapi_Mod functions to access it.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init_byref(Slapi_Mod *smod, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

<i>smod</i>	Pointer to an uninitialized Slapi_Mod.
<i>mod</i>	Pointer to an LDAPMod.

See Also

- [slapi_mod_done\(\)](#)

- [slapi_mod_init\(\)](#)
- [slapi_mod_init_byval\(\)](#)
- [slapi_mod_init_passin\(\)](#)

33.15. SLAPI_MOD_INIT_BYVAL()

Initializes a Slapi_Mod structure with a copy of anLDAPMod.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init_byval(Slapi_Mod *smod, const LDAPMod *mod);
```

Parameters

This function takes the following parameters:

<i>smod</i>	Pointer to an uninitialized Slapi_Mod.
<i>mod</i>	Pointer to an LDAPMod.

See Also

- [slapi_mod_done\(\)](#)
- [slapi_mod_init\(\)](#)
- [slapi_mod_init_byref\(\)](#)
- [slapi_mod_init_byval\(\)](#)

33.16. SLAPI_MOD_INIT_PASSIN()

Description

This function initializes a Slapi_Mod by passing in anLDAPMod. Use this function to convert an LDAPMod to a Slapi_Mod.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init_passin(Slapi_Mod *smod, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

<i>smod</i>	Pointer to an uninitialized Slapi_Mod.
<i>mod</i>	Pointer to an LDAPMod.

Memory Concerns

Responsibility for the LDAPMod is transferred to the Slapi_Mod. The LDAPMod is destroyed when the Slapi_Mod is destroyed.

See Also

- [slapi_mod_done\(\)](#)
- [slapi_mod_init\(\)](#)
- [slapi_mod_init_byref\(\)](#)
- [slapi_mod_init_byval\(\)](#)

33.17. SLAPI_MOD_INIT_VALUESET_BYVAL()

Initializes the given *smod* in a **Slapi_Mod** with the given LDAP operation and attribute type. The attribute values in the *smod* are initialized using **Slapi_ValueSet**. All function parameters are copied, and *type* and *svs* are not modified.

This function checks that an attribute being modified or added does not exist, and, if it does exist, it removes that value from the list of modifications.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init_valueset_byval(Slapi_Mod *smod, int op, const char
*type, const Slapi_ValueSet *svs)
```

Parameters

This function takes the following parameters:

<i>smod</i>	Pointer to an uninitialized Slapi_Mod.
<i>op</i>	One of LDAP_MOD_ADD , LDAP_MOD_DELETE , or LDAP_MOD_REPLACE , combined using the bitwise or operator with LDAP_MOD_BYVALUES .
<i>type</i>	An attribute type.
<i>svs</i>	Pointer to the uninitialized Slapi_ValueSet structure of which to get the count.

Memory Concerns

Use [slapi_mods_free\(\)](#) or [slapi_mod_done\(\)](#) to clean up the memory allocated by **slapi_mod_init_valueset_byval**

See Also

- [slapi_mod_done\(\)](#)
- [slapi_mods_free\(\)](#)

- [Slapi_ValueSet](#)
- [Slapi_Mod](#)

33.18. SLAPI_MOD_ISVALID()

Determines whether a Slapi_Mod structure is valid.

Description

Use this function to verify that the contents of Slapi_Mod are valid. It is considered valid if the operation type is one of **LDAP_MOD_ADD**, **LDAP_MOD_DELETE**, or **LDAP_MOD_REPLACE**, combined using the bitwise or operator with **LDAP_MOD_BYVALUES**; the attribute type is not **NULL**; and there is at least one attribute value for add and replace operations.

Syntax

```
#include "slapi-plugin.h"
int slapi_mod_isvalid(const Slapi_Mod *mod);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to a Slapi_Mod.
-------------	-------------------------

Returns

This function returns one of the following values:

- 1 if the Slapi_Mod is valid.
- 0 if the Slapi_Mod is not valid.

33.19. SLAPI_MOD_NEW()

Description

This function allocates a new uninitialized Slapi_Mod. Use this function when you need to a Slapi_Mod allocated from the heap, rather than from the stack.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mod* slapi_mod_new( void );
```

Parameters

This function takes no parameters.

Returns

This function returns a pointer to an allocated, uninitialized Slapi_Mod.

See Also

[slapi_mod_free\(\)](#)

33.20. SLAPI_MOD_REMOVE_VALUE()

Removes the value at the current Slapi_Mod iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_remove_value(Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to an initialized Slapi_Mod.
-------------	--------------------------------------

See Also

- [slapi_mod_get_first_value\(\)](#)
- [slapi_mod_get_next_value\(\)](#)

33.21. SLAPI_MOD_SET_OPERATION()

Sets the operation type of a Slapi_Mod structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_set_operation(Slapi_Mod *smod, int op);
```

Parameters

This function takes the following parameters:

<i>smod</i>	Pointer to an initialized Slapi_Mod.
<i>op</i>	One of LDAP_MOD_ADD , LDAP_MOD_DELETE , or LDAP_MOD_REPLACE , combined using the bitwise or operator with LDAP_MOD_BYVALUES .

See Also

[slapi_mod_get_operation\(\)](#)

33.22. SLAPI_MOD_SET_TYPE()

Sets the attribute type of a Slapi_Mod.

Description

Sets the attribute type of the Slapi_Mod to a copy of the given value.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_set_type(Slapi_Mod *smod, const char *type);
```

Parameters

This function takes the following parameters:

<i>smod</i>	Pointer to an initialized Slapi_Mod.
<i>type</i>	An attribute type.

See Also

[slapi_mod_get_type\(\)](#)

33.23. SLAPI_MODS2ENTRY()

Description

This function creates a Slapi_Entry from a copy of an array ofLDAPMod of type **LDAP_MODD_ADD**.

Syntax

```
#include "slapi-plugin.h"
int slapi_mods2entry(Slapi_Entry **e, const Slapi_DN *dn, LDAPMod
**attrs);
```

Parameters

This function takes the following parameters:

<i>e</i>	Address of a pointer that will be set on return to the created entry.
<i>dn</i>	The LDAP DN of the entry.
<i>attrs</i>	An array of LDAPMod of type LDAP_MOD_ADD representing the entry attributes.

Returns

This function returns one of the following values:

- **LDAP_SUCCESS** if successful.
- An LDAP return code if not successful.

See Also

[slapi_entry2mods\(\)](#)

33.24. SLAPI_MODS_ADD()

Description

This function appends a new **mod** with a single attribute value to a **Slapi_Mods**. The **mod** is constructed from copies of the values of **modtype**, **type**, **len**, and **val**.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add( Slapi_Mods *smods, int modtype, const char *type,
    unsigned long len, const char *val);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods .
<i>modtype</i>	One of LDAP_MOD_ADD , LDAP_MOD_DELETE , or LDAP_MOD_REPLACE .
<i>type</i>	The LDAP attribute type.
<i>len</i>	The length in bytes of the attribute value.
<i>val</i>	The attribute value.

Memory Concerns

This function must not be used on **Slapi_Mods** initialized with **slapi_mods_init_byref()**.

See Also

- [slapi_mods_add_ldapmod\(\)](#)
- [slapi_mods_add_modbvps\(\)](#)
- [slapi_mods_add_mod_values\(\)](#)
- [slapi_mods_add_string\(\)](#)

33.25. SLAPI_MODS_ADD_LDAPMOD()

Description

Appends an **LDAPMod** to a **Slapi_Mods**.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_ldapmod(Slapi_Mods *smods, LDAPMod *mod);
```

■

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
<i>mod</i>	Pointer to a the LDAPMod to be appended.

Memory Concerns

Responsibility for the LDAPMod is transferred to the Slapi_Mods. This function must not be used on a Slapi_Mods initialized with `slapi_mods_init_byref()`.

See Also

- [slapi_mods_add\(\)](#)
- [slapi_mods_add_modbvps\(\)](#)
- [slapi_mods_add_mod_values\(\)](#)
- [slapi_mods_add_string\(\)](#)

33.26. SLAPI_MODS_ADD_MOD_VALUES()

Description

This function appends a new **mod** to a Slapi_Mods. The **mod** is constructed from copies of the values of **modtype**, **type** and **va**. Use this function when you have the attribute values to hand as an array of Slapi_Value.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_mod_values( Slapi_Mods *smods, int modtype, const char
*type, Slapi_Value **va );;
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
<i>modtype</i>	One of LDAP_MOD_ADD , LDAP_MOD_DELETE , or LDAP_MOD_REPLACE .
<i>type</i>	The LDAP attribute type.
<i>va</i>	A null-terminated array of Slapi_Value representing the attribute values.

Memory Concerns

This function must not be used on a Slapi_Mods initialized with [slapi_mods_add_mod_values\(\)](#).

See Also

- [slapi_mods_add\(\)](#)
- [slapi_mods_add_smod\(\)](#)
- [slapi_mods_add_ldapmod\(\)](#)
- [slapi_mods_add_modbvps\(\)](#)
- [slapi_mods_add_string\(\)](#)

33.27. SLAPI_MODS_ADD_SMOD()

Appends a new *smod* to a Slapi_Mods structure. The **mod** passed in is not copied or duplicated, but the reference is used directly.

Description

This function appends a new *smod* to a Slapi_Mods. The function [slapi_mods_get_num_mods\(\)](#) gives the number of **mods** in the Slapi_Mods structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_smod( Slapi_Mods *smods, Slapi_Mod *smod );
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
<i>smod</i>	Pointer to an initialized Slapi_Mod.

Memory Concerns

This function must not be used on a Slapi_Mods initialized with [slapi_mods_init_byref\(\)](#).

See Also

- [slapi_mods_insert_at\(\)](#)
- [slapi_mods_add\(\)](#)
- [slapi_mods_add_mod_values\(\)](#)
- [slapi_mods_add_ldapmod\(\)](#)
- [slapi_mods_add_modbvps\(\)](#)
- [slapi_mods_add_string\(\)](#)

33.28. SLAPI_MODS_ADD_MODBVPS()

Description

This function appends a new **mod** to Slapi_Mods. The **mod** is constructed from copies of the values of **modtype**, **type**, and *bvps*. Use this function when you have the attribute values to hand as an array of berval.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_modbvps( Slapi_Mods *smods, int modtype, const char
*type, struct berval **bvps );
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
<i>modtype</i>	One of LDAP_MOD_ADD , LDAP_MOD_DELETE , or LDAP_MOD_REPLACE .
<i>type</i>	The LDAP attribute type.
<i>bvps</i>	A null-terminated array of berval representing the attribute values.

Memory Concerns

This function must not be used on a Slapi_Mods initialized with [slapi_mods_init_byref\(\)](#).

See Also

- [slapi_mods_add\(\)](#)
- [slapi_mods_add_ldapmod\(\)](#)
- [slapi_mods_add_mod_values\(\)](#)
- [slapi_mods_add_string\(\)](#)

33.29. SLAPI_MODS_ADD_STRING()

Appends a new **mod** to Slapi_Mods structure with a single attribute value provided as a string.

Description

This function appends a new **mod** with a single string attribute value to aSlapi_Mods. The **mod** is constructed from copies of the values of *modtype*, *type*, and *val*.

Syntax

```
#include "slapi-plugin.h"
```

```
void slapi_mods_add_string( Slapi_Mods *smods, int modtype, const char
*type, const char *val);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
<i>modtype</i>	One of LDAP_MOD_ADD , LDAP_MOD_DELETE , or LDAP_MOD_REPLACE .
<i>type</i>	The LDAP attribute type.
<i>val</i>	The attribute value represented as a null-terminated string.

Memory Concerns

This function must not be used on a Slapi_Mods initialized with [slapi_mods_init_byref\(\)](#).

See Also

- [slapi_mods_add\(\)](#)
- [slapi_mods_add_ldapmod\(\)](#)
- [slapi_mods_add_modbvps\(\)](#)
- [slapi_mods_add_mod_values\(\)](#)

33.30. SLAPI_MODS_DONE()

Description

This function frees the internals of a Slapi_Mods, leaving it in the uninitialized state. Use this function on a stack-allocated Slapi_Mods when you are finished with it, or when you wish to reuse it.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_done(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to a Slapi_Mods.
-------------	--------------------------

See Also

- [slapi_mods_init\(\)](#)

- [slapi_mods_init_byref\(\)](#)
- [slapi_mods_init_passin\(\)](#)

33.31. SLAPI_MODS_DUMP()

Description

This function uses the **LDAP_DEBUG_ANY** log level to dump the contents of a `Slapi_Mods` to the server log for debugging.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_dump(const Slapi_Mods *smods, const char *text);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to a <code>Slapi_Mods</code> .
<i>text</i>	Descriptive text that will be included in the log, preceding the <code>Slapi_Mods</code> content.

33.32. SLAPI_MODS_FREE()

Frees a `Slapi_Mods` structure.

Description

This function frees a `Slapi_Mods` that was allocated by `slapi_mods_new()`.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_free(Slapi_Mods **smods);
```

Parameters

This function takes the following parameter:

<i>mods</i>	Pointer to an allocated <code>Slapi_Mods</code> .
-------------	---

See Also

[slapi_mods_new\(\)](#)

33.33. SLAPI_MODS_GET_FIRST_MOD()

Initializes a `Slapi_Mods` iterator and returns the first `LDAPMod`.

Syntax


```
#include "slapi-plugin.h"
LDAPMod *slapi_mods_get_first_mod(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
-------------	---------------------------------------

Returns

This function returns one of the following values:

- A pointer to the first LDAPMod in the Slapi_Mods.
- **NULL** if there are no mods.

33.34. SLAPI_MODS_GET_FIRST_SMOD()

Description

This function initializes a Slapi_Mods iterator and returns the first **mod** wrapped in a Slapi_Mods structure. Use this function in conjunction with **slapi_mods_get_next_smod()** to iterate through the mods in a Slapi_Mods using a Slapi_Mods wrapper.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mod *slapi_mods_get_first_smod(Slapi_Mods *smods, Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

<i>mods</i>	A pointer to a an initialized Slapi_Mods.
<i>smod</i>	Pointer to a Slapi_Mods that will be used to hold the mod .

Returns

This function returns one of the following values:

- A pointer to the Slapi_Mod, wrapping the first **mod**.
- **NULL** if no **mod** exist.

Memory Concerns

Only one thread may be iterating through a particular Slapi_Mods at any given time.

See Also

[slapi_mods_get_first_mod\(\)](#)

33.35. SLAPI_MODS_GET_LDAPMODS_BYREF()

Description

Use this function to get a reference and direct access to the array of LDAPMod contained in a Slapi_Mods.

Syntax

```
#include "slapi-plugin.h"
LDAPMod **slapi_mods_get_ldapmods_byref(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
-------------	---------------------------------------

Returns

This function returns a null-terminated array of LDAPMod owned by the Slapi_Mods.

Memory Concerns

Responsibility for the array remains with the Slapi_Mods.

See Also

[slapi_mods_get_ldapmods_passout\(\)](#)

33.36. SLAPI_MODS_GET_LDAPMODS_PASSOUT()

Description

Gets the array of LDAPMod out of a Slapi_Mods. Responsibility for the array transfers to the caller. The Slapi_Mods is left in the uninitialized state.

Syntax

```
#include "slapi-plugin.h"
LDAPMod **slapi_mods_get_ldapmods_passout(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

<i>smod</i>	Pointer to an initialized Slapi_Mods.
-------------	---------------------------------------

Returns

This function returns a null-terminated array LDAPMod owned by the caller.

See Also

[slapi_mods_get_ldapmods_byref\(\)](#)

33.37. SLAPI_MODS_GET_NEXT_MOD()

Description

This function increments the `Slapi_Mods` iterator and returns the next `LDAPMod`. Use this function in conjunction with [slapi_mods_get_first_mod\(\)](#) to iterate through the mods in a `Slapi_Mods`. This will return an `LDAPMod` each time until the end.

Syntax

```
#include "slapi-plugin.h"
LDAPMod *slapi_mods_get_next_mod(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

<i>smod</i>	A pointer to an initialized <code>Slapi_Mods</code> .
-------------	---

Returns

This function returns either a pointer to the next `LDAPMod` or **NULL** if there are no more.

Memory Concerns

Only one thread may be iterating through a particular `Slapi_Mods` at any given time.

See Also

[slapi_mods_get_first_mod\(\)](#)

33.38. SLAPI_MODS_GET_NEXT_SMOD()

Description

This function increments the `Slapi_Mods` iterator and returns the next **mod** wrapped in a `Slapi_Mod`. Use this function in conjunction with [slapi_mods_get_first_smod\(\)](#) to iterate through the mods in a `Slapi_Mods` using a `Slapi_Mods` wrapper.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mod *slapi_mods_get_next_smod(Slapi_Mods *smods, Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized <code>Slapi_Mods</code> .
<i>smod</i>	Pointer to a <code>Slapi_Mod</code> that will be used to hold the mod .

Returns

This function returns a pointer to the Slapi_Mod, wrapping the next**mod**, or **NULL** if there are no more mods.

Memory Concerns

Only one thread may be iterating through a particular Slapi_Mods at any given time.

See Also

[slapi_mods_get_first_smod\(\)](#)

33.39. SLAPI_MODS_GET_NUM_MODS()

Gets the number of **mods** in a Slapi_Mods structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_mods_get_num_mods(const Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
-------------	---------------------------------------

Returns

This function returns the number of **mods** in Slapi_Mods.

33.40. SLAPI_MODS_INIT()

Description

Initializes a Slapi_Mods so that it is empty but initially has room for the given number of **mods**.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_init(Slapi_Mods *smods, int initCount);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
<i>initCount</i> parameter	Suggested number of mods for which to make room. The minimum value is 0 .

Memory Concerns

If you are unsure of how much room you will need, you may use an *initCount* of 0. The `Slapi_Mods` expands as necessary.

See Also

[slapi_mods_done\(\)](#)

33.41. SLAPI_MODS_INIT_BYREF()

Description

Initializes a `Slapi_Mods` containing a reference to an array of `LDAPMod`. This function provides the convenience of using `Slapi_Mods` functions to access `LDAPMod` array items.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_init_byref(Slapi_Mods *smods, LDAPMod **mods);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an uninitialized <code>Slapi_Mods</code> .
<i>mods</i>	A null-terminated array of <code>LDAPMod</code> .

Memory Concerns

The array is not destroyed when the `Slapi_Mods` is destroyed. You cannot insert new `mods` in a `Slapi_Mods` that has been initialized by reference.

See Also

[slapi_mods_done\(\)](#)

33.42. SLAPI_MODS_INIT_PASSIN()

Description

This function initializes a `Slapi_Mods` by passing in an array of `LDAPMod`. This function converts an array of `LDAPMod` to a `Slapi_Mods`.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_init_passin(Slapi_Mods *smods, LDAPMod **mods);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an uninitialized <code>Slapi_Mods</code> .
-------------	---

<i>mods</i>	A null-terminated array of LDAPMod.
-------------	-------------------------------------

Memory Concerns

The responsibility for the array and its elements is transferred to the Slapi_Mods. The array and its elements are destroyed when the Slapi_Mods is destroyed.

See Also

[slapi_mods_done\(\)](#)

33.43. SLAPI_MODS_INSERT_AFTER()

Description

This function inserts an LDAPMod in a Slapi_Mods immediately after the current position of the Slapi_Mods iterator. The iterator position is unchanged.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_after(Slapi_Mods *smods, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods with a valid iterator position.
<i>mod</i>	Pointer to the LDAPMod to be inserted.

Memory Concerns

Responsibility for the LDAPMod is transferred to the Slapi_Mods. This function must not be used on a Slapi_Mods initialized with **slapi_mods_init_byref()**.

See Also

- [slapi_mods_get_first_mod\(\)](#)
- [slapi_mods_get_next_mod\(\)](#)
- [slapi_mods_get_first_smod\(\)](#)
- [slapi_mods_get_next_smod\(\)](#)

33.44. SLAPI_MODS_INSERT_AT()

Description

This function inserts an LDAPMod at a given position *pos* in Slapi_Mods. Position **0** (zero) refers to the first *mod*. A position equal to the current number of *mods* (determined by **slapi_mods_get_num_mods()**) causes an append *mods* at and above the specified position

are moved up by one, and the given position refers to the newly inserted *mod*. Shift everything down to make room to insert the new *mod*.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_at(Slapi_Mods *smods, LDAPMod *mod, int pos);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
<i>mod</i>	Pointer to the LDAPMod to be inserted.
<i>pos</i>	Position at which to insert the new mod . Minimum value is 0 . Maximum value is the current number of mods.

Memory Concerns

Responsibility for the LDAPMod is transferred to the Slapi_Mods. This function must not be used on a Slapi_Mods initialized with [slapi_mods_init_byref\(\)](#).

See Also

[slapi_mods_insert_at\(\)](#)

[slapi_mods_insert_at\(\)](#) adds to the end of all **mods**.

33.45. SLAPI_MODS_INSERT_BEFORE()

Description

Inserts an LDAPMod into a Slapi_Mods immediately before the current position of the Slapi_Mods iterator. The iterator position is unchanged.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_before(Slapi_Mods *smods, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods with valid iterator position.
<i>mod</i>	Pointer to the LDAPMod to be inserted.

Memory Concerns

The responsibility for the LDAPMod is transferred to the Slapi_Mods. This function must not be used on a Slapi_Mods initialized with `slapi_mods_init_byref()`.

See Also

- [slapi_mods_get_first_mod\(\)](#)
- [slapi_mods_get_next_mod\(\)](#)

33.46. SLAPI_MODS_INSERT_SMOD_AT()

Description

This function inserts an *smod* at a given position *pos* in Slapi_Mods. Position **0** (zero) refers to the first *smod*. A position equal to the current number of *smods* (determined by `slapi_mods_get_num_mods()`) causes an *appendsmod* at and above the specified position are moved up by one, and the given position refers to the newly inserted *smod*. Shift everything down to make room to insert the new *mod*.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_smod_at(Slapi_Mods *smods, Slapi_Mod *smod, int
pos);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
<i>smod</i>	Pointer to the LDAPMod to be inserted.
<i>pos</i>	Position at which to insert the new <i>mod</i> . Minimum value is 0 . Maximum value is the current number of mods.

Memory Concerns

Responsibility for the *smod* is transferred to the Slapi_Mods.

See Also

- [slapi_mods_insert_at\(\)](#)
- [slapi_mods_add_ldapmod\(\)](#) adds to the end of all *mods*

33.47. SLAPI_MODS_INSERT_SMOD_BEFORE()

Description

This function inserts an *smod* immediately before the current position of the Slapi_Mods iterator. The iterator position is unchanged.

Syntax


```
#include "slapi-plugin.h"
void slapi_mods_insert_smod_before(Slapi_Mods *smods, Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
<i>smod</i>	Pointer to the Slapi_Mod to be inserted.

Memory Concerns

The Slapi_Mod argument *smod* is not duplicated or copied, but the reference of the Slapi_Mod(**smods**) is passed into the Slapi_Mods(**smods**) structure. The responsibility for the *smod* is transferred to the Slapi_Mods.

See Also

- [slapi_mods_insert_before\(\)](#)
- [slapi_mods_insert_smod_at\(\)](#)

33.48. SLAPI_MODS_ITERATOR_BACKONE()

Description

Decrements the Slapi_Mods current iterator position. This function moves the iterator back one position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_iterator_backone(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
-------------	---------------------------------------

See Also

- [slapi_mods_get_first_mod\(\)](#)
- [slapi_mods_get_next_mod\(\)](#)
- [slapi_mods_get_first_smod\(\)](#)
- [slapi_mods_get_next_smod\(\)](#)

33.49. SLAPI_MODS_NEW()

This function allocates a new initialized Slapi_Mods.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mods* slapi_mods_new( void );
```

Parameters

This function takes no parameters.

Returns

This function returns a pointer to an allocated uninitialized Slapi_Mods.

Memory Concerns

Use this function when you need a Slapi_Mods allocated from the heap rather than from the stack.

See Also

[slapi_mods_free\(\)](#)

33.50. SLAPI_MODS_REMOVE()

Description

This function removes the **mod** at the current Slapi_Mods iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_remove(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

<i>mods</i>	Pointer to an initialized Slapi_Mods.
-------------	---------------------------------------

See Also

- [slapi_mods_get_first_mod\(\)](#)
- [slapi_mods_get_next_mod\(\)](#)
- [slapi_mods_get_first_smod\(\)](#)
- [slapi_mods_get_next_smod\(\)](#)

CHAPTER 34. FUNCTIONS FOR MONITORING OPERATIONS

This chapter contains reference information on operation routines.

Table 34.1. Operation Routines

Function	Description
slapi_op_abandoned()	Determines if the client has abandoned the current operation.
slapi_op_get_type()	Gets the type of a Slapi_Operation.

34.1. SLAPI_OP_ABANDONED()

Description

This function allows you to verify if the operation associated to the pblock in the parameter has been abandoned. This function is useful to check periodically the operations status of long-running plug-ins.

Syntax

```
#include "slapi-plugin.h"
int slapi_op_abandoned( Slapi_PBlock *pb );
```

Parameters

This function takes the following parameter:

<i>pb</i>	Parameter block passed in from the current operation.
-----------	---

Returns

This function returns one of the following values:

- 1 if the operation has been abandoned.
- 0 if the operation has not been abandoned.

34.2. SLAPI_OP_GET_TYPE()

Description

This function returns the type of an Slapi_Operation. The Slapi_Operation structure can be extracted from a pblock structure using [slapi_pblock_get\(\)](#) with the **SLAPI_OPERATION** parameter. For example:

```
slapi_pblock_get (pb, SLAPI_OPERATION, &op);
```

Syntax

```
#include "slapi-plugin.h"
unsigned long slapi_op_get_type(Slapi_Operation * op);
```

Parameters

This function takes the following parameter:

<i>op</i>	The operation of which you wish to get the type.
-----------	--

Returns

This function returns one of the following operation types:

- **SLAPI_OPERATION_BIND**
- **SLAPI_OPERATION_UNBIND**
- **SLAPI_OPERATION_SEARCH**
- **SLAPI_OPERATION_MODIFY**
- **SLAPI_OPERATION_ADD**
- **SLAPI_OPERATION_DELETE**
- **SLAPI_OPERATION_MODDN**
- **SLAPI_OPERATION_MODRDN**
- **SLAPI_OPERATION_COMPARE**
- **SLAPI_OPERATION_ABANDON**
- **SLAPI_OPERATION_EXTENDED**

See Also

[slapi_pblock_get\(\)](#)

CHAPTER 35. FUNCTIONS FOR MANAGING PARAMETER BLOCK

This chapter contains reference information on parameter block routines.

Table 35.1. Parameter Block Routines

Function	Description
slapi_pblock_destroy()	Frees a pblock from memory.
slapi_pblock_get()	Gets the value from a pblock.
Section 35.3, “slapi_pblock_init()”	Initializes an existing parameter block so that it can be reused.
slapi_pblock_new()	Creates a new pblock.
slapi_pblock_set()	Sets the value of a pblock.

35.1. SLAPI_PBLOCK_DESTROY()

Frees the specified parameter block from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_pblock_destroy( Slapi_PBlock *pb );
```

Parameters

This function takes the following parameter:

<i>pb</i>	Parameter block that you want to free.
-----------	--

Memory Concerns

The parameter block that you wish to free must have been created using [slapi_pblock_new\(\)](#). Use of this function with pblock allocated on the stack (for example, **Slapi_PBlock pb**;) or using another memory allocator is not supported and may lead to memory errors and memory leaks. For example:

```
Slapi_PBlock *pb = malloc(sizeof(Slapi_PBlock));
```

After calling this function, you should set the pblock pointer to **NULL** to avoid reusing freed memory in your function context, as in the following:

```
slapi_pblock_destroy(pb);

pb = NULL;
```

If you reuse the pointer in this way, it makes it easier to identify a segmentation fault, rather than using some difficult method to detect memory leaks or other abnormal behavior.

It is safe to call this function with a NULL pointer. For example:

```
Slapi_PBlock *pb = NULL;
slapi_pblock_destroy(pb);
```

This saves the trouble of checking for NULL before calling **slapi_pblock_destroy()**.

See Also

[slapi_pblock_new\(\)](#)

35.2. SLAPI_PBLOCK_GET()

Gets the value of a name-value pair from a parameter block.

Syntax

```
#include "slapi-plugin.h"
int slapi_pblock_get( Slapi_PBlock *pb, int arg, void *value );
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block.
<i>arg</i>	ID of the name-value pair to get. For a list of IDs that you can specify, see Part V, “Parameter Block Reference” .
<i>value</i>	Pointer to the value retrieved from the parameter block.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs (for example, if an invalid ID is specified).

Memory Concerns

The void ***value** argument should always be a pointer to the type of value you are retrieving:

```
int connid = 0;
...
retval = slapi_pblock_get(pb, SLAPI_CONN_ID, &connid);
```

SLAPI_CONN_ID is an integer value, so you will pass in a pointer to the address of an integer to get the value. Similarly, for a **char * value** (a string), pass in a pointer to/address of the value. For example:

```
char *binddn = NULL;
...
retval = slapi_pblock_get(pb, SLAPI_CONN_DN, &binddn);
```

With certain compilers on some platforms, you may have to cast the value to **(void *)**.

We recommend that you set the value to **0** or **NULL** before calling **slapi_pblock_get()** to avoid reading from uninitialized memory, in case the call to **slapi_pblock_get()** fails.

In most instances, the caller should not free the returned value. The value will usually be freed internally or through the call to **slapi_pblock_destroy()**. There are two exceptions, though:

- **If the value is explicitly set by the caller through **slapi_pblock_set()****. In this case, the caller is responsible for memory management. If the value is freed, it is strongly recommended that the free is followed by a call to **slapi_pblock_set()** with a value of **NULL**.
- **With **SLAPI_CONN_DN****. For some operations like password extop, if the given DN is empty (""), then one byte is leaked when the DN is reassigned to the bind DN. Calling **slapi_pblock_get()** with **SLAPI_CONN_DN** does a strdup, unlike most other invocations of **slapi_pblock_get()**. That memory must be freed with **slapi_ch_free_string()**.

For example, this sets the value explicitly by the caller, which is the first exception:

```
char *someparam = NULL;
...
someparam = slapi_ch_strdup(somestring);

slapi_pblock_set(pb, SOME_PARAM, someparam);

someparam = NULL; /* avoid dangling reference */
...
slapi_pblock_get(pb, SOME_PARAM, &someparam);

slapi_pblock_set(pb, SOME_PARAM, NULL); /* make sure no one else can
reference this parameter */

slapi_ch_free_string(&someparam);
...
```

Some internal functions may change the value passed in, so it is recommended to use **slapi_pblock_get()** to retrieve the value again, rather than relying on a potential dangling pointer. This is shown in the example above, which sets **someparam** to **NULL** after setting it in the pblock.

See Also

- [slapi_pblock_destroy\(\)](#)
- [slapi_pblock_set\(\)](#)

- [slapi_ch_free_string\(\)](#)

35.3. SLAPI_PBLOCK_INIT()

Initializes an existing parameter block for re-use.

Syntax

```
#include "slapi-plugin.h"

void slapi_pblock_init( Slapi_PBlock *pb );
```

Parameters

This function takes the following parameter:

<i>pb</i>	Parameter block.
-----------	------------------

Memory Concerns

The parameter block that is to be freed must have been created using [slapi_pblock_new\(\)](#). When you are finished with the parameter block, you must free it using the [slapi_pblock_destroy\(\)](#).



NOTE

The search results will not be freed from the parameter block by **slapi_pblock_init()**. Any internal search results must be freed with the [slapi_free_search_results_internal\(\)](#) function before calling **slapi_pblock_init()**, otherwise the search results will be leaked.

See Also

- [slapi_pblock_destroy\(\)](#)
- [slapi_pblock_new\(\)](#)

35.4. SLAPI_PBLOCK_NEW()

Creates a new parameter block.

Syntax

```
#include "slapi-plugin.h"

Slapi_PBlock *slapi_pblock_new();
```

Returns

This function returns a pointer to the new parameter block.

Memory Concerns

The pblock pointer allocated with this function must always be freed by `slapi_pblock_destroy()`. The use of other memory de-allocators (for example, `free()`) is not supported and may lead to crashes or memory leaks.

35.5. SLAPI_PBLOCK_SET()

Sets the value of a name-value pair in a parameter block.

Syntax

```
#include "slapi-plugin.h"
int slapi_pblock_set( Slapi_PBlock *pb, int arg, void *value );
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block.
<i>arg</i>	ID of the name-value pair to set. For a list of IDs that you can specify, see Part V, “Parameter Block Reference”
<i>value</i>	Pointer to the value that you want to set in the parameter block.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs (for example, if an invalid ID is specified).

Memory Concerns

The value to be passed in must always be a pointer, even for integer arguments. For example, if you wanted to do a search with the **ManagedSAIT** control:

```
int managedsait = 1;
...
slapi_pblock_set(pb, SLAPI_MANAGEDSAIT, &managedsait);
```

A call similar to the following example will cause a crash:

```
slapi_pblock_set(pb, SLAPI_MANAGEDSAIT, 1);
```

However, for values which are already pointers (**char * strings**, **char **arrays**, **Slapi_Backend ***, etc.), you can pass in the value directly. For example:

```
char *target_dn = slapi_ch_strdup(some_dn);
slapi_pblock_set(pb, SLAPI_TARGET_DN, target_dn);
```

or

```
slapi_pblock_set(pb, SLAPI_TARGET_DN, NULL);
```

With some compilers, you will have to cast the value argument to **(void *)**. If the caller allocates the memory passed in, the caller is responsible for freeing that memory. Also, it is recommended to use **slapi_pblock_get()** to retrieve the value to free, rather than relying on a potentially dangling pointer. See the **slapi_pblock_get()** example for more details.

When setting parameters to register a plug-in, the plug-in type must always be set first, since many of the plug-in parameters depend on the type. For example, set the **SLAPI_PLUGIN_TYPE** to extended operation before setting the list of extended operation OIDs for the plug-in.

See Also

[slapi_pblock_get\(\)](#)

CHAPTER 36. FUNCTIONS FOR HANDLING PASSWORDS

This chapter contains reference information on routines for handling passwords. The routines are listed in [Chapter 36, Functions for Handling Passwords](#).

Table 36.1. Password Handling Routines

Function	Description
slapi_pw_find_sv()	Determines whether a specified password matches one of the encrypted values of an attribute.
slapi_is_encoded()	Checks whether a value is encoded with any known algorithm.
slapi_encode()	Encodes a value with the specified algorithm.
slapi_add_pwd_control()	Sends back a password expired notification or password expiration warning.
slapi_pwpolicy_make_response_control()	Sends back information on the server password policy.

36.1. SLAPI_PW_FIND_SV()

Description

This function replaces the deprecated `slapi_pw_find()` function from previous Directory Server releases.

When the Directory Server stores the password for an entry in the **userpassword** attribute, it encodes the password using different schemes. Supported schemes are **SSHA** (default), **SHA**, **CRYPT**, and **CLEAR**.

Use this function to determine if a given password is one of the values of the **userpassword** attribute. This function determines which password scheme was used to store the password and uses the appropriate comparison function to compare a given value against the encoded values of the **userpassword** attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_pw_find_sv( Slapi_Value **vals, const Slapi_Value *v );
```

Parameters

This function takes the following parameters:

<i>vals</i>	Pointer to the array of Slapi_Value structure pointers, containing the values of an attribute that stores passwords (for example, the userpassword attribute).
<i>v</i>	Pointer to the Slapi_Value structure containing the password that you wish to check; for example, you can get this value from the SLAPI_BIND_CREDENTIALS parameter in the parameter block and create the Slapi_Value using slapi_value_init_berval() .

Returns

This function returns one of the following values:

- 0 if the password specified by *v* was found in *vals*.
- A non-zero value if the password *v* was not found in *vals*.

See Also

- [slapi_is_encoded\(\)](#)
- [slapi_encode\(\)](#)

36.2. SLAPI_IS_ENCODED()

Checks whether the specified value is encoded with any known algorithm.

Syntax

```
#include "slapi-plugin.h"
int slapi_is_encoded(char *value);
```

Parameters

This function takes the following parameter:

<i>value</i>	The value, the encoding status of which needs to be determined.
--------------	---

Returns

This function returns one of the following values:

- 1 if the value is encoded.
- 0 if the value is not encoded.

See Also

- [slapi_pw_find_sv\(\)](#)

- [slapi_encode\(\)](#)

36.3. SLAPI_ENCODE()

Encodes a value with the specified algorithm.

Syntax

```
#include "slapi-plugin.h"
char* slapi_encode(char *value, char *alg);
```

Parameters

This function takes the following parameters:

<i>value</i>	The value that needs to be encoded.
<i>alg</i>	<p>The encoding algorithm. The following algorithms are supported in a default Directory Server installation:</p> <ul style="list-style-type: none"> • CRYPT • CLEAR • SSHA • SHA <p>If you pass NULL for the <i>alg</i> parameter, the scheme used is determined by the setting of the server's passwordStorageScheme value within the server configuration entry (cn=config). If no value is present, SSHA is the default.</p>

Returns

This function returns one of the following values:

- The encoded (hashed) value.
- **NULL** if an error occurs; for example, if no matching algorithm is found.

See Also

- [slapi_pw_find_sv\(\)](#)
- [slapi_is_encoded\(\)](#)

36.4. SLAPI_ADD_PWD_CONTROL()

Sends back information about expired or expiring passwords.

Syntax

```
#include "slapi-plugin.h"
int slapi_add_pwd_control ( Slapi_PBlock *pb, char *arg, long time )
```

Parameters

This function takes the following parameter:

<i>pb</i>	Parameter block.
<i>arg</i>	Argument to the function.

Returns

This function returns one of the following values:

- **LDAP_CONTROL_PWEXPIRED** (0) if the password has expired.
- **LDAP_CONTROL_PWEXPIRING** (1, with the time in seconds) if the password has not yet expired but is within the warning period.

36.5. SLAPI_PWPOLICY_MAKE_RESPONSE_CONTROL()

Description

Sends back detailed information about password policies.

Syntax

```
#include "slapi-plugin.h"
int slapi_pwpolicy_make_response_control (Slapi_PBlock *pb, int seconds,
int logins, int error)
```

Parameters

This function takes the following parameter:

<i>pb</i>	Parameter block.
-----------	------------------

Returns

This function returns any of the following values:

- **LDAP_PWPOLICY_PWDEXPIRED** (0), if the password for the entry has expired.
- **LDAP_PWPOLICY_ACCTLOCKED** (1), if the account is locked (after repeated failed login attempts).
- **LDAP_PWPOLICY_CHGAFTERRESET** (2), if the password must be changed after an administrator as reset it.
- **LDAP_PWPOLICY_PWDMODNOTALLOWED** (3), if a password cannot be modified by the user.

- **LDAP_PWPOLICY_MUSTSUPPLYOLDPWD** (4), if the old password is necessary for a modification.
- **LDAP_PWPOLICY_INVALIDPWDSYNTAX** (5), if the password violates the policy; e.g., not using special characters or capital letters if they are required.
- **LDAP_PWPOLICY_PWDTOOSHORT** (6), if the new password is shorter than the minimum length set by the policy.
- **LDAP_PWPOLICY_PWDTOOYOUNG** (7), if there has been a minimum age set before a password can be modified.
- **LDAP_PWPOLICY_PWDINHISTORY** (8), if old passwords are stored in history.

CHAPTER 37. FUNCTIONS FOR MANAGING RDNS

This chapter contains reference information on RDN routines.

Table 37.1. RDN Routines

Function	Description
<code>slapi_rdn_add()</code>	Adds a new RDN to an existing RDN structure.
<code>slapi_rdn_compare()</code>	Compares two RDNs.
<code>slapi_rdn_contains()</code>	Checks if a Slapi_RDN structure holds any RDN matching a give type/value pair.
<code>slapi_rdn_contains_attr()</code>	Checks if a Slapi_RDN structure contains any RDN matching a given type.
<code>slapi_rdn_done()</code>	Clears a Slapi_RDN structure.
<code>slapi_rdn_free()</code>	Frees a Slapi_RDN structure.
<code>slapi_rdn_get_first()</code>	Gets the type/value pair of the first RDN.
<code>slapi_rdn_get_index()</code>	Gets the index of the RDN.
<code>slapi_rdn_get_index_attr()</code>	Gets the position and the attribute value of the first RDN.
<code>slapi_rdn_get_next()</code>	Gets the RDN type/value pair from the RDN.
<code>slapi_rdn_get_num_components()</code>	Gets the number of RDN type/value pairs.
<code>slapi_rdn_get_rdn()</code>	Gets the RDN from a Slapi_RDN structure.
<code>slapi_rdn_get_nrdn()</code>	<i>Not implemented; do not use.</i> Gets the normalized RDN from a Slapi_RDN structure.
<code>slapi_rdn_init()</code>	Initializes a Slapi_RDN structure.
<code>slapi_rdn_init_dn()</code>	Initializes a Slapi_RDN structure with an RDN value taken from a given DN.
<code>slapi_rdn_init_rdn()</code>	Initializes a Slapi_RDN structure with an RDN value.
<code>slapi_rdn_init_sdn()</code>	Initializes a Slapi_RDN structure with an RDN value taken from the DN contained in a given Slapi_RDN structure.

Function	Description
<code>slapi_rdn_isempty()</code>	Checks if an RDN value is stored in a <code>Slapi_RDN</code> structure.
<code>slapi_rdn_new()</code>	Allocates a new <code>Slapi_RDN</code> structure.
<code>slapi_rdn_new_dn()</code>	Creates a new <code>Slapi_RDN</code> structure.
<code>slapi_rdn_new_rdn()</code>	Creates a new <code>Slapi_RDN</code> structure and sets an RDN value.
<code>slapi_rdn_new_sdn()</code>	Creates a new <code>Slapi_RDN</code> structure and sets an RDN value taken from the DN contained in a given <code>Slapi_RDN</code> structure.
<code>slapi_rdn_remove()</code>	Removes an RDN type/value pair.
<code>slapi_rdn_remove_attr()</code>	Removes an RDN type/value pair from a <code>Slapi_RDN</code> structure.
<code>slapi_rdn_remove_index()</code>	Removes an RDN type/value pair from a <code>Slapi_RDN</code> structure.
<code>slapi_rdn_set_dn()</code>	Sets an RDN value in a <code>Slapi_RDN</code> structure.
<code>slapi_rdn_set_rdn()</code>	Sets an RDN in a <code>Slapi_RDN</code> structure.
<code>slapi_rdn_set_sdn()</code>	Sets an RDN value in a <code>Slapi_RDN</code> structure.
<code>slapi_sdn_add_rdn()</code>	Adds an RDN to a DN.
<code>slapi_rdn2typeval()</code>	Converts the second RDN type value to the berval value.

37.1. SLAPI_RDN_ADD()

Description

This function adds a new type/value pair to an existing RDN or sets the type/value pair as the new RDN if `rdn` is empty. This function resets the **FLAG_RDNS** flags, which means that the RDN array within the `Slapi_RDN` structure is no longer current with the new RDN.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_add(Slapi_RDN *rdn, const char *type, const char *value);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The target Slapi_RDN structure.
<i>type</i>	The type (cn , o , ou , etc.) of the RDN to be added. This parameter cannot be NULL .
<i>value</i>	The value of the RDN to be added. This parameter cannot be NULL .

Returns

This function always returns 1.

See Also

[slapi_rdn_get_num_components\(\)](#)

37.2. SLAPI_RDN_COMPARE()

Description

This function compares *rdn1* and *rdn2*. For *rdn1* and *rdn2* to be considered equal RDNs, their components do not necessarily have to be in the same order.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_compare(Slapi_RDN *rdn1, Slapi_RDN *rdn2);
```

Parameters

This function takes the following parameters:

<i>rdn1</i>	The first RDN to compare.
<i>rdn2</i>	The second RDN to compare.

Returns

This function returns one of the following values:

- 0 if *rdn1* and *rdn2* have the same RDN components.
- -1 if they do not have the same components.

37.3. SLAPI_RDN_CONTAINS()

Description

This function searches for an RDN inside of the Slapi_RDN structure *rdn* that matches both type and value as given in the parameters. This function makes a call to [slapi_rdn_get_index\(\)](#) and verifies that the returned value is anything but -1.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_contains(Slapi_RDN *rdn, const char *type, const char
*value, size_t length);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The Slapi_RDN structure containing the RDN value(s).
<i>type</i>	The type (cn , o , ou , etc.) of the RDN searched.
<i>value</i>	The value of the RDN searched.
<i>length</i>	Gives the length of value that should be taken into account for the string operation when searching for the RDN.

Returns

This function returns one of the following values:

- 1 if *rdn* contains an RDN that matches the type, value, and length.
- 0 if no RDN matches the desired type/value.

See Also

- [slapi_rdn_get_index\(\)](#)
- [slapi_rdn_contains_attr\(\)](#)

37.4. SLAPI_RDN_CONTAINS_ATTR()

Description

This function checks whether a Slapi_RDN structure contains any RDN matching a given type and, if true, gets the corresponding attribute value. This function looks for an RDN inside the Slapi_RDN structure *rdn* that matches the type given in the parameters. This function makes a call to [slapi_rdn_get_index_attr\(\)](#) and verifies that the returned value is anything but -1. If successful, it also returns the corresponding attribute value.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_contains_attr(Slapi_RDN *rdn, const char *type, char
**value);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The Slapi_RDN structure containing the RDN value(s).
<i>type</i>	Type (cn , o , ou , etc.) of the RDN searched.
<i>value</i>	Repository that will hold the value of the first RDN whose type matches the content of the parameter type. If this parameter is NULL at the return of the function, no RDN with the desired type exists within <i>rdn</i> .

Returns

This function returns one of the following values:

- 1 if *rdn* contains a RDN that matches the given type.
- 0 if there is no match.

See Also

- [slapi_rdn_get_index_attr\(\)](#)
- [slapi_rdn_contains\(\)](#)

37.5. SLAPI_RDN_DONE()

Description

This function clears the contents of a Slapi_RDN structure. It frees both the RDN value and the array of split RDNs. Those pointers are then set to **NULL**.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_done(Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

<i>rdn</i>	Pointer to the structure to be cleared.
------------	---

See Also

- [slapi_rdn_free\(\)](#)
- [slapi_rdn_init\(\)](#)

37.6. SLAPI_RDN_FREE()

Description

This function frees both the contents of the `Slapi_RDN` structure and the structure itself pointed to by the content of *rdn*.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_free(Slapi_RDN **rdn);
```

Parameters

This function takes the following parameter:

<i>rdn</i>	Pointer to the pointer of the <code>Slapi_RDN</code> structure to be freed.
------------	---

See Also

- [slapi_rdn_new\(\)](#)
- [slapi_rdn_done\(\)](#)

37.7. SLAPI_RDN_GET_FIRST()

Description

This function gets the type/value pair corresponding to the first RDN stored in a `Slapi_RDN` structure. For example, if the RDN is **cn=Joey**, the function will place **cn** in the type return parameter and **Joey** in value.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_first(Slapi_RDN *rdn, char **type, char **value);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The <code>Slapi_RDN</code> structure containing the RDN value(s).
<i>type</i>	Repository that will hold the type of the first RDN. If this parameter is NULL at the return of the function, it means <i>rdn</i> is empty.
<i>value</i>	Repository that will hold the type of the first RDN. If this parameter is NULL at the return of the function, it means <i>rdn</i> is empty.

Returns

This function returns one of the following values:

- -1 if *rdn* is empty.
- 1 if the operation is successful.

See Also

- [slapi_rdn_get_next\(\)](#)
- [slapi_rdn_get_rdn\(\)](#)

37.8. SLAPI_RDN_GET_INDEX()

Description

This function gets the index of the RDN that follows the RDN with a given type and value. The function searches for an RDN inside the Slapi_RDN structure *rdn* that matches both type and value as given in the parameters. If it succeeds, the position of the matching RDN is returned.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_index(Slapi_RDN *rdn, const char *type, const char
*value, size_t length);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The Slapi_RDN structure containing the RDN value(s).
<i>type</i>	Type (cn , o , ou , etc.) of the RDN that is searched.
<i>value</i>	Value of the RDN searched.
<i>length</i>	Gives the length of value that should be taken into account for the string comparisons when searching for the RDN.

Returns

This function returns one of the following values:

- The index of the RDN that follows the RDN matching the contents of the parameters type and value.
- -1 if no RDN stored in *rdn* matches the given type/value pair.

See Also

- [slapi_rdn_get_index_attr\(\)](#)
- [slapi_rdn_contains\(\)](#)

37.9. SLAPI_RDN_GET_INDEX_ATTR()

Description

This function searches for an RDN inside of the Slapi_RDN structure *rdn* that matches the type given in the parameters. If successful, the position of the matching RDN, as well as the corresponding attribute value, is returned.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_index_attr(Slapi_RDN *rdn, const char *type, char
**value);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The Slapi_RDN structure containing the RDN value(s).
<i>type</i>	Type (cn , o , ou , etc.) of the RDN searched.
<i>value</i>	Repository that will hold the value of the first RDN whose type matches the content of the parameter <i>type</i> . If this parameter is NULL at the return of the function, no RDN exists within <i>rdn</i> .

Returns

This function returns one of the following values:

- The real position of the first RDN within RDN that matches the content of *type*.
- -1 if there is no RDN that matches the content *type*.

See Also

[slapi_rdn_get_index\(\)](#)

37.10. SLAPI_RDN_GET_NEXT()

Description

This function gets the type/value pair corresponding to the RDN stored in the next (index+1) position inside a Slapi_RDN structure. The index of an element within an array of values is always one unit below its real position in the array.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_next(Slapi_RDN *rdn, int index, char **type, char
**value);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The Slapi_RDN structure containing the RDN value(s).
<i>index</i>	Indicates the position of the RDN the precedes the currently desired RDN.
<i>type</i>	Repository that will hold the type (cn , o , ou , etc.) of the next (index+1) RDN. If this parameter is NULL at the return of the function, the RDN does not exist.
<i>value</i>	Repository that will hold the value of the next (index+1) RDN. If this parameter is NULL , the RDN does not exist.

Returns

This function returns one of the following values:

- The real position of the retrieved RDN if the operation was successful.
- -1 if there is no RDN in the index position.

See Also

- [slapi_rdn_get_first\(\)](#)
- [slapi_rdn_get_rdn\(\)](#)

37.11. SLAPI_RDN_GET_NUM_COMPONENTS()

Gets the number of RDN type/value pairs present in a Slapi_RDN structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_num_components(Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

<i>rdn</i>	The target Slapi_RDN structure.
------------	---------------------------------

Returns

This function returns the number of RDN type/value pairs present in *rdn*.

See Also

`slapi_rdn_add()`

37.12. SLAPI_RDN_GET_RDN()

Gets the RDN from a Slapi_RDN structure.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_rdn_get_rdn(const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

<i>rdn</i>	The Slapi_RDN structure holding the RDN value.
------------	--

Returns

This function returns the RDN value.

37.13. SLAPI_RDN_GET_NRDN()

Not implemented; do not use.

Gets the new normalized RDN from a Slapi_RDN structure.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_rdn_get_nrdn(const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

<i>rdn</i>	The Slapi_RDN structure holding the RDN value.
------------	--

Returns

This function returns the new RDN value.

37.14. SLAPI_RDN_INIT()

Description

This function initializes a given Slapi_RDN structure with **NULL** values; both the RDN value and the array of split RDNs are set to **NULL**.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init(Slapi_RDN *rdn);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The Slapi_RDN structure to be initialized.
------------	--

See Also

- [slapi_rdn_new\(\)](#)
- [slapi_rdn_free\(\)](#)
- [slapi_rdn_done\(\)](#)

37.15. SLAPI_RDN_INIT_DN()

Description

This function initializes a given Slapi_RDN structure with the RDN value taken from the DN passed in the *dn* parameter.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init_dn(Slapi_RDN *rdn,const Slapi_DN *dn);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The Slapi_RDN structure to be initialized.
<i>dn</i>	The DN value whose RDN will be used to initialize the new Slapi_RDN structure.

See Also

- [slapi_rdn_init_sdn\(\)](#)
- [slapi_rdn_init_rdn\(\)](#)

37.16. SLAPI_RDN_INIT_RDN()

Description

This function initializes a given Slapi_RDN structure with the RDN value *infromrdn*.

Syntax

```
#include "slapi-plugin.h"
```

```
void slapi_rdn_init_rdn(Slapi_RDN *rdn,const Slapi_RDN *fromrdn);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The Slapi_RDN structure to be initialized.
<i>fromrdn</i>	The RDN value to be set in the new Slapi_RDN structure.

See Also

- [slapi_rdn_init_dn\(\)](#)
- [slapi_rdn_init_sdn\(\)](#)

37.17. SLAPI_RDN_INIT_SDN()

Description

This function initializes a given Slapi_RDN structure with the RDN value taken from the DN passed within the Slapi_DN structure of the *sdn* parameter.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init_sdn(Slapi_RDN *rdn,const Slapi_DN *sdn);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The Slapi_RDN structure to be initialized.
<i>sdn</i>	The Slapi_DN structure containing the DN value whose RDN will be used to initialize the new Slapi_RDN structure.

See Also

- [slapi_rdn_init_dn\(\)](#)
- [slapi_rdn_init_rdn\(\)](#)

37.18. SLAPI_RDN_ISEMPTY()

Checks whether an RDN value is stored in a Slapi_RDN structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_isempty(const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

<i>rdn</i>	The target Slapi_RDN structure.
------------	---------------------------------

Returns

This function returns one of the following values:

- 1 if there is no RDN value present.
- 0 if *rdn* contains a value.

See Also

- [slapi_rdn_init\(\)](#)
- [slapi_rdn_done\(\)](#)
- [slapi_rdn_free\(\)](#)

37.19. SLAPI_RDN_NEW()

Description

This function creates a new Slapi_RDN structure by allocating the necessary memory and initializing both the RDN value and the array of split RDNs to **NULL**.

Syntax

```
#include "slapi-plugin.h"
Slapi_RDN * slapi_rdn_new();
```

Parameters

This function takes no parameters.

Returns

This function returns a pointer to the newly allocated, and still empty, Slapi_RDN structure.

See Also

- [slapi_rdn_init\(\)](#)
- [slapi_rdn_done\(\)](#)
- [slapi_rdn_free\(\)](#)

37.20. SLAPI_RDN_NEW_DN()

Description

This function creates a new `Slapi_RDN` structure and initializes its RDN with the value taken from the DN passed in the *dn* parameter.

Syntax

```
#include "slapi-plugin.h"
Slapi_RDN *slapi_rdn_new_dn(const Slapi_DN *dn);
```

Parameters

This function takes the following parameter:

<i>dn</i>	The DN value whose RDN will be used to initialize the new <code>Slapi_RDN</code> structure.
-----------	---

Returns

This function returns a pointer to the new `Slapi_RDN` structure initialized with the RDN taken from the DN value in *dn*.

Memory Concerns

The memory is allocated by the function itself.

See Also

- [slapi_rdn_new_sdn\(\)](#)
- [slapi_rdn_new_rdn\(\)](#)

37.21. SLAPI_RDN_NEW_RDN()

Description

This function creates a new `Slapi_RDN` structure and initializes its RDN with the value of *fromrdn*.

Syntax

```
#include "slapi-plugin.h"
Slapi_RDN * slapi_rdn_new_rdn(const Slapi_RDN *fromrdn);
```

Parameters

This function takes the following parameter:

<i>fromrdn</i>	The RDN value to be set in the new <code>Slapi_RDN</code> structure.
----------------	--

Returns

This function returns a pointer to the new `Slapi_RDN` structure with an RDN set to the content of *fromrdn*.

Memory Concerns

The memory is allocated by the function itself.

See Also

[slapi_rdn_new_sdn\(\)](#) [slapi_rdn_new_dn\(\)](#)

37.22. SLAPI_RDN_NEW_SDN()

Description

This function creates a new `Slapi_RDN` structure and initializes its RDN with the value taken from the DN passed within the `Slapi_RDN` structure of the *sdn* parameter.

Syntax

```
#include "slapi-plugin.h"
vSlapi_RDN *slapi_rdn_new_sdn(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

<i>sdn</i>	Slapi_RDN structure containing the DN value whose RDN will be used to initialize the new Slapi_RDN structure.
------------	---

Returns

This function returns a pointer to the new `Slapi_RDN` structure initialized with the RDN taken from the DN value in *dn*.

Memory Concerns

The memory is allocated by the function itself.

See Also

- [slapi_rdn_new_dn\(\)](#)
- [slapi_rdn_new_rdn\(\)](#)

37.23. SLAPI_RDN_REMOVE()

This function removes the RDN from *rdn* that matches the given criteria (*type*, *value*, and *length*).

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_remove(Slapi_RDN *rdn, const char *type, const char *value,
size_t length);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The target Slapi_RDN structure.
<i>type</i>	Type (<i>cn</i> , o , ou , etc.) of the RDN searched.
<i>value</i>	The value of the RDN searched.
<i>length</i>	Gives the length of <i>value</i> that should be taken into account for the string comparisons when searching for the RDN.

Returns

This function returns one of the following values:

- 1 if the RDN is removed from *rdn*.
- 0 if no RDN is removed.

See Also

- [slapi_rdn_remove_attr\(\)](#)
- [slapi_rdn_remove_index\(\)](#)

37.24. SLAPI_RDN_REMOVE_ATTR()

This function removes the first RDN from a Slapi_RDN structure matches the given type.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_remove_attr(Slapi_RDN *rdn, const char *type);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The target Slapi_RDN structure.
<i>type</i>	Type (cn , o , ou , etc.) of the RDN searched.

Returns

This function returns one of the following values:

- 1 if the RDN is removed from *rdn*.
- 0 if no RDN is removed.

See Also

- [slapi_rdn_remove\(\)](#)

- [slapi_rdn_remove_index\(\)](#)

37.25. SLAPI_RDN_REMOVE_INDEX()

Description

This function removes an RDN type/value pair from a Slapi_RDN structure with *atindex* index (placed in the *atindex+1* position).

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_remove_index(Slapi_RDN *rdn, int atindex);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The target Slapi_RDN structure.
<i>atindex</i>	The index of the RDN type/value pair to remove.

Returns

This function returns one of the following values:

- 1 if the RDN is removed from *rdn*.
- 0 if no RDN is removed because either *rdn* is empty or the index goes beyond the number of RDNs present.

See Also

- [slapi_rdn_remove\(\)](#)
- [slapi_rdn_remove_attr\(\)](#)

37.26. SLAPI_RDN_SET_DN()

Description

This function sets an RDN value in a Slapi_RDN structure. The structure is freed from memory and freed of any previous content before setting the new RDN. The new RDN is taken from the DN value present in the *dn* parameter.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_set_dn(Slapi_RDN *rdn, const Slapi_DN *dn);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The target Slapi_RDN structure.
<i>dn</i>	The DN value whose RDN will be set in <i>rdn</i> .

See Also

- [slapi_rdn_set_sdn\(\)](#)
- [slapi_rdn_set_rdn\(\)](#)

37.27. SLAPI_RDN_SET_RDN()**Description**

This function sets an RDN value in a Slapi_RDN structure. The structure is freed from memory and freed of any previous content before setting the new RDN.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_set_rdn(Slapi_RDN *rdn, const Slapi_RDN *fromrdn);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The target Slapi_RDN structure.
<i>fromrdn</i>	The RDN value to be set in <i>rdn</i> .

See Also

- [slapi_rdn_set_dn\(\)](#)
- [slapi_rdn_set_sdn\(\)](#)

37.28. SLAPI_RDN_SET_SDN()**Description**

This function sets an RDN value in a Slapi_RDN structure. The structure is freed from memory and freed of any previous content before setting the new RDN. The new RDN is taken from the DN value present inside of a Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_set_sdn(Slapi_RDN *rdn, const Slapi_DN *sdn);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	The target Slapi_RDN structure.
<i>sdn</i>	The Slapi_RDN structure containing the DN value whose RDN will be set in <i>rdn</i> .

See Also

- [slapi_rdn_set_dn\(\)](#)
- [slapi_rdn_set_rdn\(\)](#)

37.29. SLAPI_RDN2TYPEVAL()

Converts the second RDN type value to the berval value.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn2typeval( char *rdn, char **type, struct berval *bv );
```

Parameters

This function takes the following parameters:

<i>rdn</i>	Second RDN value
<i>type</i>	Pointer to the attribute type of the second RDN.
<i>bv</i>	Pointer to the berval value structure.

Returns

This function returns the new RDN value as a berval value in *bv*. This function can be used for creating the RDN as an attribute value since it returns the value of the RDN in the berval structure.

See Also

`moddn_rdn_add_needed()`

CHAPTER 38. FUNCTIONS FOR MANAGING ROLES

This chapter contains reference information on routines that help you deal with roles.

Table 38.1. Routines for Roles

Function	Description
slapi_role_check()	Checks if the entry pointed to by entry_to_check contains the role indicated by role_dn .
slapi_register_role_check()	Allows registering of another function, other than the default, to use in slapi_role_check() .

38.1. SLAPI_ROLE_CHECK()

Checks if the entry pointed to by *entry_to_check* contains the role indicated by *role_dn*.

Syntax

```
#include "slapi-plugin.h"
int slapi_role_check(Slapi_Entry *entry_to_check, Slapi_DN *role_dn, int
*present);
```

Parameters

This function takes the following parameters:

<i>entry_to_check</i>	The entry in which the presence of a role is to be checked.
<i>role_dn</i>	The DN of the role for which to check.
<i>present</i>	Pointer to an integer where the result, present or not present, will be placed.

Returns

This function returns one of the following values:

- 0 for success; if *role_dn* is present in *entry_to_check*, *present* is set to 0.
- A non-zero value (error condition) if the presence of the role is undetermined.

38.2. SLAPI_REGISTER_ROLE_CHECK()

Allows registering of another function, other than the default, to use in [slapi_role_check\(\)](#). It is strongly recommended that the default should be used.

Syntax

–

```
#include "slapi-plugin.h"
void slapi_register_role_check(roles_check_fn_type check_fn);
```

Parameters

This function takes the following parameter:

<i>check_fn</i>	Function for registering.
-----------------	---------------------------

CHAPTER 39. FUNCTIONS FOR MANAGING DNS

This chapter contains reference information on DN routines.

Table 39.1. DN Routines

Function	Description
<code>slapi_dn_isroot()</code>	Determines if the DN is the root DN for the local database.
<code>slapi_dn_normalize_case()</code>	Converts a DN to canonical format and all characters to lower case.
<code>slapi_dn_normalize_to_end()</code>	Normalizes part of a DN value.
<code>slapi_moddn_get_newdn()</code>	Builds the new DN of an entry.
<code>slapi_sdn_add_rdn()</code>	Adds the RDN contained in a <code>Slapi_RDN</code> structure to the DN contained in a <code>Slapi_DN</code> structure.
<code>slapi_sdn_compare()</code>	Compares two DN's.
<code>slapi_sdn_copy()</code>	Copies a DN.
<code>slapi_sdn_done()</code>	Clears a <code>Slapi_DN</code> structure.
<code>slapi_sdn_dup()</code>	Duplicates a <code>Slapi_DN</code> structure.
<code>slapi_sdn_free()</code>	Frees a <code>Slapi_DN</code> structure.
<code>slapi_sdn_get_backend_parent()</code>	Gets the DN of the parent within a specific backend.
<code>slapi_sdn_get_dn()</code>	Gets the DN from a <code>Slapi_DN</code> structure.
<code>slapi_sdn_get_ndn()</code>	Gets the normalized DN of a <code>Slapi_DN</code> structure.
<code>slapi_sdn_get_ndn_len()</code>	Gets the length of the normalized DN of a <code>Slapi_DN</code> structure.
<code>slapi_sdn_get_parent()</code>	Get the parent DN of a given <code>Slapi_DN</code> structure.
<code>slapi_sdn_get_rdn()</code>	Gets the RDN from an NDN.

Function	Description
<code>slapi_sdn_is_rdn_component()</code>	<i>Not implemented; do not use.</i> Checks if there is a RDN value that is a component of the DN structure.
<code>slapi_sdn_isempty()</code>	Checks if there is a DN value stored in a Slapi_DN structure.
<code>slapi_sdn_isgrandparent()</code>	Checks if a DN is the parent of the parent of a DN.
<code>slapi_sdn_isparent()</code>	Checks if a DN is the parent of a DN.
<code>slapi_sdn_issuffix()</code>	Checks if a Slapi_DN structure contains a suffix of another.
<code>slapi_sdn_new()</code>	Allocates new Slapi_DN structure.
<code>slapi_sdn_new_dn_byref()</code>	Creates a new Slapi_DN structure.
<code>slapi_sdn_new_dn_byval()</code>	Creates a new Slapi_DN structure.
<code>slapi_sdn_new_dn_passin()</code>	Creates a new Slapi_DN structure.
<code>slapi_sdn_new_ndn_byref()</code>	Creates a new Slapi_DN structure.
<code>slapi_sdn_new_ndn_byval()</code>	Creates a new Slapi_DN structure.
<code>slapi_sdn_scope_test()</code>	Checks if an entry is in the scope of a certain base DN.
<code>slapi_sdn_set_dn_byref()</code>	Sets a DN value in a Slapi_DN structure.
<code>slapi_sdn_set_dn_byval()</code>	Sets a DN value in a Slapi_DN structure.
<code>slapi_sdn_set_dn_passin()</code>	Sets a DN value in a Slapi_DN structure.
<code>slapi_sdn_set_ndn_byref()</code>	Sets a normalized DN in a Slapi_DN structure.
<code>slapi_sdn_set_ndn_byval()</code>	Sets a normalized DN in a Slapi_DN structure.
<code>slapi_sdn_set_parent()</code>	Sets a new parent in an entry.
<code>slapi_sdn_set_rdn()</code>	Sets a new RDN for an entry.

39.1. SLAPI_DN_ISROOT()

Determines whether the specified DN is the root DN for this local database. Before calling this function, you should call [slapi_dn_normalize_case\(\)](#) to normalize the DN and convert all characters to lowercase.

Syntax

```
#include "slapi-plugin.h"
int slapi_dn_isroot( const Slapi_DN *dn );
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block.
<i>dn</i>	DN that you want to check.

Returns

This function returns one of the following values:

- 1 if the specified DN is the root DN of the local database.
- 0 if the specified DN is not the root DN of the local database.

See Also

[slapi_be_issuffix\(\)](#)

39.2. SLAPI_DN_NORMALIZE_CASE()

Converts a distinguished name (DN) to canonical format and converts all characters to lowercase. Calling this function has the same effect as calling the [slapi_sdn_get_ndn\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_normalize_case( Slapi_DN *dn );
```

Parameters

This function takes the following parameter:

<i>dn</i>	DN that you want to normalize and convert to lowercase.
-----------	---

Returns

This function returns the normalized DN with all lowercase characters. A variable passed in as the **dn** argument is also converted in place.

39.3. SLAPI_DN_NORMALIZE_TO_END()

Normalizes part of a DN value, specifically the part going from what is pointed to by **dn** to that pointed to by **end**. This routine does not **NULL** terminate the normalized bit pointed to by **dn** at the return of the function.

If the argument **end** happens to be **NULL**, this routine does basically the same thing as `slapi_sdn_get_ndn()`, except for **NULL** terminating the normalized DN.



WARNING

This function does *not* null-terminate the string. Use this function only if you know what you are doing.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_normalize_to_end( Slapi_DN *dn, char *end);
```

Parameters

This function takes the following parameters:

<i>dn</i>	DN value to be normalized.
<i>end</i>	Pointer to the end of what will be normalized from the DN value in dn . If this argument is NULL , the DN value in dn will be wholly normalized.

Returns

This function returns a pointer to the end of the *dn* that has been normalized. For example, if the RDN is **cn=Jane** and the DN is **l=US, dc=example,dc=com**, the new DN will be **cn=Jane,l=US, dc=example,dc=com**.

39.4. SLAPI_MODDN_GET_NEWDN()

Description

This function is used for *moddn* operations and builds a new DN out of a new RDN and the DN of the new parent.

The new DN is worked out by adding the new RDN in *newrdn* to a parent DN. The parent will be the value in *newsuperordn* if different from **NULL**, and will otherwise be taken from *dn_olddn* by removing the old RDN (the parent of the entry will still be the same as the new DN).

Syntax


```
#include "slapi-plugin.h"
char * slapi_moddn_get_newdn(Slapi_DN *dn_olddn, char *newrdn, char
*newsuperiordn);
```

Parameters

This function takes the following parameters:

<i>dn_olddn</i>	The old DN value.
<i>newrdn</i>	The new RDN value.
<i>newsuperiordn</i>	If not NULL , will be the DN of the future superior entry of the new DN, which will be worked out by adding the value in <i>newrdn</i> in front of the content of this parameter.

Returns

This function returns the new DN for the entry whose previous DN was *dn_olddn*.

39.5. SLAPI_SDN_ADD_RDN()

Adds the RDN contained in a Slapi_RDN structure to the DN contained in a Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_add_rdn(Slapi_DN *sdn, const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	Slapi_DN structure containing the value to which a new RDN is to be added.
<i>rdn</i>	Slapi_RDN structure containing the RDN value that is to be added to the DN value.

Returns

This function returns the Slapi_DN structure with the new DN formed by adding the RDN value in *rdn* to the DN value in **dn**.

See Also

[slapi_sdn_set_rdn\(\)](#)

39.6. SLAPI_SDN_COMPARE()

Description

This function compares two DN's, **sdn1** and *sdn2*. The comparison is case sensitive.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_compare( const Slapi_DN *sdn1, const Slapi_DN *sdn2 );
```

Parameters

This function takes the following parameters:

<i>sdn1</i>	DN to compare with the value in sdn2 .
<i>sdn2</i>	DN to compare with the value in sdn1 .

Returns

This function returns one of the following values:

- 0 if *sdn1* is equal to *sdn2*.
- -1 if *sdn1* is **NULL**.
- 1 if *sdn2* is **NULL** and *sdn1* is not **NULL**.

39.7. SLAPI_SDN_COPY()

Description

This function copies the DN in *from* to the structure pointed by *to*.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_copy(const Slapi_DN *from, Slapi_DN *to);
```

Parameters

This function takes the following parameters:

<i>from</i>	The original DN.
<i>to</i>	Destination of the copied DN, containing the copy of the DN in <i>from</i> .

Memory Concerns

to must be allocated in advance of calling this function.

See Also

[slapi_sdn_dup\(\)](#)

39.8. SLAPI_SDN_DONE()

Description

This function clears the contents of a `Slapi_DN` structure. It frees both the DN and the normalized DN, if any, and sets those pointers to **NULL**.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_done(Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

<i>sdn</i>	Pointer to the structure to clear.
------------	------------------------------------

See Also

[slapi_sdn_free\(\)](#)

39.9. SLAPI_SDN_DUP()

Duplicates a `Slapi_DN` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN * slapi_sdn_dup(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

<i>sdn</i>	Pointer to the <code>Slapi_DN</code> structure to duplicate.
------------	--

Returns

This function returns a pointer to a duplicate of *sdn*.

See Also

- [slapi_sdn_copy\(\)](#)
- [slapi_sdn_new\(\)](#)

39.10. SLAPI_SDN_FREE()

Description

This function frees the `Slapi_DN` structure and its contents pointed to by the contents of *sdn*.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_free(Slapi_DN **sdn);
```

Parameters

This function takes the following parameter:

<i>sdn</i>	Pointer tot he pointer of the Slapi_DN structure to be freed.
------------	---

See Also

- [slapi_sdn_done\(\)](#)
- [slapi_sdn_new\(\)](#)

39.11. SLAPI_SDN_GET_BACKEND_PARENT()

Gets the DN of the parent of an entry within a specific backend.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_get_backend_parent(const Slapi_DN *sdn, Slapi_DN
*sdn_parent,const Slapi_Backend *backend);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	DN of the entry whose parent is searched.
<i>sdn_parent</i>	Parent DN of <i>sdn</i> .
<i>backend</i>	Backend of which the parent of <i>sdn</i> is to be searched.

Returns

This function gets the parent DN of an entry within a given backend. The parent DN is returned is *sdn_parent*, unless *sdn* is empty or is a suffix of the backend itself. In this case, *sdn_parent* is empty.

Memory Concerns

A Slapi_DN structure for *sdn_parent* must be allocated before calling this function.

See Also

[slapi_sdn_get_parent\(\)](#)

39.12. SLAPI_SDN_GET_DN()

Description

This function retrieves the DN value of a Slapi_DN structure. The returned value can be the normalized DN (in a canonical format and in lower case) if no other value is present.

Syntax

```
#include "slapi-plugin.h"
const char * slapi_sdn_get_dn(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

<i>sdn</i>	The Slapi_DN structure containing the DN value.
------------	---

Returns

This function returns the DN value.

See Also

- [slapi_sdn_get_ndn\(\)](#)
- [slapi_sdn_get_parent\(\)](#)
- [slapi_sdn_get_rdn\(\)](#)

39.13. SLAPI_SDN_GET_NDN()

Description

This function retrieves the normalized DN (in a canonical format and lower case) from a Slapi_DN structure and normalizes *sdn* if it has not already been normalized.

Syntax

```
#include "slapi-plugin.h"
const char * slapi_sdn_get_ndn(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

<i>sdn</i>	The Slapi_DN structure containing the DN value.
------------	---

Returns

This function returns the normalized DN value.

See Also

[slapi_sdn_get_dn\(\)](#)

39.14. SLAPI_SDN_GET_NDN_LEN()

Description

This function gets the length of the normalized DN of a `Slapi_DN` structure. This function contains the length of the normalized DN and normalizes *sdn* if it has not already been normalized.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_get_ndn_len(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

<i>sdn</i>	The <code>Slapi_DN</code> structure containing the DN value.
------------	--

Returns

This function returns the length of the normalized DN.

39.15. SLAPI_SDN_GET_PARENT()

Description

This function returns a `Slapi_DN` structure containing the parent DN of the DN kept in the `Slapi_DN` structure pointed to by *sdn*.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_get_parent(const Slapi_DN *sdn, Slapi_DN *sdn_parent);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	Pointer to the <code>Slapi_DN</code> structure containing the DN whose parent is searched.
<i>sdn_parent</i>	Pointer to the <code>Slapi_DN</code> structure where the parent DN is returned.

See Also

[slapi_sdn_get_backend_parent\(\)](#)

39.16. SLAPI_SDN_GET_RDN()

Description

This function takes the DN stored in the `Slapi_DN` structure pointed to by *sdn* and retrieves its returned RDN within the `Slapi_RDN` structure pointed to by *rdn*.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_get_rdn(const Slapi_DN *sdn, Slapi_RDN *rdn);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	Pointer to the <code>Slapi_DN</code> structure containing the DN.
<i>rdn</i>	Pointer to the <code>Slapi_RDN</code> structure where the RDN is returned.

See Also

- [slapi_sdn_get_dn\(\)](#)
- [slapi_sdn_add_rdn\(\)](#)
- [slapi_sdn_is_rdn_component\(\)](#)

39.17. SLAPI_SDN_IS_RDN_COMPONENT()

Not implemented; do not use.

Description

This function checks whether a `Slapi_DN` structure contains an RDN value that is a component of the DN structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_is_rdn_component(const Slapi_DN *rdn, const Slapi_Attr *a,
const Slapi_Value *v);
```

Parameters

This function takes the following parameters:

<i>rdn</i>	Pointer to the <code>Slapi_DN</code> structure that is going to be checked.
<i>a</i>	A pointer to an attribute used to check the RDN value.

<i>v</i>	Holds the value of the attribute.
----------	-----------------------------------

Returns

This function returns one of the following values:

- **1** if there is no RDN value (normalized or not) present in the `Slapi_DN` structure.
- **0** if *rdn* is a component of the `Slapi_DN` structure.

See Also

[slapi_sdn_get_rdn\(\)](#)

39.18. SLAPI_SDN_ISEMPTY()

Description

This function checks whether a `Slapi_DN` structure contains a normalized or non-normalized value.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_isempty( const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

<i>sdn</i>	Pointer to the <code>Slapi_DN</code> structure that is going to be checked.
------------	---

Returns

This function returns one of the following values:

- **1** if there is no DN value (normalized or not) present in the `Slapi_DN` structure.
- **0** if *sdn* is not empty.

See Also

[slapi_sdn_done\(\)](#)

39.19. SLAPI_SDN_ISGRANDPARENT()

Checks whether a DN is the parent of the parent of a given DN.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_isgrandparent( const Slapi_DN *parent, const Slapi_DN *child
);
```


Parameters

This function takes the following parameters:

<i>parent</i>	Pointer to the Slapi_DN structure containing the DN which claims to be the grandparent DN of the DN in <i>child</i> .
<i>child</i>	Pointer to the Slapi_DN structure containing the DN of the supposed "grandchild" of the DN in the structure pointed to by <i>parent</i> .

Returns

This function returns one of the following values:

- 1 if the DN in *parent* is the grandparent of the DN in *child*.
- 0 if the DN in *parent* does not match the DN of the grandparent of the DN in *child*.

See Also

- [slapi_sdn_isparent\(\)](#)
- [slapi_sdn_issuffix\(\)](#)
- [slapi_sdn_get_parent\(\)](#)

39.20. SLAPI_SDN_ISPARENT()

Checks whether a DN is the parent of a given DN.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_isparent( const Slapi_DN *parent, const Slapi_DN *child );
```

Parameters

This function takes the following parameters:

<i>parent</i>	Pointer to the Slapi_DN structure containing the DN which claims to be the parent of the DN in <i>child</i> .
<i>child</i>	Pointer to the Slapi_DN structure containing the DN of the supposed child of the DN in the structure pointed to by <i>parent</i> .

Returns

This function returns one of the following values:

- 1 if the DN in *parent* is the parent of the DN in *child*.

- 0 if the DN in *parent* does not match the DN of the parent of the DN in *child*.

See Also

- [slapi_sdn_issuffix\(\)](#)
- [slapi_sdn_get_parent\(\)](#)

39.21. SLAPI_SDN_ISSUFFIX()

Checks whether a Slapi_DN structure contains a suffix of another Slapi_DN structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_issuffix(const Slapi_DN *sdn, const Slapi_DN *suffixsdn);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	Pointer to the Slapi_DN structure to be checked.
<i>suffixsdn</i>	Pointer to the Slapi_DN structure of the suffix.

Returns

This function returns one of the following values:

- 1 if the DN in *suffixsdn* is the suffix of *sdn*.
- 0 if the DN in *suffixsdn* is not a suffix of *sdn*.

See Also

[slapi_sdn_isparent\(\)](#)

39.22. SLAPI_SDN_NEW()

Description

This function creates a new Slapi_DN structure by allocating the necessary memory and initializing both DN and normalized DN values to **NULL**.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new();
```

Parameters

This function takes no parameters.

Returns

This function returns a pointer to the newly allocated, and still empty, `Slapi_DN` structure.

See Also

- [slapi_sdn_free\(\)](#)
- [slapi_sdn_copy\(\)](#)
- [slapi_sdn_done\(\)](#)

39.23. SLAPI_SDN_NEW_DN_BYREF()

Description

This function creates a new `Slapi_DN` structure and initializes its DN with the value of *dn*. The DN of the new structure will point to the same string pointed to by *dn*; the DN value is passed in to the parameter by reference. However, the **FLAG_DN** flag is not set, and no counter is incremented.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_dn_byref(const char *dn);
```

Parameters

This function takes the following parameter:

<i>dn</i>	The DN value to be set in the new <code>Slapi_DN</code> structure.
-----------	--

Returns

This function returns a pointer to the new `Slapi_DN` structure with a DN value set to the content of *dn*.

Memory Concerns

The memory is allocated by the function itself.

See Also

- [slapi_sdn_new_dn_byval\(\)](#)
- [slapi_sdn_new_dn_passin\(\)](#)

39.24. SLAPI_SDN_NEW_DN_BYVAL()

Description

This function creates a new `Slapi_DN` structure and initializes its DN with the value of *dn*. The DN of the new structure will point to a copy of the string pointed to by *dn*; the DN value is passed in to the parameter by value. The **FLAG_DN** flag is set, and the internal counter is incremented.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_dn_byval(const char *dn);
```

Parameters

This function takes the following parameter:

<i>dn</i>	The DN value to be set in the new Slapi_DN structure.
-----------	---

Returns

This function returns a pointer to the new Slapi_DN structure with a DN valueset to the content of *dn*.

Memory Concerns

The memory is allocated by the function itself.

See Also

- [slapi_sdn_new_dn_byref\(\)](#)
- [slapi_sdn_new_dn_passin\(\)](#)

39.25. SLAPI_SDN_NEW_DN_PASSIN()

Description

This function creates a new Slapi_DN structure and initializes its DN with the value of *dn*. The DN of the new structure will point to the string pointed to by *dn*. The **FLAG_DN** flag is set, and the internal counter is incremented.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_dn_passin(const char *dn);
```

Parameters

This function takes the following parameter:

<i>dn</i>	The DN value to be set the new Slapi_DN structure.
-----------	--

Returns

This function returns a pointer to the new Slapi_DN structure with DN valueset to the content of *dn*.

Memory Concerns

The memory is allocated by the function itself.

See Also

- [slapi_sdn_new_dn_byval\(\)](#)
- [slapi_sdn_new_ndn_byref\(\)](#)

39.26. SLAPI_SDN_NEW_NDN_BYREF()

Description

This function creates a new `Slapi_DN` structure and initializes its normalized DN with the value of *ndn*. The normalized DN of the new structure will point to the same string pointed to by *ndn*; the normalized DN value is passed into the parameter by reference. However, the **FLAG_NDN** flag is not set, and no counter is incremented.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_ndn_byref(const char *ndn);
```

Parameters

This function takes the following parameter:

<i>ndn</i>	The normalized DN value to be set in the new <code>Slapi_DN</code> structure.
------------	---

Returns

This function returns a pointer to the new `Slapi_DN` structure with a normalized DN valueset to the content of *ndn*.

Memory Concerns

The memory is allocated by the function itself.

See Also

[slapi_sdn_new_ndn_byval\(\)](#)

39.27. SLAPI_SDN_NEW_NDN_BYVAL()

Description

This function creates a new `Slapi_DN` structure and initializes its normalized DN with the value of *ndn*. The normalized DN of the new structure will point to a copy of the string pointed to by *ndn*; the normalized DN value is passed into the parameter by value. The **FLAG_DND** flag is set, and the internal counter is incremented.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_ndn_byval(const char *ndn);
```

Parameters

This function takes the following parameter:

<i>ndn</i>	The normalized DN value to be set in the new Slapi_DN structure.
------------	--

Returns

This function returns a pointer to the new Slapi_DN structure with a normalized DN valueset to the content of *ndn*.

Memory Concerns

The memory is allocated by the function itself.

See Also

[slapi_sdn_new_ndn_byref\(\)](#)

39.28. SLAPI_SDN_SCOPE_TEST()

Description

This function carries out a simple test to check whether the DN passed in the *dn* parameter is actually in the scope of the base DN according to the values passed into the *scope* and *base* parameters.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_scope_test( const Slapi_DN *dn, const Slapi_DN *base, int
scope );
```

Parameters

This function takes the following parameters:

<i>dn</i>	The DN of the entry subject of scope test.
<i>base</i>	The base DN against which <i>dn</i> is going to be tested.
<i>scope</i>	The scope tested. This parameter can take one of the following levels: <ul style="list-style-type: none">• LDAP_SCOPE_BASE- where the entry DN should be the same as the base DN.• LDAP_SCOPE_ONELEVEL- where the base DN should be the parent of the entry.• DNLDAAP_SCOPE_SUBTREE- where the base DN should at least be the suffix of the entry DN.

Returns

This function returns non-zero if *dn* matches the scoping criteria given by base and scope.

See Also

- [slapi_sdn_compare\(\)](#)
- [slapi_sdn_isparent\(\)](#)
- [slapi_sdn_issuffix\(\)](#)

39.29. SLAPI_SDN_SET_DN_BYREF()

Description

This function sets a DN value in a Slapi_DN structure. The DN of the new structure will point to the same string pointed to by *dn*; the DN value is passed into the parameter by value. However, the **FLAG_DN** flag is not set, and no internal counter is incremented.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_dn_byref(Slapi_DN *sdn, const Slapi_DN *dn);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	The target Slapi_DN structure.
<i>dn</i>	The DN value to be set in <i>sdn</i> .

Returns

This function returns a pointer to the Slapi_DN structure containing the new DN value.

See Also

- [slapi_sdn_set_dn_byval\(\)](#)
- [slapi_sdn_set_dn_passin\(\)](#)

39.30. SLAPI_SDN_SET_DN_BYVAL()

Description

This function sets a DN value in a Slapi_DN structure. The DN of the new structure will point to a copy of the string pointed to by *dn*; the DN value is passed into the parameter by value. The **FLAG_DN** flag is set, and the internal counters are incremented.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_dn_byval(Slapi_DN *sdn, const Slapi_DN *dn);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	The target Slapi_DN structure.
<i>dn</i>	The DN value to be set in <i>sdn</i> .

Returns

This function returns a pointer to the Slapi_DN structure containing the new DN value.

See Also

- [slapi_sdn_set_ndn_byref\(\)](#)
- [slapi_sdn_set_dn_passin\(\)](#)

39.31. SLAPI_SDN_SET_DN_PASSIN()

Description

This function sets a DN value in a Slapi_DN structure. The DN of the new structure will point to the same string pointed to by *dn*. The **FLAG_DN** flag is set, and the internal counters are incremented.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_dn_passin(Slapi_DN *sdn, const Slapi_DN *dn);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	The target Slapi_DN structure.
<i>dn</i>	DN value to be set in <i>sdn</i> .

Returns

This function returns a pointer to the Slapi_DN structure containing the new DN value.

See Also

- [slapi_sdn_set_dn_byval\(\)](#)
- [slapi_sdn_set_dn_byref\(\)](#)

39.32. SLAPI_SDN_SET_NDN_BYREF()

Description

This function sets a normalized DN value in a Slapi_DN structure. The normalized DN of the new structure will point to the same string pointed to by *ndn*; the normalized DN value is passed into the parameter by reference. However, the **FLAG_DN** flag is not set, and no

internal counter is incremented.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_ndn_byref(Slapi_DN *sdn, const char *ndn);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	The target Slapi_DN structure.
<i>ndn</i>	Normalized DN value to be set in <i>sdn</i> .

Returns

This function returns a pointer to the Slapi_DN structure containing the new normalized DN value.

See Also

- [slapi_sdn_set_dn_byval\(\)](#)
- [slapi_sdn_set_dn_passin\(\)](#)

39.33. SLAPI_SDN_SET_NDN_BYVAL()

Description

This function sets a normalized DN value in a Slapi_DN structure. The normalized DN of the new structure will point to a copy of the string pointed to by *ndn*; the normalized DN value is passed into the parameter by value. The **FLAG_DN** flag is set, and the internal counters are incremented.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_ndn_byval(Slapi_DN *sdn, const char *ndn);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	The target Slapi_DN structure.
<i>ndn</i>	The normalized DN value to be set in <i>sdn</i> .

Returns

This function returns a pointer to the Slapi_DN structure containing the new normalized DN value.

See Also

- [slapi_sdn_set_ndn_byref\(\)](#)
- [slapi_sdn_set_dn_passin\(\)](#)

39.34. SLAPI_SDN_SET_PARENT()

Description

This function sets a new parent for a given entry. This is done by keeping the RDN of the original DN of the entry and by adding the DN of its new parent (the value of *parentdn*) to it.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_parent(Slapi_DN *sdn, const Slapi_DN *parentdn);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	The Slapi_DN structure containing the DN of the entry.
<i>parentdn</i>	The new parent DN.

Returns

The function returns a pointer to the Slapi_DN structure that contains the DN of the entry after the new parent DN has been set.

See Also

- [slapi_sdn_isparent\(\)](#)
- [slapi_sdn_get_parent\(\)](#)

39.35. SLAPI_SDN_SET_RDN()

Description

This function sets a new RDN for a given entry. This is done by retrieving the DN of the entry's parent of the origin DN of the entry and then adding it to the RDN (the value of *rdn*) to it.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_rdn(Slapi_DN *sdn, const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameters:

<i>sdn</i>	The Slapi_DN structure containing the DN of the entry.
<i>rdn</i>	The new RDN.

Returns

This function returns a pointer to the Slapi_DN structure that keeps the DN of the entry after the new RDN has been set.

See Also

[slapi_sdn_get_rdn\(\)](#)

CHAPTER 40. FUNCTIONS FOR SENDING ENTRIES AND RESULTS TO THE CLIENT

This chapter contains reference information on routines for sending entries and results to the client.

Table 40.1. Routines for Sending Entries and Results to Clients

Function	Description
slapi_send_ldap_referral()	Processes an entry's LDAP v3 referrals.
slapi_send_ldap_result()	Sends an LDAP result code back to the client.
slapi_send_ldap_search_entry()	Sends an entry found by a search back to the client.

40.1. SLAPI_SEND_LDAP_REFERRAL()

Description

When you call this function, the server processes the LDAP referrals specified in the *refs* argument. The server processes referrals in different ways, depending on the version of the LDAP protocol supported by the client:

- In the LDAPv3 protocol, references to other LDAP servers (search result references) can be sent to clients as search results. For example, a server can send a mixture of entries found by the search and references to other LDAP servers as the results of a search.

This function processes an entry's LDAPv3 referrals, which are found in the entry's **ref** attribute. For LDAPv3 clients, this function sends the LDAP referrals back to the client.

When you call the [slapi_send_ldap_referral\(\)](#) function for LDAPv3 clients, the server sends the referrals specified in the *refs* argument back to the client as search result references. The **urls** argument is not used in this case.

- In the LDAPv2 protocol, servers can send the LDAP result code **LDAP_PARTIAL_RESULTS** to refer the client to other LDAP server.

For LDAPv2 clients, this function copies the referrals to an array of *berval* structures that you can pass to [slapi_send_ldap_referral\(\)](#) function at a later time.

When you call the [slapi_send_ldap_referral\(\)](#) function for LDAPv2 clients, the server collects the referrals specified in *refs* in the **urls** argument. No data is sent to the LDAPv2 client.

To get the referrals to an LDAPv2 client, you need to pass the **urls** argument (along with an **LDAP_PARTIAL_RESULTS** result code) to the [slapi_send_ldap_result\(\)](#) function. [slapi_send_ldap_result\(\)](#) concatenates the referrals specified in the **urls** argument and sends the resulting string to the client as part of the error message.

If you want to define your own function for sending referrals, write a function that complies with the type definition [send_ldap_referral_fn_ptr_t](#) and set the

`SLAPI_PLUGIN_DB_REFERRAL_FN` parameter in the parameter block to the name of your function.

Syntax

```
#include "slapi-plugin.h"
int slapi_send_ldap_referral( Slapi_PBlock *pb, Slapi_Entry *e, struct
berval **refs, struct berval ***urls );
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block.
<i>e</i>	Pointer to the Section 14.22, “Slapi_Entry” structure representing the entry with which you are working.
<i>refs</i>	Pointer to the NULL-terminated array of berval structures containing the LDAPv3 referrals (search result references) found in the entry.
<i>urls</i>	Pointer to the array of berval structures used to collect LDAP referrals for LDAPv2 clients.

Returns

This function returns one of the following values:

- 0 if successful.
- -1 if an error occurs.

See Also

- [slapi_send_ldap_result\(\)](#)
- [slapi_send_ldap_search_entry\(\)](#)

40.2. SLAPI_SEND_LDAP_RESULT()

Description

Call `slapi_send_ldap_result()` to send an LDAP result code (such as `LDAP_SUCCESS`) back to the client.

The following arguments are intended for use only in certain situations:

- `matched`

When sending an `LDAP_NO_SUCH_OBJECT` result code back to a client, use `matched` to specify how much of the target DN could be found in the database. For example, if the client was attempting to find the DN

```
cn=Babs Jensen, ou=Product Division, l=US, dc=example,dc=com
```

and the database contains entries for **c=US** and **dc=example,dc=com,c=US** but no entry for **ou=Product Division,l=US,dc=example,dc=com**, you should set the *matched* parameter to

```
l=US, dc=example,dc=com
```

- *urls*

When sending an **LDAP_PARTIAL_RESULTS** result code back to an LDAPv2 client or an **LDAP_REFERRAL** result code back to an LDAPv3 client, use *urls* to specify the referral URLs.

For LDAPv3 referrals, you can call the [slapi_str2filter\(\)](#) to send referrals to LDAPv3 clients and collect them for LDAPv2 clients. You can pass the array of collected referrals to the *urls* argument of **slapi_send_ldap_results()**. For example:

```
struct berval **urls;
...
slapi_send_ldap_referral( ld, e, &refs, &urls );
slapi_send_ldap_result( ld, LDAP_PARTIAL_RESULTS, NULL, NULL, 0, \
urls );
```

If you want to define your own function for sending result codes, write a function that complies with the type definition [send_ldap_result_fn_ptr_t](#), and set the **SLAPI_PLUGIN_DB_RESULT_FN** parameter in the parameter block to the name of your function.

Syntax

```
#include "slapi-plugin.h"
void slapi_send_ldap_result( Slapi_PBlock *pb, int err, char *matched,
char *text, int nentries, struct berval **urls );
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block.
<i>err</i>	LDAP result code that you want sent back to the client; for example, LDAP_SUCCESS .
<i>matched</i>	When sending back an LDAP_NO_SUCH_OBJECT result code, use this argument to specify the portion of the target DN that could be matched. Pass NULL in other situations.
<i>text</i>	Error message that you want sent back to the client. Pass NULL if you do not want an error message sent back.

<i>nentries</i>	When sending back the result code for an LDAP search operation, use this argument to specify the number of matching entries found.
<i>urls</i>	When sending back an LDAP_PARTIAL_RESULTS result code to an LDAPv2 client or an LDAP_REFERRAL result code to an LDAPv3 client, use this argument to specify the array of berval structures containing the referral URLs. Pass NULL in other situations.

See Also

- [slapi_str2filter\(\)](#)
- [slapi_send_ldap_search_entry\(\)](#)

40.3. SLAPI_SEND_LDAP_SEARCH_ENTRY()

Description

Call **slapi_send_ldap_search_entry()** to send an entry found by a search back to the client.

attrs is the array of attribute types that you want to send from the entry. This value is equivalent to the **SLAPI_SEARCH_ATTRS** parameter in the parameter block.

attrsonly specifies whether you want to send only the attribute types or the attribute types and their values:

- Pass **0** for this parameter if you want to send both the attribute types and values to the client.
- Pass **1** for this parameter if you want to send only the attribute types (not the attribute values) to the client.

This value is equivalent to the **SLAPI_SEARCH_ATTRSONLY** parameter in the parameter block.

If you want to define your own function for sending entries, write a function that complies with the type definition [send_ldap_search_entry_fn_ptr_t](#), and set the **SLAPI_PLUGIN_DB_ENTRY_FN** parameter in the parameter block to the name of your function.

Syntax

```
#include "slapi-plugin.h"
int slapi_send_ldap_search_entry( Slapi_PBlock *pb, Slapi_Entry *e,
LDAPControl **ectrls, char **attrs, int attrsonly );
```

Parameters

This function takes the following parameters:

<i>pb</i>	Parameter block.
<i>e</i>	Pointer to the Section 14.22, “Slapi_Entry” structure representing the entry that you want to send back to the client.
<i>ectrls</i>	Pointer to the array of LDAPControl structures representing the control associated with the search request.
<i>attrs</i>	Attribute types specified in the LDAP search request
<i>attrsonly</i>	Specifies whether the attribute values should be sent back with the result. <ul style="list-style-type: none"> • If 0, the values are included. • If 1, the values are not included.

Returns

This function returns one of the following values:

- 0 if successful.
- 1 if the entry is not sent; for example, if access control did not allow it to be sent.
- -1 if an error occurs.

See Also

- [slapi_str2filter\(\)](#)
- [slapi_send_ldap_search_entry\(\)](#)

CHAPTER 41. FUNCTIONS RELATED TO UTF-8

This chapter contains reference information on routines that are related to UTF-8.

Table 41.1. UTF-8 Related Routines

Function	Description
<code>slapi_has8thBit()</code>	Checks if a string has an 8-bit character.
<code>slapi_utf8casecmp()</code>	Makes case-insensitive string comparison of two UTF-8 strings.
<code>slapi_UTF8CASECMP()</code>	Compares two UTF-8 strings.
<code>slapi_utf8ncasecmp()</code>	Makes case-insensitive string comparison of first n characters of two UTF-8 strings.
<code>slapi_UTF8NCASECMP()</code>	Compares a specified number of UTF-8 characters.
<code>slapi_utf8isLower()</code>	Verifies if a UTF-8 character is a lower-case letter.
<code>slapi_UTF8ISLOWER()</code>	Verifies if a UTF-8 character is a lower-case letter.
<code>slapi_utf8isUpper()</code>	Verifies if a UTF-8 character is an uppercase letter.
<code>slapi_UTF8ISUPPER()</code>	Verifies if a UTF-8 character is an upper-case letter.
<code>slapi_utf8StrToLower()</code>	Converts upper-case characters in a UTF-8 string to lower-case characters.
<code>slapi_UTF8STRTOLOWER()</code>	Converts upper-case characters in a UTF-8 string to lower-case characters.
<code>slapi_utf8StrToUpper()</code>	Converts lower-case characters in a UTF-8 string to uppercase characters.
<code>slapi_UTF8STRTOUPPER()</code>	Converts lower-case characters in a UTF-8 string to upper-case characters.
<code>slapi_utf8ToLower()</code>	Convert an upper-case UTF-8 character to lower-case character.
<code>slapi_UTF8TOLOWER()</code>	Converts an upper-case UTF-8 character to lower-case character.

Function	Description
<code>slapi_utf8ToUpper()</code>	Converts a lower-case UTF-8 character to an upper-case character.
<code>slapi_UTF8TOUPPER()</code>	Converts a lower-case UTF-8 character to an upper-case character.

41.1. SLAPI_HAS8THBIT()

Checks if a string has an 8-bit character.

Syntax

```
#include "slapi-plugin.h"
int slapi_has8thBit(unsigned char *s);
```

Parameters

This function takes the following parameter:

<code>s</code>	Pointer to the null-terminated string to test.
----------------	--

Returns

This function returns one of the following values:

- 1 if the string contains an 8-bit character.
- 0 if it does not.

41.2. SLAPI_UTF8CASECMP()

Description

The function takes two UTF-8 strings (`s0`, `s1`) of **unsigned char** to be compared. The comparison rules are as follows:

- If both UTF-8 strings are NULL or zero-length, 0 is returned.
- If one of the strings is NULL or zero-length, the NULL/zero-length string is smaller.
- If one or both of the strings are not UTF-8, system provided **strcasecmp** is used.
- If one of the two strings contains no 8-bit characters, **strcasecmp** is used.
- The strings are compared after converted to lower-case UTF-8.
- Each character is compared from the beginning.

Evaluation occurs in this order:

- If the length of one character is shorter than the other, the difference of the two lengths is returned.
- If the length of the corresponding characters is the same, each byte in the characters is compared.
- If there is a difference between two bytes, the difference is returned.
- If one string is shorter than the other, the difference is returned.

Do not use this function for collation as there's no notion of locale in this function; it's UTF-8 code order, which is different from the locale-based collation.

Syntax

```
#include "slapi-plugin.h"
int slapi_utf8casecmp(unsigned char *s0, unsigned char *s1);
```

Parameters

This function takes the following parameters:

<i>s0</i>	A null-terminated UTF-8 string.
<i>s1</i>	A null-terminated UTF-8 string.

Returns

This function returns one of the following values:

- A positive number if *s0* is after *s1*.
- 0 if the two string are identical, ignoring case.
- A negative number if *s1* is after *s0*.

41.3. SLAPI_UTF8CASECMP()

Description

The function takes two UTF-8 strings (*s0*, *s1*) of **signed char** to be compared. The comparison rules are as follows:

- If both UTF-8 strings are NULL or zero-length, 0 is returned.
- If one of the strings is NULL or zero-length, the NULL/zero-lengthstring is smaller.
- If one or both of the strings are not UTF-8, system provided **strcasecmp** is used.
- If one of the two strings contains no 8-bit characters, **strcasecmp** is used.
- The strings are compared after they are converted to lower-case UTF-8.
- Each character is compared from the beginning.

Evaluation occurs in this order:

- If the length of one character is shorter than the other, the difference of the two lengths is returned.
- If the length of the corresponding characters is the same, each byte in the characters is compared.
- If there is a difference between two bytes, the difference is returned.
- If one string is shorter than the other, the difference is returned.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF8CASECMP(char *s0, char *s1);
```

Parameters

This function takes the following parameters:

<i>s0</i>	A null-terminated UTF-8 string.
<i>s1</i>	A null-terminated UTF-8 string.

Returns

This function returns one of the following values:

- A positive number if *s0* is after *s1*.
- 0 if the two string are identical, ignoring case.
- A negative number if *s1* is after *s0*.

41.4. SLAPI_UTF8NCASECMP()

Description

This function takes two UTF-8 strings (*s0*, *s1*) of unsigned char to be compared for a specified number of characters. The rules are the same as in [slapi_utf8casecmp\(\)](#) except the *n* characters limit.

Do not use this function for collation as there is no notion of locale in this function; it's UTF-8 code order, which is different from the locale-based collation. Also, the comparison is for *n* characters, not *n* bytes.

Syntax

```
#include "slapi-plugin.h"
int slapi_utf8ncasecmp(unsigned char *s0, unsigned char *s1, int n);
```

Parameters

This function takes the following parameters:

<i>s0</i>	A null-terminated UTF-8 string.
<i>s1</i>	A null-terminated UTF-8 string.
<i>n</i>	The number of UTF-8 characters (not bytes) from <i>s0</i> and <i>s1</i> to compare.

Returns

This function returns one of the following values:

- A positive number if *s0* is after *s1*.
- 0 if the two string are identical, ignoring case.
- A negative number if *s1* is after *s0*.

41.5. SLAPI_UTF8NCASECMP()

Description

Compares a specified number of UTF-8 characters. This function has the following rules:

- If both UTF-8 strings are NULL or zero-length, 0 is returned.
- If one of the strings is NULL or zero-length, the NULL/zero-length string is smaller.
- If one or both of the strings are not UTF-8, system provided **strcasecmp** is used.
- If one of the two string contains no 8-bit characters, **strcasecmp** is used.
- The strings are compared after they are converted to lower-case UTF-8.
- Each character is compared from the beginning.

Evaluation occurs in this order:

- If the length of one character is shorter then the other, the difference of the two lengths is returned.
- If the length of the corresponding characters is the same, each byte in the characters is compared.
- If there is a difference between two bytes, the difference is returned.
- If one string is shorter then the other, the difference is returned.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF8NCASECMP(char *s0, char *s1, int n);
```

Parameters

This function takes the following parameters:

<i>s0</i>	A null-terminated UTF-8 string.
<i>s1</i>	A null-terminated UTF-8 string.
<i>n</i>	The number of UTF-8 characters (not bytes) from <i>s0</i> and <i>s1</i> to compare.

Returns

This function returns one of the following values:

- A positive number if *s0* is after *s1*.
- 0 if the two string are identical, ignoring case.
- A negative number if *s1* is after *s0*.

41.6. SLAPI_UTF8ISLOWER()

Verifies if a UTF-8 character is a lower-case letter.

Syntax

```
#include "slapi-plugin.h"
int slapi_utf8isLower(unsigned char *s);
```

This function takes the following parameter:

<i>s</i>	Pointer to a single UTF-8 character (could be multiple bytes).
----------	--

Returns

This function returns one of the following values:

- 1 if the character is a lowercase letter.
- 0 if the character is not a lowercase letter.

41.7. SLAPI_UTF8ISLOWER()

Verifies if a UTF-8 character is a lower-case letter.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF8ISLOWER(char *s);
```

Parameters

This function takes the following parameter:

<i>s</i>	Pointer to a single UTF-8 character (could be multiple bytes).
----------	--

Returns

This function returns one of the following values:

- 1 if the character is a lowercase letter.
- 0 if the character is not a lowercase letter.

41.8. SLAPI_UTF8ISUPPER()

Verifies if a UTF-8 character is an upper-case letter.

Syntax

```
#include "slapi-plugin.h"
int slapi_utf8isUpper(unsigned char *s);
```

Parameters

This function takes the following parameter:

<i>s</i>	A single UTF-8 character (could be multiple bytes)
----------	--

Returns

This function returns one of the following values:

- 1 if the character is an upper-case letter.
- 0 if the character is not an upper-case letter.

41.9. SLAPI_UTF8ISUPPER()

Verifies if a UTF-8 character is an upper-case letter.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF8ISUPPER(char *s);
```

Parameters

This function takes the following parameter:

<i>s</i>	A single UTF-8 character (could be multiple bytes)
----------	--

Returns

This function returns one of the following values:

- 1 if the character is an upper case letter.
- 0 if the character is not an uppercase letter.

41.10. SLAPI_UTF8STRTOLOWER()

Description

This function converts a string of multiple UTF-8 upper-case characters to lower-case characters, not a single character as in [slapi_UTF8TOLOWER\(\)](#).

Syntax

```
#include "slapi-plugin.h"
unsigned char *slapi_utf8StrToLower(unsigned char *s);
```

Parameters

This function takes the following parameter:

<i>s</i>	A null-terminated UTF-8 string to be converted to lower case.
----------	---

Returns

This function returns one of the following values:

- A pointer to a null-terminated UTF-8 string whose characters are converted to lower case; characters which are not upper case are copied as-is.
- **NULL** if the string is not found to be a UTF-8 string.

Memory Concerns

The output string is allocated and needs to be released when it is no longer needed.

See Also

[slapi_utf8ToLower\(\)](#)

41.11. SLAPI_UTF8STRTOLOWER()

Description

This function converts a string of multiple UTF-8 upper-case characters to lower-case characters, not a single character as in [slapi_UTF8TOLOWER\(\)](#).

Syntax

```
#include "slapi-plugin.h"
unsigned char *slapi_UTF8STRTOLOWER(char *s);;
```

Parameters

This function takes the following parameter:

<code>s</code>	A null-terminated UTF-8 string to be converted to lower case.
----------------	---

Returns

This function returns one of the following values:

- A pointer to a null-terminated UTF-8 string whose characters are converted to lower case. Character which are not upper case are copied as-is.
- **NULL** if the string is not found to be a UTF-8 string.

Memory Concerns

The output string is allocated and needs to be released when no longer needed.

See Also

[slapi_UTF8TOLOWER\(\)](#)

41.12. SLAPI_UTF8STRTOUPPER()

Description

This function converts a string of multiple UTF-8 lower-case characters to upper-case characters, not a single character as in [slapi_utf8ToUpper\(\)](#).

Syntax

```
#include "slapi-plugin.h"
unsigned char *slapi_utf8StrToUpper(unsigned char *s);
```

Parameters

This function takes the following parameter:

<code>s</code>	A null-terminated UTF-8 string.
----------------	---------------------------------

Returns

This function returns one of the following values:

- A null-terminated UTF-8 string whose characters are converted to upper case; characters that are not lower case are copied as-is.
- **NULL** if the string is not considered to be a UTF-8 string.

Memory Concerns

The output string is allocated in this function and needs to be released when it is no longer used.

41.13. SLAPI_UTF8STRTOUPPER()

Converts lower-case characters in a UTF-8 string to upper-case characters.

Syntax

```
#include "slapi-plugin.h"
unsigned char *slapi_UTF8STRTOUPPER(char *s);
```

Parameters

This function takes the following parameter:

<i>s</i>	A null-terminated UTF-8 string.
----------	---------------------------------

Returns

This function returns one of the following values:

- A null-terminated UTF-8 string whose characters are converted to upper case. Character which are not lower case are copied as-is.
- **NULL** if the string is not considered to be a UTF-8 string.

Memory Concerns

The output string is allocated in this function and needs to be released when it is no longer used.

41.14. SLAPI_UTF8TOLOWER()

Converts an upper-case UTF-8 character to a lower-case character.

Syntax

```
#include "slapi-plugin.h"
void slapi_utf8ToLower(unsigned char *s, unsigned char *d, int *ssz, int *dsz);
```

Parameters

This function takes the following parameters:

<i>s</i>	A single UTF-8 character (could be multiple bytes).
<i>d</i>	Pointer to the lower case form of <i>s</i> . The memory for this must be allocated by the caller before calling the function.
<i>ssz</i>	Length in bytes of the input character.
<i>dsz</i>	Length in bytes of the output character.

Memory Concerns

Memory for the output character is not allocated in this function; caller should have allocated it (*d*). **memmove** is used since *s* and *d* are overlapped.

41.15. SLAPI_UTF8TOLOWER()

Converts an upper-case UTF-8 character to a lower-case character.

Syntax

```
#include "slapi-plugin.h"
void slapi_UTF8TOLOWER(char *s, char *d, int *ssz, int *dsz);
```

Parameters

This function takes the following parameters:

<i>s</i>	A single UTF-8 character (could be multiple bytes).
<i>d</i>	Pointer to the lower case form of <i>s</i> . The memory for this must be allocated by the caller before calling the function.
<i>ssz</i>	Returns the length in bytes of the input character.
<i>dsz</i>	Returns the length in bytes of the output character.

41.16. SLAPI_UTF8TOUPPER()

Converts a lower-case UTF-8 character to an upper-case character.

Syntax

```
#include "slapi-plugin.h"
void slapi_utf8ToUpper(unsigned char *s, unsigned char *d, int *ssz, int *dsz);
```

Parameters

This function takes the following parameters:

<i>s</i>	Pointer to a single UTF-8 character (could be multiple bytes).
<i>d</i>	Pointer to the upper case version of <i>s</i> . The memory for this must be allocated by the caller before calling the function.
<i>ssz</i>	Length in bytes of the input character.

<i>dsz</i>	Length in bytes of the output character.
------------	--

Memory Concerns

Memory for the output character is not allocated in this function; caller should have allocated it (*d*). **memmove** is used since *s* and *d* are overlapped.

41.17. SLAPI_UTF8TOUPPER()

Converts a lower-case UTF-8 character to an upper-case character.

Syntax

```
#include "slapi-plugin.h"
void slapi_UTF8TOUPPER(char *s, char *d, int *ssz, int *dsz);
```

Parameters

This function takes the following parameters:

<i>s</i>	Pointer to a single UTF-8 character (could be multiple bytes).
<i>d</i>	Pointer to the uppercase version of <i>s</i> . The memory for this must be allocated by the caller before calling the function.
<i>ssz</i>	Returns the length in bytes of the input character.
<i>dsz</i>	Returns the length in bytes of the output character.

CHAPTER 42. FUNCTIONS FOR HANDLING VALUES

This chapter contains reference information on value routines.

Table 42.1. Value Routines

Function	Description
<code>slapi_value_compare()</code>	Compares two values.
<code>slapi_value_dup()</code>	Duplicates a value.
<code>slapi_value_free()</code>	Frees a <code>Slapi_Value</code> structure from memory.
<code>slapi_value_get_berval()</code>	Gets the <code>berval</code> structure of the value.
<code>slapi_value_get_flags()</code>	Get flags from a <code>Slapi_Value</code> structure.
<code>slapi_value_get_int()</code>	Converts the value of an integer.
<code>slapi_value_get_length()</code>	Gets the length of a value.
<code>slapi_value_get_long()</code>	Converts a value into a long integer.
<code>slapi_value_get_string()</code>	Returns the value as a string.
<code>slapi_value_get_uint()</code>	Converts the value into an unsigned integer.
<code>slapi_value_get_ulong()</code>	Converts the value into an unsigned long.
<code>slapi_value_init()</code>	Initializes a <code>Slapi_Value</code> structure with no values.
<code>slapi_value_init_berval()</code>	Initializes a <code>Slapi_Value</code> structure from the <code>berval</code> structure.
<code>slapi_value_init_string()</code>	Initializes a <code>Slapi_Value</code> structure from a string.
<code>slapi_value_init_string_passin()</code>	Initializes a <code>Slapi_Value</code> structure with a value contained in a string.
<code>slapi_value_new()</code>	Allocates a new <code>Slapi_Value</code> structure.
<code>slapi_value_new_berval()</code>	Allocates a new <code>Slapi_Value</code> structure from a <code>berval</code> structure.
<code>slapi_value_new_string()</code>	Allocates a new <code>Slapi_Value</code> structure from a string.

Function	Description
<code>slapi_value_new_string_passin()</code>	Allocates a new <code>Slapi_Value</code> structure and initializes it from a string.
<code>slapi_value_new_value()</code>	Allocates a new <code>Slapi_Value</code> from another <code>Slapi_Value</code> structure.
<code>slapi_value_set()</code>	Sets the value.
<code>slapi_value_set_berval()</code>	Copies the value from a <code>berval</code> structure into a <code>Slapi_Value</code> structure.
<code>slapi_value_set_flags()</code>	Sets flags for a <code>Slapi_Value</code> structure.
<code>slapi_value_set_int()</code>	Sets the integer value of a <code>Slapi_Value</code> structure.
<code>slapi_value_set_string()</code>	Copies a string into the value.
<code>slapi_value_set_string_passin()</code>	Sets the value.
<code>slapi_value_set_value()</code>	Copies the value of a <code>Slapi_Value</code> structure into another <code>Slapi_Value</code> structure.
<code>slapi_values_set_flags()</code>	Sets flags to the array of a <code>Slapi_Value</code> structure.

42.1. SLAPI_VALUE_COMPARE()

Description

This function compares two **Slapi_Values** using the matching rule associated to the attribute *a* to determine if they are equals.

This function replaces the deprecated `slapi_attr_value_cmp()` function used in previous releases and uses the `Slapi_Value` attribute values instead of the `berval` attribute values.

Syntax

```
#include "slapi-plugin.h"
slapi_value_compare(const Slapi_Attr *a, const Slapi_Value *v1, const
Slapi_Value *v2);
```

Parameters

This function takes the following parameters:

<i>a</i>	A pointer to an attribute used to determine how the two values will be compared.
----------	--

<i>v1</i>	Pointer to the Slapi_Value structure containing the first value to compare.
<i>v2</i>	Pointer to the Slapi_Value structure containing the second value to compare.

Returns

This function returns one of the following values:

- 0 if the two values are equal.
- -1 if *v1* is smaller than *v2*.
- 1 if *v1* is greater than *v2*.

42.2. SLAPI_VALUE_DUP()

Duplicates a value.

Syntax

```
#include "slapi-plugin.h"
slapi_value_dup(const Slapi_Value *v);
```

Parameters

This function takes the following parameter:

<i>v</i>	Pointer to the Slapi_Value structure you wish to duplicate.
----------	---

Returns

This function returns a pointer to a newly allocated Slapi_Value.

Memory Concerns

The new Slapi_Value is allocated and needs to be freed by the caller, using [slapi_value_free\(\)](#).

See Also

[slapi_value_free\(\)](#)

42.3. SLAPI_VALUE_FREE()

Description

This function frees from memory the Slapi_Value structure and its members (if it is not **NULL**), and sets the pointer to **NULL**.

Syntax

```
#include "slapi-plugin.h"
slapi_value_free(Slapi_Value **value);
```

Parameters

This function takes the following parameter:

<i>value</i>	Address of the pointer to the Slapi_Value you wish to free.
--------------	---

Memory Concerns

Call this function when you are finished working with the structure.

42.4. SLAPI_VALUE_GET_BERVAL()

Gets the berval structure of the value.

Syntax

```
#include "slapi-plugin.h"
slapi_value_get_berval( const Slapi_Value *value );
```

Parameters

This function takes the following parameter:

<i>value</i>	Pointer to the Slapi_Value of which you wish to get the berval.
--------------	---

Returns

This function returns a pointer to the berval structure contained in the Slapi_Value. This function returns a pointer to the actual berval structure, not a copy of it.

Memory Concerns

You should not free the berval structure unless you plan to replace it by calling [slapi_value_set_berval\(\)](#).

See Also

[slapi_value_set_berval\(\)](#)

42.5. SLAPI_VALUE_GET_FLAGS()

This function retrieves the flags from a Slapi_Value structure.

Syntax

```
#include "slapi-plugin.h"
unsigned long slapi_value_get_flags(Slapi_Value *v);
```

Parameters

This function takes the following parameter:

<i>v</i>	Pointer to the Slapi_Value structure from which the flags are to be retrieved.
----------	---

42.6. SLAPI_VALUE_GET_INT()

Description

Converts the value in the Slapi_Value to an integer.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_get_int(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

<i>value</i>	Pointer to the Slapi_Value that you want to get as an integer.
--------------	--

Returns

This function returns one of the following values:

- An integer that corresponds to the value stored in the Slapi_Value structure.
- 0 if there is no value.

See Also

- [slapi_value_get_long\(\)](#)
- [slapi_value_get_uint\(\)](#)

42.7. SLAPI_VALUE_GET_LENGTH()

Description

This function returns the actual length of a value contained in the Slapi_Value structure.

Syntax

```
#include "slapi-plugin.h"
size_t slapi_value_get_length(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

<i>value</i>	Pointer to the Slapi_Value of which you wish to get the length.
--------------	---

Returns

This function returns one of the following values:

- The length of the value contained in Slapi_Value.
- 0 if there is no value.

42.8. SLAPI_VALUE_GET_LONG()

Description

This function converts the value contained in the Slapi_Value structure into a long integer.

Syntax

```
#include "slapi-plugin.h"
long slapi_value_get_long(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

<i>value</i>	Pointer to the Slapi_Value that you wish to get as a long integer.
--------------	--

Returns

This function returns one of the following values:

- A long integer which corresponds to the value stored in the Slapi_Value structure.
- 0 if there is no value.

See Also

- [slapi_value_get_int\(\)](#)
- [slapi_value_get_ulong\(\)](#)
- [slapi_value_get_uint\(\)](#)

42.9. SLAPI_VALUE_GET_STRING()

Returns the value as a string.

Syntax

```
#include "slapi-plugin.h"
const char*slapi_value_get_string(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

<i>value</i>	Pointer to the value you wish to get as a string.
--------------	---

Returns

This function returns one of the following values:

- A string containing the value. The function returns a pointer to the actual string value in `Slapi_Value`, not a copy of it.
- **NULL** if there is no value.

Memory Concerns

You should not free the string unless to plan to replace it by calling [slapi_value_set_string\(\)](#).

See Also

[slapi_value_set_string\(\)](#)

42.10. SLAPI_VALUE_GET_UINT()

Description

Converts the value contained in `Slapi_Value` into an unsigned integer.

Syntax

```
#include "slapi-plugin.h"
unsigned int slapi_value_get_uint(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

<i>value</i>	Pointer to the value that you wish to get as an unsigned integer.
--------------	---

Returns

This function returns one of the following values:

- An unsigned integer which corresponds to the value stored in the `Slapi_Value` structure.
- 0 if there is no value.

See Also

- [slapi_value_get_int\(\)](#)
- [slapi_value_get_long\(\)](#)

- [slapi_value_get_ulong\(\)](#)

42.11. SLAPI_VALUE_GET_ULONG()

Description

Converts the value contained in the `Slapi_Value` structure into an unsigned long integer.

Syntax

```
#include "slapi-plugin.h"
unsigned long slapi_value_get_ulong(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

<i>value</i>	Pointer to the value that you wish to get as an unsigned integer.
--------------	---

Returns

This function returns one of the following values:

- An unsigned long integer which corresponds to the value stored in the `Slapi_Value` structure.
- 0 if there is no value.

See Also

- [slapi_value_get_int\(\)](#)
- [slapi_value_get_long\(\)](#)
- [slapi_value_get_uint\(\)](#)

42.12. SLAPI_VALUE_INIT()

Description

This function initializes the `Slapi_Value` structure, resetting all of its fields to zero. The value passed as the parameter must be a valid `Slapi_Value`.

Syntax

```
#include "slapi-plugin.h"
slapi_value_init(Slapi_Value *v);
```

Parameters

This function takes the following parameter:

<i>v</i>	Pointer to the value to be initialized. The pointer must not be NULL .
----------	---

Returns

This function returns a pointer to the initialized `Slapi_Value` structure (itself).

42.13. SLAPI_VALUE_INIT_BERVAL()**Description**

This function initializes the `Slapi_Value` structure with the value contained in the `berval` structure. The content of the `berval` structure is duplicated.

Syntax

```
#include "slapi-plugin.h"
slapi_value_init_berval(Slapi_Value *v, struct berval *bval);
```

Parameters

This function takes the following parameters:

<i>v</i>	Pointer to the value to initialize. The pointer must not be NULL .
<i>bval</i>	Pointer to the <code>berval</code> structure to be used to initialize the value.

Returns

This function returns a pointer to the initialized `Slapi_Value` structure (itself).

42.14. SLAPI_VALUE_INIT_STRING()**Description**

This function initializes the `Slapi_Value` structure with the value contained in the string. The string is duplicated.

Syntax

```
#include "slapi-plugin.h"
slapi_value_init_string(Slapi_Value *v, const char *s);
```

Parameters

This function takes the following parameters:

<i>v</i>	Pointer to the value to be initialized. The pointer must not be NULL .
----------	---

s	A null-terminated string used to initialize the value.
---	--

Returns

This function returns a pointer to the initialized Slapi_Value structure (itself).

42.15. SLAPI_VALUE_INIT_STRING_PASSIN()

Description

This function initializes a Slapi_Value structure with the value contained in the string. The string is not duplicated and must be freed.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_init_string_passin (Slapi_value *v, char *s);
```

Parameters

This function takes the following parameters:

v	Pointer to the value to initialize. The pointer must not be NULL .
s	A null-terminated string used to initialize the value.

Returns

This function returns a pointer to the initialized Slapi_Value structure (itself).

Memory Concerns

The string will be freed when the Slapi_Value structure is freed from memory by calling [slapi_value_init_string_passin\(\)](#).

See Also

- [slapi_value_free\(\)](#)
- [slapi_value_new_string_passin\(\)](#)
- [slapi_value_set_string_passin\(\)](#)

42.16. SLAPI_VALUE_NEW()

Description

This function returns an empty Slapi_Value structure. You can call other functions of the API to set the value.

Syntax

```
#include "slapi-plugin.h"
slapi_value_new();
```

Parameters

This function does not take any parameters.

Returns

This function returns a pointer to the newly allocated `Slapi_Value` structure. If space cannot be allocated (for example, if no more virtual memory exists), the **slapd** program terminates.

Memory Concerns

When you are no longer using the value, free it from memory by calling [slapi_value_free\(\)](#).

See Also

- [slapi_value_dup\(\)](#)
- [slapi_value_free\(\)](#)
- [slapi_value_new_berval\(\)](#)

42.17. SLAPI_VALUE_NEW_BERVAL()

Description

This function returns a `Slapi_Value` structure containing a value duplicated from the `berval` structure passed as the parameter.

Syntax

```
#include "slapi-plugin.h"
slapi_value_new_berval(const struct berval *bval);
```

Parameters

This function takes the following parameter:

<i>bval</i>	Pointer to the <code>berval</code> structure used to initialize the newly allocated <code>Slapi_Value</code> .
-------------	--

Returns

This function returns a pointer to the newly allocated `Slapi_Value`. If space cannot be allocated (for example, if no more virtual memory exists), the **slapd** program will terminate.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling [slapi_value_free\(\)](#).

See Also

- [slapi_value_new\(\)](#)

- [slapi_value_dup\(\)](#)
- [slapi_value_free\(\)](#)
- [slapi_value_new_string\(\)](#)

42.18. SLAPI_VALUE_NEW_STRING()

Description

This function returns a Slapi_Value structure containing a value duplicated from the string passed as the parameter.

Syntax

```
#include "slapi-plugin.h"
slapi_value_new_string(const char *s);
```

Parameters

This function takes the following parameter:

<i>s</i>	A null-terminated string used to initialize the newly-allocated Slapi_Value.
----------	--

Returns

This function returns a pointer to the newly allocated Slapi_Value. If space cannot be allocated (for example, if no more virtual memory exists), the **slapd** program will terminate.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling [slapi_value_free\(\)](#).

See Also

- [slapi_value_new\(\)](#)
- [slapi_value_new_berval\(\)](#)
- [slapi_value_free\(\)](#)
- [slapi_value_dup\(\)](#)

42.19. SLAPI_VALUE_NEW_STRING_PASSIN()

Description

This function returns a Slapi_Value structure containing the string passed as the parameter. The string passed in must not be freed from memory.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new_string_passin ( char *s );
```


-

Parameters

This function takes the following parameter:

<i>s</i>	A null-terminated string used to initialize the newly-allocated <code>Slapi_Value</code> structure.
----------	---

Returns

This function returns a pointer to a newly allocated `Slapi_Value` structure. If space cannot be allocated (for example, if no virtual memory exists), the **slapd** program terminates.

Memory Concerns

The value should be freed by the caller, using [slapi_value_free\(\)](#).

See Also

- [slapi_value_free\(\)](#)
- [slapi_value_new\(\)](#)
- [slapi_value_dup\(\)](#)

42.20. SLAPI_VALUE_NEW_VALUE()**Description**

This function returns a `Slapi_Value` structure containing a value duplicated from the `Slapi_Value` structure passed as the parameter. This function is identical to [slapi_value_dup\(\)](#).

Syntax

```
#include "slapi-plugin.h"
slapi_value_new_value(const Slapi_Value *v);
```

Parameters

This function takes the following parameter:

<i>v</i>	Pointer to the <code>Slapi_Value</code> structure used to initialize the newly allocated <code>Slapi_Value</code> .
----------	---

Returns

This function returns a pointer to the newly allocated `Slapi_Value`. If space cannot be allocated (for example, if no more virtual memory exists), the **slapd** program will terminate.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling the [slapi_value_free\(\)](#) function.

See Also

- [slapi_value_dup\(\)](#)
- [slapi_value_free\(\)](#)

42.21. SLAPI_VALUE_SET()

Description

This function sets the value in the Slapi_Value structure. The value is a duplicate of the data pointed to by *val* and of the length *len*.

Syntax

```
#include "slapi-plugin.h"
slapi_value_set( Slapi_Value *value, void *val, unsigned long len);
```

Parameters

This function takes the following parameters:

<i>value</i>	Pointer to the Slapi_Value in which to set the value.
<i>val</i>	Pointer to the value.
<i>len</i>	Length of the value.

Returns

This function returns a pointer to the Slapi_Value with the valueset.

Memory Concerns

If the pointer to the Slapi_Value structure is **NULL**, then nothing is done, and the function returns **NULL**. If the Slapi_Value structure already contains a value, it is freed from memory before the new one is set.

When you are no longer using the Slapi_Value structure, you should free it from memory by calling [slapi_value_free\(\)](#).

See Also

[slapi_value_free\(\)](#)

42.22. SLAPI_VALUE_SET_BERVAL()

Description

This function sets the value of Slapi_Value structure. The value is duplicated from the berval structure *bval*.

Syntax

```
#include "slapi-plugin.h"
slapi_value_set_berval( Slapi_Value *value, const struct berval *bval );
```

Parameters

This function takes the following parameters:

<i>value</i>	Pointer to the <code>Slapi_Value</code> structure in which to set the value.
<i>bval</i>	Pointer to the <code>berval</code> value to be copied.

Returns

This function returns one of the following values:

- The pointer to the `Slapi_Value` structure passed as the parameter.
- `NULL` if it was `NULL`.

Memory Concerns

If the pointer to the `Slapi_Value` structure is `NULL`, nothing is done, and the function returns `NULL`. If the `Slapi_Value` already contains a value, it is freed from memory before the new one is set.

When you are no longer using the `Slapi_Value` structure, you should free it from memory by calling [slapi_value_free\(\)](#).

See Also

[slapi_value_free\(\)](#)

42.23. SLAPI_VALUE_SET_FLAGS()

This function sets the flags in a `Slapi_Value` structure.

This function is used to share the knowledge about the value. Currently, one flag is supported, `SLAPI_ATTR_FLAG_NORMALIZED`. With this flag, the value is already normalized.

The flag is bit-wise.

Syntax

```
#include "slapi-plugin.h"
void slapi_value_set_flags(Slapi_Value *v, unsigned long flags);
```

Parameters

This function takes the following parameter:

<i>v</i>	Pointer to the <code>Slapi_Value</code> structure for which to set the flags.
----------	--

42.24. SLAPI_VALUE_SET_INT()

Description

This function sets the value of the `Slapi_Value` structure from the integer *intVal*.

Syntax

```
#include "slapi-plugin.h"
slapi_value_set_int(Slapi_Value *value, int intVal);
```

Parameters

This function takes the following parameters:

<i>value</i>	Pointer to the <code>Slapi_Value</code> structure in which to set the integer value.
<i>intVal</i>	The integer containing the value to set.

Returns

This function returns one of the following values:

- 0 if the value is set.
- 1 if the pointer to the `Slapi_Value` is **NULL**.

Memory Concerns

If the pointer to the `Slapi_Value` structure is **NULL**, nothing is done, and the function returns -1. If the `Slapi_Value` already contains a value, it is freed from memory before the new one is set.

When you are no longer using the `Slapi_Value` structure, you should free it from memory by calling [slapi_value_free\(\)](#).

See Also

[slapi_value_free\(\)](#)

42.25. SLAPI_VALUE_SET_STRING()

Description

This function sets the value of the `Slapi_Value` structure by duplicating the string *strVal*.

Syntax

```
#include "slapi-plugin.h"
slapi_value_set_string(Slapi_Value *value, const char *strVal);
```

Parameters

This function takes the following parameters:

<i>value</i>	Pointer to the <code>Slapi_Value</code> structure in which to set the value.
--------------	--

<i>strVal</i>	The string containing the value to set.
---------------	---

Returns

This function returns one of the following:

- 0 if value is set.
- -1 if the pointer to the `Slapi_Value` is **NULL**.

Memory Concerns

If the pointer to the `Slapi_Value` is **NULL**, nothing is done, and the function returns -1. If the `Slapi_Value` already contains a value, it is freed from memory before the new one is set.

When you are no longer using the `Slapi_Value` structure, you should free it from memory by calling [slapi_value_free\(\)](#).

See Also

[slapi_value_free\(\)](#)

42.26. SLAPI_VALUE_SET_STRING_PASSIN()

Description

This function sets the value of `Slapi_Value` structure with the string *strVal*. If the `Slapi_Value` structure already contains a value, it is freed from memory before the new one is set. The string *strVal* must not be freed from memory.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_set_string_passin ( Slapi_Value *value, char *strVal);
```

Parameters

This function takes the following parameters:

<i>value</i>	Pointer to the <code>Slapi_Value</code> structure into which the value will be set.
<i>strVal</i>	The string containing the value to set.

Returns

This function returns one of the following values:

- 0 if the value is set.
- -1 if the pointer to the `Slapi_Value` structure is **NULL**.

Memory Concerns

Use `slapi_value_free()` when you are finished working with the structure to free it from memory.

42.27. SLAPI_VALUE_SET_VALUE()

Description

This function sets the value of the `Slapi_Value` structure. This value is duplicated from the `Slapi_Value` structure `vfrom`. `vfrom` must not be **NULL**.

Syntax

```
#include "slapi-plugin.h"
slapi_value_set_value( Slapi_Value *value, const Slapi_Value *vfrom);
```

Parameters

This function takes the following parameters:

<i>value</i>	Pointer to the <code>Slapi_Value</code> in which to set the value.
<i>vfrom</i>	Pointer to the <code>Slapi_Value</code> from which to get the value.

Returns

This function returns one of the following values:

- The pointer to the `Slapi_Value` structure passed as the parameter.
- **NULL** if it was **NULL**.

Memory Concerns

If the pointer to the `Slapi_Value` is **NULL**, nothing is done, and the function returns **NULL**. If the `Slapi_Value` already contains a value, it is freed from before the new one is set.

When you are no longer using the `Slapi_Value` structure, you should free it from memory by calling [slapi_value_free\(\)](#).

See Also

[slapi_value_free\(\)](#)

42.28. SLAPI_VALUES_SET_FLAGS()

This function sets the flags to an array of `Slapi_Value` structures.

slapi_values_set_flags() calls [slapi_value_set_flags\(\)](#) for each **Slapi_Value** structure in the array.

Syntax

```
#include "slapi-plugin.h"
void slapi_values_set_flags(Slapi_Value **vs, unsigned long flags);
```

Parameters

This function takes the following parameter:

v	Pointer to the Slapi_Value array for which to set the flags.
---	---

CHAPTER 43. FUNCTIONS FOR HANDLING VALUESETS

This chapter contains reference information on valueset routines.

Table 43.1. Valueset Routines

Function	Description
<code>slapi_valueset_add_value()</code>	Adds a <code>Slapi_Value</code> in the <code>Slapi_ValueSet</code> structure.
<code>slapi_valueset_add_value_ext()</code>	Enables adding of a <code>Slapi_Value</code> in the <code>Slapi_ValueSet</code> structure without having to duplicate and free the target value.
<code>slapi_valueset_count()</code>	Returns the number of values contained in a <code>Slapi_ValueSet</code> structure.
<code>slapi_valueset_done()</code>	Frees the values contained in the <code>Slapi_ValueSet</code> structure.
<code>slapi_valueset_find()</code>	Finds the value in a valueset using the syntax of an attribute.
<code>slapi_valueset_first_value()</code>	Gets the first value of a <code>Slapi_ValueSet</code> structure.
<code>slapi_valueset_free()</code>	Frees the specified <code>Slapi_ValueSet</code> structure and its members from memory.
<code>slapi_valueset_init()</code>	Resets a <code>Slapi_ValueSet</code> structure to no values.
<code>slapi_valueset_new()</code>	Allocates a new <code>Slapi_ValueSet</code> structure.
<code>slapi_valueset_next_value()</code>	Gets the next value from a <code>Slapi_ValueSet</code> structure.
<code>slapi_valueset_set_from_smod()</code>	Copies the values of <code>Slapi_Mod</code> structure into a <code>Slapi_ValueSet</code> structure.
<code>slapi_valueset_set_valueset()</code>	Initializes a <code>Slapi_ValueSet</code> structure from another <code>Slapi_ValueSet</code> structure.

43.1. SLAPI_VALUESET_ADD_VALUE()

Description

This function adds a value in the form of a `Slapi_Value` structure in a `Slapi_ValueSet` structure.

Syntax


```
#include "slapi-plugin.h"
void slapi_valueset_add_value(Slapi_ValueSet *vs, const Slapi_Value
*addval);
```

Parameters

This function takes the following parameters:

<i>vs</i>	Pointer to the <code>Slapi_ValueSet</code> structure to which to add the value.
<i>addval</i>	Pointer to the <code>Slapi_Value</code> to add to the <code>Slapi_ValueSet</code> .

Memory Concerns

The value is duplicated from the `Slapi_Value` structure, which can be freed from memory after using it without altering the `Slapi_ValueSet` structure.

This function does not verify if the value is already present in the `Slapi_ValueSet` structure. You can manually check this using [slapi_valueset_first_value\(\)](#) and [slapi_valueset_next_value\(\)](#).

See Also

- [slapi_valueset_first_value\(\)](#)
- [slapi_valueset_next_value\(\)](#)

43.2. SLAPI_VALUESET_ADD_VALUE_EXT()

Description

This function enables adding of a `Slapi_Value` in the `Slapi_ValueSet` structure without having to duplicate and free the target value. Sometimes, it is desirable to have a pass-in interface to add a `Slapi_Value` to a list without having to duplicate and free the target value. This function is similar to an existing function [slapi_valueset_add_value\(\)](#) but has one more parameter, **unsigned long flags**, for setting flags. If **SLAPI_VALUE_FLAG_PASSIN** bit is set in the flags, the function would simply take over the ownership of the new value to be added without duplicating it.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_add_value_ext(Slapi_ValueSet *vs, Slapi_Value *addval,
unsigned long flags);
```

Parameters

This function takes the following parameters:

<i>vs</i>	Pointer to the <code>Slapi_ValueSet</code> structure to which to add the value.
-----------	---

<i>addval</i>	Pointer to the Slapi_Value to add to the Slapi_ValueSet.
<i>flags</i>	If SLAPI_VALUE_FLAG_PASSIN bit is set in the flags, the function would takeover the ownership of the new value to be added without duplicating it.

See Also

- [slapi_valueset_add_value\(\)](#)
- [slapi_valueset_first_value\(\)](#)
- [slapi_valueset_next_value\(\)](#)

43.3. SLAPI_VALUESET_COUNT()

Returns the number of values contained in a Slapi_ValueSet structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_valueset_count( const Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameter:

<i>vs</i>	Pointer to the Slapi_ValueSet structure of which you wish to get the count.
-----------	---

Returns

This function returns the number of values contained in the Slapi_ValueSet structure.

43.4. SLAPI_VALUESET_DONE()

Frees the values contained in the Slapi_ValueSet structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_done(Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameter:

<i>vs</i>	Pointer to the Slapi_ValueSet structure from which you wish to free its values.
-----------	---

Memory Concerns

Use this function when you are no longer using the values but you want to re-use the `Slapi_ValueSet` structure for a new set of values.

43.5. SLAPI_VALUESET_FIND()

Description

This function finds the value in a valueset using the syntax of an attribute. Use this to check for duplicate values in an attribute.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value *slapi_valueset_find(const Slapi_Attr *a, const Slapi_ValueSet
*vs, const Slapi_Value *v);
```

Parameters

This function takes the following parameters:

<i>a</i>	Pointer to the attribute. This is used to determine the syntax of the values and how to match them.
<i>vs</i>	Pointer to the <code>Slapi_ValueSet</code> structure from which you wish to get the value.
<i>v</i>	Address of the pointer to the <code>Slapi_Value</code> structure for the returned value.

Returns

This function returns one of the following values:

- A pointer to the value in the valueset if the value was found.
- `NULL` if the value was not found.

43.6. SLAPI_VALUESET_FIRST_VALUE()

Description

Call this function when you wish to get the first value of a `Slapi_ValueSet` or you wish to iterate through all of the values. The returned value is the index of the value in the `Slapi_ValueSet` structure and must be passed to call [slapi_valueset_next_value\(\)](#) to get the next value.

Syntax

```
#include "slapi-plugin.h"
int slapi_valueset_first_value( Slapi_ValueSet *vs, Slapi_Value **v );
```

Parameters

This function takes the following parameters:

<i>vs</i>	Pointer to the Slapi_ValueSet structure from which you wish to get the value.
<i>v</i>	Address of the pointer to the Slapi_Value structure for the returned value.

Returns

This function returns one of the following values:

- The index of the value in the Slapi_ValueSet.
- -1 if there was no value.

Memory Concerns

This function gives a pointer to the actual value within the Slapi_ValueSet. You should not free it from memory.

See Also

[slapi_valueset_next_value\(\)](#)

43.7. SLAPI_VALUESET_FREE()

Description

This function frees the Slapi_ValueSet structure and its members if it is not **NULL**. Call this function when you are done working with the structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_free(Slapi_ValueSet *vs)
```

Parameters

This function takes the following parameter:

<i>vs</i>	Pointer to the Slapi_ValueSet to free.
-----------	--

See Also

[slapi_valueset_done\(\)](#)

43.8. SLAPI_VALUESET_INIT()

Description

This function returns the values contained in the `Slapi_ValueSet` structure (sets them to `0`). This does not free the values contained in the structure. To free the values, use [slapi_valueset_done\(\)](#).

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_init(Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameter:

<code>vs</code>	Pointer to the <code>Slapi_ValueSet</code> to replace.
-----------------	--

Memory Concerns

When you are no longer using the `Slapi_ValueSet` structure, you should free it from memory by using [slapi_valueset_free\(\)](#).

See Also

- [slapi_valueset_done\(\)](#)
- [slapi_valueset_free\(\)](#)

43.9. SLAPI_VALUESET_NEW()

Description

This function returns an empty `Slapi_ValueSet` structure. You can call others `slapi_valuset` functions of the API to set the values in the `Slapi_ValueSet` structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_ValueSet *slapi_valueset_new( void );
```

Parameters

This function takes no parameters.

Returns

This function returns a pointer to the newly allocated `Slapi_ValueSet` structure. If no space could be allocated (for example, if no more virtual memory exists), the **slapd** program terminates.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling [slapi_valueset_free\(\)](#).

See Also

[slapi_valueset_free\(\)](#)

43.10. SLAPI_VALUESET_NEXT_VALUE()

Description

Call this function when you wish to get the next value of a Slapi_ValueSet, after having first called [slapi_valueset_first_value\(\)](#). The returned value is the index of the value in the Slapi_ValueSet structure and must be passed to [slapi_valueset_next_value\(\)](#).

Syntax

```
#include "slapi-plugin.h"
int slapi_valueset_next_value( Slapi_ValueSet *vs, int index, Slapi_Value
**v);
```

Parameters

This function takes the following parameters:

<i>vs</i>	Pointer to the Slapi_ValueSet structure from which you wish to get the value.
<i>index</i>	Value returned by the previous call to slapi_valueset_next_value or slapi_value_first_value() .
<i>v</i>	Address to the pointer to the Slapi_Value structure for the returned value.

Returns

This function returns one of the following values:

- The index of the value in the Slapi_ValueSet.
- -1 if there was no more value or the input index is incorrect.

Memory Concerns

This function gives a pointer to the actual value within the Slapi_ValueSet and you should not free it from memory.

See Also

[slapi_valueset_first_value\(\)](#)

43.11. SLAPI_VALUESET_SET_FROM_SMOD()

Copies the values of Slapi_Mod structure into a Slapi_ValueSet structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_set_from_smod(Slapi_ValueSet *vs, Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

<i>vs</i>	Pointer to the <code>Slapi_ValueSet</code> structure into which you wish to copy the values.
<i>smod</i>	Pointer to the <code>Slapi_Mod</code> structure from which you wish to copy the values.

Description

This function copies all of the values contained in a `Slapi_Mod` structure into a `Slapi_ValueSet` structure.

Memory Concerns

This function does not verify that the `Slapi_ValueSet` structure already contains values, so it is your responsibility to verify that there are no values prior to calling this function. If you do not verify this, the allocated memory space will leak. You can free existing values by calling [slapi_valueset_done\(\)](#).

See Also

[slapi_valueset_done\(\)](#)

43.12. SLAPI_VALUESET_SET_VALUESET()

Description

This function initializes a `Slapi_ValueSet` structure by copying the values contained in another `Slapi_ValueSet` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_set_valueset(Slapi_ValueSet *vs1, const Slapi_ValueSet
*vs2);
```

Parameters

This function takes the following parameters:

<i>vs1</i>	Pointer to the <code>Slapi_ValueSet</code> structure to which you wish to set the values.
<i>vs2</i>	Pointer to the <code>Slapi_ValueSet</code> structure from which you wish to copy the values.

Memory Concerns

The function does not verify that the `Slapi_ValueSet` structure contains values, so it is your responsibility to verify that there are no values prior to calling this function. If you do not verify this, the allocated memory space will leak. You can free existing values by calling [slapi_valueset_done\(\)](#).

See Also

`slapi_valueset_done()`

CHAPTER 44. FUNCTIONS SPECIFIC TO VIRTUAL ATTRIBUTE SERVICE

This chapter contains reference information on routines that are specific to virtual attribute services.

Table 44.1. Virtual Attribute Service Routines

Function	Description
slapi_vattr_list_attrs()	Returns all the attribute types, both real and virtual, from an entry.
slapi_vattr_attrs_free()	Frees the attribute list returned by slapi_vattr_list_attrs() .
slapi_vattr_schema_check_type()	Performs a schema check on the attribute types in the entry.
slapi_vattr_value_compare()	Compares the attribute and the name in a given entry.
slapi_vattr_values_free()	Frees the attribute value and name in a given entry.
slapi_vattr_values_get()	Returns the values of a virtual attribute for the given an entry and the attribute-type name.
slapi_vattr_values_get_ex()	Returns the values for an attribute type from an entry.
slapi_vattr_values_type_thang_get()	Gets values for an attribute type in the list only if the results field for that attribute type is NULL.

44.1. SLAPI_VATTR_LIST_ATTRS()

Description

This function returns all the attribute types, both real and virtual, from an entry. You can call [slapi_vattr_values_type_thang_get\(\)](#) and take the values present in the **vattr_type_thang** list rather than calling [slapi_vattr_values_get\(\)](#) to retrieve the value.

This function should be used to return both the real and virtual attributes for an entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_list_attrs (Slapi_Entry *e, vattr_type_thang **types, int
flags, int *buffer_flags);
```

Parameters

This function takes the following parameters:

<i>e</i>	The entry of interest.
<i>types</i>	Pointer to receive the list.
<i>flags</i>	Bit mask of options. Valid values include: <ul style="list-style-type: none">• SLAPI_VIRTUALATTRS_REQUEST_PO INTERS• SLAPI_VIRTUALATTRS_ONLY SLAPI_REALATTRS_ONLY
<i>buffer_flags</i>	Bit mask of options. Valid values include: <ul style="list-style-type: none">• SLAPI_VIRTUALATTRS_RETURNED_C OPIES• SLAPI_VIRTUALATTRS_RETURNED_P OINTERS• SLAPI_VIRTUALATTRS_REALATTRS_ ONLY

Memory Concerns

The list that is returned from this API should be freed by the user by calling [slapi_vattr_attrs_free\(\)](#) for that list.

See Also

- [slapi_vattr_values_type_thang_get\(\)](#)
- [slapi_vattr_values_free\(\)](#)

44.2. SLAPI_VATTR_ATTRS_FREE()

This function should be used to free the list of attributes returned from [slapi_vattrspi_add_type\(\)](#).

Syntax

```
#include "slapi-plugin.h"
void slapi_vattr_attrs_free(vattr_type_thang **types, int flags);
```

Parameters

This function takes the following parameters:

<i>types</i>	Pointer to the list of attributes to be freed.
--------------	--

<i>flags</i>	Bit mask of options. Valid value is as follows: SLAPI_VIRTUALATTRS_RETURNED_POINTER S
--------------	---

Memory Concerns

Free the pointer block using [slapi_ch_free\(\)](#).

See Also

[slapi_vattr_list_attrs\(\)](#)

44.3. SLAPI_VATTR_SCHEMA_CHECK_TYPE()

Performs a schema check on the attribute types in the entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_schema_check_type(Slapi_Entry *e, char *type);
```

Parameters

This function takes the following parameters:

<i>e</i>	The entry to be checked.
<i>type</i>	The attribute type in the schema.

Returns

Return 0 if success, -1 if error.

44.4. SLAPI_VATTR_VALUE_COMPARE()

Description

This function compares attribute type and name in a given entry. There is no need to call [slapi_vattr_values_free\(\)](#) after calling this function.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_value_compare( Slapi_Entry *e, char *type, Slapi_Value
*test_this, int *result, int flags);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to be compared.
----------	-----------------------

<i>type</i>	Attribute type name.
<i>test_this</i>	Value to be tested.
<i>result</i>	0 if the compare is true, 1 if the compare is false.
<i>flags</i>	Not used. You should pass 0 for this parameter.

Returns

This function returns 1 for success, in which *caseresult* contains the result of the comparison. Otherwise, this function returns **0** and one of the following:

- **SLAPI_VIRTUALATTRS_LOOP_DETECTED** (failed to evaluate a *vattr*).
- **SLAPI_VIRTUAL_NOT_FOUND** (type not recognized by any *vattr* and not a real *attr* in entry).
- **ENOMEM** (memory error).

44.5. SLAPI_VATTR_VALUES_FREE()

Description

This function should be used to free the valueset and type names returned from [slapi_vattr_values_get_ex\(\)](#).

Syntax

```
#include "slapi-plugin.h"
void slapi_vattr_values_free ( Slapi_ValueSet **value, char
**actual_type_name, int flags);
```

Parameters

This function takes the following parameters:

<i>value</i>	Valueset to be freed.
<i>actual_type_name</i>	List of type names.
<i>flags</i>	The buffer flags returned from slapi_vattr_values_get_ex() . This contains information that this function needs to determine which objects need to be freed.

See Also

[slapi_vattr_values_get_ex\(\)](#)

44.6. SLAPI_VATTR_VALUES_GET()

Returns the values of a virtual attribute for the given an entry and the attribute-type name.

Syntax

```
int slapi_vattr_values_get ( Slapi_Entry *e, char *type, Slapi_ValueSet**
results,
int *type_name_disposition, char **actual_type_name, int flags,int
*buffer_flags);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to be compared.
<i>type</i>	Attribute type name.
<i>results</i>	Pointer to the result set: 0 if the compare is true, 1 if the compare is false.
<i>type_name_disposition</i>	Matching result. Valid value is as follows: SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_EXACTLY_OR_ALIAS
<i>actual_type_name</i>	Type name as found.
<i>flags</i>	Not used. You should pass 0 for this parameter.
<i>buffer_flags</i>	Bit mask of options. Valid value is as follows: SLAPI_VIRTUALATTRS_RETURNED_POINTERS

Returns

This function returns 0 for success. Otherwise, this function returns the following:

- **SLAPI_VIRTUALATTRS_LOOP_DETECTED** (failed to evaluate a *vattr*).
- **SLAPI_VIRTUAL_NOT_FOUND** (type not recognized by any *vattr* and not a real *attr* in entry).
- **ENOMEM** (memory error).

Memory Concerns

Gets values for an attribute type (**vattr_type_thang**) in the list.

See Also

[slapi_vattr_values_get_ex\(\)](#)

44.7. SLAPI_VATTR_VALUES_GET_EX()

Description

This function returns the values for an attribute type from an entry, including the values for any subtypes of the specified attribute type. The routine will return the values of virtual attributes in that entry if requested to do so.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_values_get_ex(Slapi_Entry *e, char *type,
Slapi_ValueSet*** results,
int **type_name_disposition, char ***actual_type_name, int flags, int
*buffer_flags, int *subtype_count);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry from which to get the values.
<i>type</i>	Attribute type name.
<i>results</i>	Pointer to result set.
<i>type_name_disposition</i>	Matching result.
<i>actual_type_name</i>	Type name as found.
<i>flags</i>	Bit mask of options. Valid values are as follows: <ul style="list-style-type: none">• SLAPI_REALATTRS_ONLY• SLAPI_VIRTURALATTRS_ONLY• SLAPI_VIRTUALATTRS_REQUEST_PO INTERS• SLAPI_VIRTUALATTRS_LIST_OPERA TIONAL_ATTRS
<i>buffer_flags</i>	Bit mask to be used as input flags for slapi_values_free() .
<i>subtype_count</i>	Number of subtypes matched.

Returns

This function returns 0 for success, in which case:

- *results* contains the current values for type all of the subtypes ine.

- *type_name_disposition* contains information on how each type was matched. Valid values are: **SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_EXACTLY_OR_ALIAS** and **SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_SUBTYPE**.
- *actual_type_name* contains the type name as found.
- *buffer_flags* contains the bit mask to be used as input flags for **slapi_values_free()**.
- *subtype_count* contains the number of subtypes matched.

Otherwise, this function returns the following:

- **SLAPI_VIRTUALATTRS_LOOP_DETECTED** (failed to evaluate a *vattr*).
- **SLAPI_VIRTUAL_NOT_FOUND** (type not recognized by any *vattr* and not a real *attr* in entry).
- **ENOMEM** (memory error).

Memory Concerns

[slapi_vattr_values_free\(\)](#) should be used to free the returned result set and type names, passing the *buffer_flags* value returned from this routine.

See Also

[slapi_vattr_values_free\(\)](#)

44.8. SLAPI_VATTR_VALUES_TYPE_THANG_GET()

Description

This function gets values for an attribute type in the list only if the results field for that attribute type is **NULL**. The [slapi_vattr_values_type_thang_get\(\)](#) function is faster for getting the values of an attribute when a **vattr_type_thang** list is returned from a **slapi_vattr_list_types()** call. However, when the list for that call returns **NULL**, the computation becomes similar to **slapi_vattr_values_get()**. In functionality, **slapi_vattr_values_type_thang_get()** mimics **slapi_vattr_values_get()**.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_values_type_thang_get ( Slapi_Entry *e, vattr_type_thang
    *type_thang, Slapi_ValueSet** results, int *type_name_disposition, char
    **actual_type_name, int flags, int *buffer_flags);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to be compared.
<i>type</i>	Attribute type name.

<i>results</i>	Pointer to the result set: 0 if the compare is true, 1 if the compare is false.
<i>type_name_disposition</i>	Matching result. Valid value is as follows: SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_EXACTLY_OR_ALIAS
<i>actual_type_name</i>	Type name as found.
<i>flags</i>	Not used. You should pass 0 for this parameter.
<i>buffer_flags</i>	Bit mask of options. Valid value is as follows: SLAPI_VIRTUALATTRS_RETURNED_POINTERS

See Also

[slapi_vattr_values_get\(\)](#)

CHAPTER 45. FUNCTIONS FOR MANAGING LOCKS AND SYNCHRONIZATION

This chapter contains reference information on locks and synchronization routines.

Table 45.1. Locks and Synchronization Routines

Function	Description
<code>slapi_destroy_condvar()</code>	Frees a Slapi_CondVar structure from memory.
<code>slapi_destroy_mutex()</code>	Frees a Slapi_Mutex structure from memory.
<code>slapi_lock_mutex()</code>	Locks the specified mutex.
<code>slapi_new_condvar()</code>	Creates a new condition variable and returns a pointer to the corresponding Slapi_CondVar structure.
<code>slapi_new_mutex()</code>	Creates a new mutex and returns a pointer to the corresponding Slapi_Mutex structure.
<code>slapi_notify_condvar()</code>	Notifies a thread that is waiting on the specified condition variable.
<code>slapi_unlock_mutex()</code>	Unlocks the specified mutex.
Chapter 45, Functions for Managing Locks and Synchronization	Waits on a condition variable.

45.1. SLAPI_DESTROY_CONDVAR()

This function frees a `Slapi_CondVar` structure from memory. Before calling this function, you should make sure that this condition variable is no longer in use. See [Slapi_CondVar](#).

Syntax

```
#include "slapi-plugin.h"
void slapi_destroy_condvar( Slapi_CondVar *cvar );
```

Parameters

This function takes the following parameters:

<code>cvar</code>	Pointer to the <code>Slapi_CondVar</code> structure that you want to free from memory.
-------------------	--

45.2. SLAPI_DESTROY_MUTEX()

Description

This function frees a `Slapi_Mutex` structure from memory. The calling function must ensure that no thread is currently in a lock-specific function. Locks do not provide self-referential protection against deletion. See [Slapi_Mutex](#).

Syntax

```
#include "slapi-plugin.h"
void slapi_destroy_mutex( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

<i>mutex</i>	Pointer to the <code>Slapi_Mutex</code> structure that you want to free from memory.
--------------	--

45.3. SLAPI_LOCK_MUTEX()

Description

This function locks the mutex specified by the `Slapi_Mutex` structure. After this function returns, any other thread that attempts to acquire the same lock is blocked until the holder of the lock releases the lock. Acquiring the lock is not an interruptible operation, nor is there any time-out mechanism.

Syntax

```
#include "slapi-plugin.h"
void slapi_lock_mutex( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

<i>mutex</i>	Pointer to a Slapi_Mutex structure representing the mutex that you want to lock.
--------------	--

45.4. SLAPI_NEW_CONDVAR()

Description

This function creates a new condition variable and returns a pointer to the `Slapi_CondVar` structure. You can create the `Slapi_Mutex` structure by calling the [slapi_new_mutex\(\)](#) function. See [Slapi_CondVar](#)

To wait on the condition variable, call the [slapi_wait_condvar\(\)](#) function. To notify waiting threads, call the [slapi_notify_condvar\(\)](#) function.

When you are done working with this `Slapi_CondVar` structure, call the [slapi_destroy_condvar\(\)](#) function to free the structure from memory.

Syntax

```
#include "slapi-plugin.h"
Slapi_CondVar *slapi_new_condvar( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

<i>mutex</i>	Pointer to a Slapi_Mutex structure representing the mutex that you want used to protect this condition variable.
--------------	--

Returns

This function returns one of the following values:

- A pointer to the new `Slapi_CondVar` structure.
- `NULL` if memory cannot be allocated.

45.5. SLAPI_NEW_MUTEX()

Description

This function creates a new mutex and returns a pointer to the `Slapi_Mutex` structure. You can lock this mutex by calling the [slapi_lock_mutex\(\)](#) function and unlock the mutex by calling the [slapi_unlock_mutex\(\)](#) function. See [Slapi_Mutex](#)

When you are done working with the mutex, you can free the `Slapi_Mutex` structure by calling the [slapi_destroy_mutex\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mutex *slapi_new_mutex();
```

Returns

This function returns one of the following values:

- A pointer to the new `Slapi_Mutex` structure.
- `NULL` if memory cannot be allocated.

45.6. SLAPI_NOTIFY_CONDVAR()

Description

This function notifies one or all threads that are waiting on the condition variable (see the [slapi_wait_condvar\(\)](#) function). Before calling this function, the calling thread must lock the mutex associated with this condition variable.

Syntax

```
#include "slapi-plugin.h"
int slapi_notify_condvar( Slapi_CondVar *cvar, int notify_all );
```

Parameters

This function takes the following parameters:

<i>cvar</i>	Pointer to an Slapi_CondVar structure representing the condition variable.
<i>notify_all</i>	If 1 , notifies all threads that are waiting on the condition variable.

Returns

This function returns one of the following values:

- A non-zero value if the thread (or threads) are successfully notified.
- 0 if an error occurs; for example, if the condition variable is **NULL** or if the mutex associated with the condition variable is not locked.

45.7. SLAPI_UNLOCK_MUTEX()

Description

This function unlocks the mutex specified by the [Slapi_Mutex](#) structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_unlock_mutex( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

<i>mutex</i>	Pointer to an Slapi_Mutex structure representing the mutex that you want to unlock.
--------------	---

Returns

This function returns one of the following values:

- A non-zero value if the mutex was successfully unlocked.
- 0 if the mutex was **NULL** or was not locked by the calling thread.

45.8. SLAPI_WAIT_CONDVAR()

Description

This function waits on the condition variable until it receives notification; see the [slapi_notify_condvar\(\)](#) function. Before calling this function, the calling thread must lock the mutex associated with this condition variable.

Syntax

```
#include "slapi-plugin.h"
int slapi_wait_condvar( Slapi_CondVar *cvar, struct timeval *timeout );
```

Parameters

This function takes the following parameters:

<i>cvar</i>	Pointer to a Slapi_CondVar structure representing the condition variable on which you want to wait.
<i>timeout</i>	Time period to wait for notification on the condition variable. If NULL , the calling function blocks indefinitely.

Returns

This function returns one of the following values:

- A non-zero value if successful.
- 0 if an error occurs; for example, if the condition variable is **NULL** or if the mutex associated with the condition variable is not locked.

CHAPTER 46. FUNCTIONS FOR MANAGING COMPUTED ATTRIBUTES

This chapter contains reference information on computed-attribute routines.

Table 46.1. Routines for Computed Attributes

Function	Description
<code>slapi_compute_add_evaluator()</code>	Registers a function as an evaluator that the server will call to generate a computed attribute.
<code>slapi_compute_add_search_rewriter()</code>	Registers callbacks for filter and search rewriting.
<code>compute_rewrite_search_filter()</code>	Call evaluator functions to see if there is a match with a search filter.

46.1. SLAPI_COMPUTE_ADD_EVALUATOR()

Description

The `slapi_compute_add_evaluator()` function registers a function of the `slapi_compute_callback_t` type as an evaluator of computed attributes.

Before the server sends an entry as a search result back to the client, the server determines if any of the requested attributes are computed attributes and generates the values for those attributes.

To do this, the server calls each registered evaluator function for each individually requested attribute. An evaluator function has the type `slapi_compute_callback_t`. If you want to set up the server to generate the value of a computed attribute and send the attribute back with each entry, you can define an evaluator function and register the function with the server by calling the `slapi_compute_add_evaluator()` function.

Syntax

```
#include "slapi-plugin.h"
int slapi_compute_add_evaluator( slapi_compute_callback_t function);
```

Parameters

This function takes the following parameter:

<i>function</i>	Function registered by the plug-in that will be used in evaluating the computed attributes.
-----------------	---

Returns

This function returns one of the following values:

- 0 if the function is successfully registered.
- **ENOMEM** if memory cannot be allocated to register this function.

46.2. SLAPI_COMPUTE_ADD_SEARCH_REWRITER()

Description

The `slapi_compute_add_search_rewriter()` function registers callback functions for filter searching and rewriting.

Syntax

```
#include "slapi-plugin.h"
int slapi_compute_add_search_rewriter (slapi_search_rewrite_callback_t
function);
```

Parameters

This function takes the following parameter:

<i>function</i>	Function registered by the plug-in to rewrite the search filters.
-----------------	---

Returns

This function returns one of the following values:

- -1 if no attribute matched the requested type
- 0 if one matched and it was processed without error
- >0 if an error happened

46.3. COMPUTE_REWRITE_SEARCH_FILTER()

Description

This function calls evaluator functions to see if there is a match with a search filter. Before the server sends an entry as a search result back to the client, the server determines if any of the requested attributes are computed attributes and generates the values for those attributes.

Syntax

```
#include "slapi-plugin.h"
int compute_rewrite_search_filter (Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

<i>pb</i>	Parameter block that matches the rewrite search filter.
-----------	---

Returns

This function returns one of the following values:

- -0 indicates the function should keep looking for a match.
- 0 indicates the rewrite is successful.
- 1 indicates the function refuses to perform the search.
- 2 indicates the function encountered an error.

See Also

[slapi_compute_add_evaluator\(\)](#)

CHAPTER 47. FUNCTIONS FOR MANIPULATING BITS

This chapter contains reference information on routines for manipulating bits.

Table 47.1. Bit Manipulator Routines

Function	Description
<code>slapi_isbitset_int()</code>	Checks whether a particular bit is set in an integer.
<code>slapi_isbitset_uchar()</code>	Checks whether a particular bit is set in a character.
<code>slapi_setbit_int()</code>	Sets the specified bit in an integer.
<code>slapi_setbit_uchar()</code>	Sets the specified bit in a character.
<code>slapi_unsetbit_int()</code>	Unsets the specified bit in an integer.
<code>slapi_unsetbit_uchar()</code>	Unsets the specified bit in a character.

47.1. SLAPI_ISBITSET_INT()

Checks whether a particular bit is set in the specified integer.

Syntax

```
#include "slapi-plugin.h"
int slapi_isbitset_int(unsigned int f,unsigned int bitnum);
```

Parameters

This function takes the following parameters:

<i>f</i>	The unsigned integer, a bit of which is to be checked.
<i>bitnum</i>	The bit number in the unsigned integer that needs to be checked.

Returns

This function returns one of the following values:

- 1 if the specified bit is set.
- 0 if the specified bit is not set.

See Also

- [slapi_setbit_int\(\)](#)
- [slapi_unsetbit_int\(\)](#)

47.2. SLAPI_ISBITSET_UCHAR()

Checks whether a particular bit is set in the specifier character.

Syntax

```
#include "slapi-plugin.h"
int slapi_isbitset_uchar(unsigned char f,unsigned char bitnum);
```

Parameters

This function takes the following parameters:

<i>f</i>	The unsigned character, a bit of which is to be checked.
<i>bitnum</i>	The bit number in the unsigned character that needs to be checked.

Returns

This function returns one of the following values:

- 1 if the specified bit is set.
- 0 if the specified bit is not set.

See Also

- [slapi_setbit_uchar\(\)](#)
- [slapi_unsetbit_uchar\(\)](#)

47.3. SLAPI_SETBIT_INT()

Sets the specified bit in an integer.

Syntax

```
#include "slapi-plugin.h"
unsigned int slapi_setbit_int(unsigned int f,unsigned int bitnum);
```

Parameters

This function takes the following parameters:

<i>f</i>	The integer in which a bit is to be set.
----------	--

<i>bitnum</i>	The bit number that needs to be set in the integer.
---------------	---

Returns

This function returns the integer with the specified bit set.

See Also

- [slapi_isbitset_int\(\)](#)
- [slapi_unsetbit_int\(\)](#)

47.4. SLAPI_SETBIT_UCHAR()

Sets the specified bit in a character.

Syntax

```
#include "slapi-plugin.h"
unsigned char slapi_setbit_uchar(unsigned char f, unsigned char bitnum);
```

Parameters

This function takes the following parameters:

<i>f</i>	The character in which a bit is to be set.
<i>bitnum</i>	The bit number that needs to be set in the character.

Returns

This function returns the character with the specified bit set.

See Also

- [slapi_isbitset_uchar\(\)](#)
- [slapi_unsetbit_uchar\(\)](#)

47.5. SLAPI_UNSETBIT_INT()

Unsets the specified bit in an integer.

Syntax

```
#include "slapi-plugin.h"
unsigned int slapi_unsetbit_int(unsigned int f, unsigned int bitnum);
```

Parameters

This function takes the following parameters:

<i>f</i>	The integer in which a bit is to be unset.
<i>bitnum</i>	The bit number that needs to be unset in the integer.

Returns

This function returns the integer with the specified bit unset.

See Also

- [slapi_isbitset_int\(\)](#)
- [slapi_setbit_int\(\)](#)

47.6. SLAPI_UNSETBIT_UCHAR()

Unsets the specified bit in a character.

Syntax

```
#include "slapi-plugin.h"
unsigned char slapi_unsetbit_uchar(unsigned char f, unsigned char bitnum);
```

Parameters

This function takes the following parameters:

<i>f</i>	The character in which a bit is to be unset.
<i>bitnum</i>	The bit number that needs to be unset in the character.

Returns

This function returns the character with the specified bit unset.

See Also

- [slapi_isbitset_uchar\(\)](#)
- [slapi_setbit_uchar\(\)](#)

CHAPTER 48. FUNCTIONS FOR REGISTERING AND UNREGISTERING OBJECT EXTENSIONS

This chapter contains reference information on routines for registering object extensions. This set of functions provides a means for extending core server objects; this functionality is provided so that you can efficiently pass state information between plug-in calls. Typically, a plug-in might register both a pre-operation and post-operation call. It is very convenient for the plug-in to associate its private data with the operation object that is passed through the parameter block.

Table 48.1. Routines for Registering Object Extensions

Function	Description
slapi_get_object_extension()	Retrieves a pointer to the plug-in extension.
slapi_register_object_extension()	Registers a plug-in's object extension.
slapi_set_object_extension()	Changes a plug-in's object extension.

48.1. SLAPI_GET_OBJECT_EXTENSION()

Description

A plug-in retrieves a pointer to its own extension by calling **slapi_get_object_extension** with the object from which the extension is to be retrieved. The factory uses **objecttype** to find the offset into the object where the extension block is stored. The extension handle is then used to find the appropriate extension within the block.

Syntax

```
#include "slapi-plugin.h"
void *slapi_get_object_extension(int objecttype, void *object, int
extensionhandle);
```

Parameters

This function takes the following parameters:

<i>objecttype</i>	The object type handle that was returned from the slapi_register_object_extension() call.
<i>object</i>	A pointer to the core server object from which the extension is to be retrieved.
<i>extensionhandle</i>	The extension handle that was returned from the slapi_register_object_extension() call.

Returns

This function returns a pointer to the plug-in's extension.

See Also

- [slapi_register_object_extension\(\)](#)
- [slapi_set_object_extension\(\)](#)

48.2. SLAPI_REGISTER_OBJECT_EXTENSION()

Description

When a plug-in is initialized, it must register its object extensions. It must provide the name of the object to be extended, e.g., Operation, and constructor and destructor functions. These functions are called when the object is constructed and destroyed. The extension functions must allocate some memory and initialize it for its own use. The registration function will fail if any objects have already been created; this is why the registration must happen during plug-in initialization. In return, the plug-in will receive two handles, one for the object type and another one for the object extension; these only have meaning for the [slapi_get_object_extension\(\)](#) function.

Syntax

```
#include "slapi-plugin.h"
int slapi_register_object_extension( const char *pluginname, const char
    *objectname,
    slapi_extension_constructor_fnptr constructor,
    slapi_extension_destructor_fnptr destructor,
    int *objecttype, int *extensionhandle);
```

Parameters

This function takes the following parameters:

<i>pluginname</i>	Plug-in name.
<i>objectname</i>	<div>The name of the core server object to be extended. Objects that can be extended (possible values for the <i>objectname</i> parameter):<ul style="list-style-type: none">• SLAPI_EXT_CONNECTION (Connection)• SLAPI_EXT_OPERATION (Operation)• SLAPI_EXT_ENTRY (Entry)• SLAPI_EXT_MTNODE (Mapping Tree Node)Currently, only Operation and Connection are supported.</div>

<i>constructor</i>	The function provided by the plug-in which is to be called when an instance of the core server object is created. This function must allocate some memory and return a pointer to be stored in the extension block on the object.
<i>destructor</i>	The function which is called when an instance of an object is destroyed. This function must release any resources acquired by the constructor function.
<i>objecttype</i>	Handle to find the offset into the object where the extension block is stored.
<i>extensionhandle</i>	Address, which is used to find the extension within the block.

Returns

This function returns 0 if successful, error code otherwise.

See Also

- [slapi_get_object_extension\(\)](#)
- [slapi_set_object_extension\(\)](#)

48.3. SLAPI_SET_OBJECT_EXTENSION()

Description

This function enables a plug-in to change its extensions.

Syntax

```
#include "slapi-plugin.h"
void slapi_set_object_extension(int objecttype, void *object, int
extensionhandle, void *extension);
```

Parameters

This function takes the following parameters:

<i>objecttype</i>	Handle to find the offset into the object where the extension block is stored.
<i>object</i>	A pointer to the core server object that needs to be changed.
<i>extensionhandle</i>	Address for finding the extension within the block.

<i>extension</i>	Pointer to the extension block.
------------------	---------------------------------

See Also

- [slapi_register_object_extension\(\)](#)
- [slapi_get_object_extension\(\)](#)

48.4. SLAPI_UNREGISTER_OBJECT_EXTENSION()

Description

This function unregisters objects. Note that plug-ins that register objects must unregister them in the close function.

Syntax

```
int slapi_unregister_object_extension(  
    const char* pluginname,  
    const char* objectname,  
    int *objecttype,  
    int *ext_handle);
```

Parameters

<i>pluginname</i>	The name of the plug-in.
<i>objectname</i>	<p>The name of the core server object to be extended. Objects that can be extended (possible values for the <i>objectname</i> parameter):</p> <ul style="list-style-type: none">• SLAPI_EXT_CONNECTION (Connection)• SLAPI_EXT_OPERATION (Operation)• SLAPI_EXT_ENTRY (Entry)• SLAPI_EXT_MTNODE (Mapping Tree Node) <p>Currently, only Operation and Connection are supported.</p>
<i>objecttype</i>	Handle to find the offset into the object where the extension block is stored.
<i>ext_handle</i>	Address for finding the extension within the block.

CHAPTER 49. FUNCTIONS RELATED TO DATA INTEROPERABILITY

This chapter contains reference information on routines that support the data interoperability feature of Directory Server, which is explained in [Chapter 13, Using Data Interoperability Plug-ins](#). This set of functions allows a custom plug-in to preserve the default behavior of the Directory Server and bypass access control checking.

Table 49.1. Routines Related to Data Interoperability

Function	Description
slapi_op_reserved()	Allows a plug-in to recognize reserved default operations of the Directory Server.
slapi_operation_set_flag()	Sets a flag for the operation.
slapi_operation_clear_flag()	Clears a flag for the operation.
slapi_operation_is_flag_set()	Determines whether a flag is set in the operation.

49.1. SLAPI_OP_RESERVED()

Description

This function allows a plug-in to recognize reserved default operations, such as the base-scope search on the root dse and the operations on the reserved naming contexts, for handling by the core Directory Server and not by the DIOP plug-in. This function allows you to implement a custom DIOP plug-in that does not affect the default behavior of the server. The code snippet below is a sample for a plug-in that handles the LDAP delete operation. The callback for the LDAP delete operation **nullsuffix_delete** will ignore all the LDAP delete operations on the reserved-naming contexts (such as **cn=schema**, **cn=config**, and **cn=monitor**).

```
#define PLUGIN_OPERATION_HANDLED 0
#define PLUGIN_OPERATION_IGNORED 1

static int nullsuffix_delete( Slapi_PBlock *pb )
{
    if( slapi_op_reserved(pb) ){
        return PLUGIN_OPERATION_IGNORED;
    }
    slapi_log_error( SLAPI_LOG_PLUGIN, PLUGIN_NAME,
        "nullsuffix_delete\n" );
    /* do the deletes */
    send_ldap_result( pb, LDAP_SUCCESS, NULL, NULL, 0, NULL );
    return PLUGIN_OPERATION_HANDLED;
}
```

Syntax

```
#include "slapi-plugin.h"
int slapi_op_reserved(Slapi_PBlock *pb);
```

Parameter

This function takes the following parameter:

<i>pb</i>	Parameter block.
-----------	------------------

Returns

This function returns 0 if the operation is not reserved and a non-zero value if the operation is reserved.

49.2. SLAPI_OPERATION_SET_FLAG()

Description

This function sets the specified flag for the operation. The code sample demonstrates how the flag for the operation is to be set:

```
Slapi_Operation *op;
if ( slapi_pblock_get( pb, SLAPI_OPERATION, &op ) != 0 ) {
    slapi_operation_set_flag( op, SLAPI_OP_FLAG_NO_ACCESS_CHECK );
}
```

Syntax

```
#include "slapi-plugin.h"
void slapi_operation_set_flag( Slapi_Operation *op, unsigned long flag)
```

Parameter

This function takes the following parameters:

<i>op</i>	Operation data structure.
<i>flag</i>	Flag to be set. By default, only one flag is supported, the SLAPI_OP_FLAG_NO_ACCESS_CHECK flag, which specifies that access control should not be checked.

See Also

- [slapi_operation_clear_flag\(\)](#)
- [slapi_operation_is_flag_set\(\)](#)

49.3. SLAPI_OPERATION_CLEAR_FLAG()

Clears the specified flag for the operation.

Syntax

```
#include "slapi-plugin.h"
void slapi_operation_clear_flag( Slapi_Operation *op, unsigned long flag)
```

Parameter

This function takes the following parameters:

<i>op</i>	Operation data structure.
<i>flag</i>	Flag to be cleared. By default, only one flag is supported, the SLAPI_OP_FLAG_NO_ACCESS_CHECK flag, which specifies that access control should not be checked.

See Also

- [slapi_operation_set_flag\(\)](#)
- [slapi_operation_is_flag_set\(\)](#)

49.4. SLAPI_OPERATION_IS_FLAG_SET()

Description

This function determines whether the specified flag is set in the operation. The code sample below demonstrates how ACL checks for internal operations are skipped if the plug-in specifies to not check for access control.

```
if (operation_is_flag_set(operation, SLAPI_OP_FLAG_NO_ACCESS_CHECK))
return LDAP_SUCCESS;
// Success indicates that access is allowed.
```

Syntax

```
#include "slapi-plugin.h"
int slapi_operation_is_flag_set(Slapi_Operation *op, unsigned long flag)
```

Parameter

This function takes the following parameters:

<i>op</i>	Operation data structure.
-----------	---------------------------

<i>flag</i>	<p>Flag to check. There are three possible flags:</p> <div><p>SLAPI_OP_FLAG_NO_ACCESS_CHECK flag, which specifies that access control should not be checked.</p><p>SLAPI_OPERATION_FLAG_INTERNAL flag, which is set for any internal operation.</p><p>SLAPI_OP_FLAG_NEVER_CHAIN flag, which specifies that disables chaining.</p></div>
-------------	--

Returns

This function returns 0 if the flag is not set and a non-zero value if the flag is set.

See Also

- [slapi_operation_set_flag\(\)](#)
- [slapi_operation_clear_flag\(\)](#)

CHAPTER 50. FUNCTIONS FOR REGISTERING ADDITIONAL PLUG-INS

This chapter contains reference information on interfaces that allow a plug-in to register additional plug-ins.

Table 50.1. Routines for Registering Additional Plug-ins

Function	Description
<code>slapi_register_plugin()</code>	Allows a plug-in to register a plug-in.

50.1. SLAPI_REGISTER_PLUGIN()

Description

This function allows a plug-in to register a plug-in. This was added so that an object plug-in can register all the plug-in interfaces that it supports, including legacy plug-ins.

Syntax

```
#include "slapi-plugin.h"
int slapi_register_plugin( const char *plugintype, int enabled, const char
*initsymbol,
slapi_plugin_init_fnptr initfunc, const char *name, char **argv, void
*group_identity);
```

Parameters

This function takes the following parameters:

<i>plugintype</i>	Handle to find the offset into the object where the extension block is stored.
<i>enabled</i>	A pointer to the core server object that needs to be changed.
<i>initsymbol</i>	Address for finding the extension within the block.
<i>initfunc</i>	Pointer to the extension block.
<i>name</i>	Pointer to the name of the plug-in.
<i>argv</i>	Pointer to an array of the plug-ins.
<i>group_identity</i>	Group identity of the plug-ins.

CHAPTER 51. FUNCTIONS FOR SERVER TASKS

Directory Server tasks are usually maintenance operations for the server, such as indexing entries or importing an LDIF file.

Certain server tasks can be performed through a special task plug-in. When a task entry is added to the **cn=tasks** subtree in the directory, the task's handler is called and performs the task.

This chapter describes functions available for task structures.

Table 51.1. Routines for Task Plug-ins

Function	Description
<code>slapi_destroy_task()</code>	Frees a Slapi_Task structure when the task is completed.
<code>slapi_plugin_new_task()</code>	Creates a new server task and returns a pointer to the Slapi_Task structure.
<code>slapi_task_begin()</code>	Updates a task entry to indicate that the task is running.
<code>slapi_task_cancel()</code>	Cancels a current task.
<code>slapi_task_dec_refcount()</code>	Decrements the task reference count.
<code>slapi_task_finish()</code>	Called when the process is complete to write to the task entry and to return the result code.
<code>slapi_task_get_data()</code>	Retrieves an opaque data pointer from the task.
<code>slapi_task_get_refcount()</code>	Checks the current reference count of the task. If a task has multiple threads, this shows whether the individual tasks have completed.
<code>slapi_task_get_state()</code>	Shows the current state of the task.
<code>slapi_task_inc_progress()</code>	Automatically increments the task progress, which updates the task entry.
<code>slapi_task_inc_refcount()</code>	Increments the task reference count, if the task uses multiple threads.
<code>slapi_task_log_notice()</code>	Writes changes to a log attribute for the task entry.

Function	Description
slapi_task_log_status()	Updates the task status attribute in the entry to maintain a running display of the task status.
slapi_plugin_task_register_handler()	Registers a task handler function.
slapi_plugin_task_unregister_handler()	Unregisters a plug-in task.
slapi_task_set_cancel_fn()	Sets a callback to be used when a task is canceled.
slapi_task_set_data()	Appends an opaque object pointer to the task process.
slapi_task_set_destructor_fn()	Sets a callback to be used when a task is destroyed.
slapi_task_status_changed()	Contains the task's status.

51.1. SLAPI_DESTROY_TASK()

Frees a **Slapi_Task** structure. This does not update the task status, so this function should only be used to free a task that was not started processing with [slapi_task_begin\(\)](#). For tasks were started processing with [slapi_task_begin\(\)](#), [slapi_task_finish\(\)](#) or [slapi_task_cancel\(\)](#) to update the tasks status and free the task structure.

This function calls any custom destructor callback functions that have been set, so it is safe to use this function even if some private task data are set with [slapi_task_set_data\(\)](#).

Syntax

```
void slapi_destroy_task(void *arg)
```

Parameters

This function takes the following parameter:

<i>arg</i>	Contains arguments to passed to the task.
------------	---

51.2. SLAPI_PLUGIN_NEW_TASK()

Creates a new server task and returns a pointer to the **Slapi_Task** structure.

Syntax

```
Slapi_Task *slapi_plugin_new_task(const char *dn, void *arg)
```

Parameters

This function takes the following parameters:

<i>dn</i>	The DN of the task entry that was defined for this instance of the task. This DN is used by the task process to identify the appropriate task configuration entry so the process can write log and status messages.
<i>arg</i>	Represents the Slapi_Pblock used when starting the plug-in.

51.3. SLAPI_TASK_BEGIN()

Updates the task entry state to indicate that the task is running. Call this as soon as necessary to begin performing the work for the custom task.

Syntax

```
void slapi_task_begin(Slapi_Task *task, int total_work)
```

Parameters

This function takes the following parameters:

<i>task</i>	Points to the task operation which is being performed by the server.
<i>total_work</i>	Gives the total number of work items needed to complete the task. This can be used to update the task status with partial progress as the task runs.

51.4. SLAPI_TASK_CANCEL()

Called to cancel a task. This function updates the state attributes in the task configuration entry to record that the task is canceled and returns a result code to the user.

slapi_task_cancel() queues an event to destroy the task, which frees the task itself as well as any cleanup defined in a custom destructor callback function.

Syntax

```
void slapi_task_cancel(Slapi_Task *task, int rc)
```

Parameters

This function takes the following parameters:

<i>task</i>	Points to the task operation which is being performed by the server.
-------------	--

<i>rc</i>	Contains the result code to set for the task.
-----------	---

See Also

- [slapi_task_finish\(\)](#)

51.5. SLAPI_TASK_DEC_REFCOUNT()

Decrements the task reference count to prevent freeing any shared data before the threads are finished with it.

Reference counts are used by tasks which initiate multiple threads.

Syntax

```
void slapi_task_dec_refcount(Slapi_Task *task)
```

Parameters

This function takes the following parameter:

<i>task</i>	Points to the task operation which is being performed by the server.
-------------	--

See Also

- [slapi_task_inc_refcount\(\)](#)
- [slapi_task_get_refcount\(\)](#)

51.6. SLAPI_TASK_FINISH()

Called when the task is completed. As with other functions which terminate a task, this updates the task entry to record that the task is complete and returns a result code to the user. This function queues an event to destroy the task, which frees the task itself as well as any cleanup defined in a custom destructor callback function.

Syntax

```
void slapi_task_finish(Slapi_Task *task, int rc)
```

Parameters

This function takes the following parameters:

<i>task</i>	Points to the task operation which is being performed by the server.
<i>rc</i>	Contains the result code for the task.

See Also

- [slapi_task_cancel\(\)](#)

51.7. SLAPI_TASK_GET_DATA()

Retrieves an opaque data pointer from a task.

Syntax

```
void * slapi_task_get_data(Slapi_Task *task)
```

Parameters

This function takes the following parameter:

<i>task</i>	Points to the task operation which is being performed by the server.
-------------	--

Memory Concerns

This function returns the pointer that was passed into [slapi_task_set_data\(\)](#). Most of the time, do not free this memory directly. It is recommended that a destructor callback function, such as [slapi_task_set_destructor_fn\(\)](#), be specified after creating the task. This destructor callback function is then responsible for cleaning up the task data.

See Also

- [slapi_task_set_data\(\)](#)
- [slapi_task_set_destructor_fn\(\)](#)
- [TaskCallbackFn](#)

51.8. SLAPI_TASK_GET_REFCOUNT()

Checks the current reference count of the task. Reference counts are used by tasks that open multiple threads; **slapi_task_get_refcount()** checks to see if any or all of the threads have finished and allows any shared data to be freed safely.

Syntax

```
int slapi_task_get_refcount(Slapi_Task *task)
```

Parameters

This function takes the following parameter:

<i>task</i>	Points to the task operation which is being performed by the server.
-------------	--

See Also

- [slapi_task_dec_refcount\(\)](#)
- [slapi_task_inc_refcount\(\)](#)

51.9. SLAPI_TASK_GET_STATE()

Checks the current state of a task.

Syntax

```
int slapi_task_get_state(Slapi_Task *task)
```

Parameters

This function takes the following parameter:

<i>task</i>	Points to the task operation which is being performed by the server.
-------------	--

Returns

This can return four different states:

- **SLAPI_TASK_SETUP (0)**, meaning the task has been initiated
- **SLAPI_TASK_RUNNING (1)**, meaning the task is in progress
- **SLAPI_TASK_FINISHED (2)**, meaning the task completed (there is a separate return code to indicate whether the task was successful)
- **SLAPI_TASK_CANCELLED (3)**, meaning the task was stopped before it was completed

51.10. SLAPI_TASK_INC_PROGRESS()

Increments the task progress by one (1) work unit and updates the task status in the entry.

Incrementing the progress allows the task to report partial progress states to the user. The total number of work units needed to complete the task are set when [slapi_task_begin\(\)](#) is called.

Syntax

```
void slapi_task_inc_progress(Slapi_Task *task)
```

Parameters

This function takes the following parameter:

<i>task</i>	Points to the task operation which is being performed by the server.
-------------	--

See Also

- [slapi_task_log_status\(\)](#)
- [slapi_task_status_changed\(\)](#)
- [slapi_task_begin\(\)](#)

51.11. SLAPI_TASK_INC_REFCOUNT()

Increments the task reference count.

Reference counts are used by tasks which initiate multiple threads. Using **slapi_task_inc_refcount()** ensures that all threads are done before freeing any shared data.

Syntax

```
void slapi_task_inc_refcount(Slapi_Task *task)
```

Parameters

This function takes the following parameter:

<i>task</i>	Points to the task operation which is being performed by the server.
-------------	--

See Also

- [slapi_task_dec_refcount\(\)](#)
- [slapi_task_get_refcount\(\)](#)

51.12. SLAPI_TASK_LOG_NOTICE()

Writes changes to a log attribute for the task entry.

Directory task entries log all of their messages to a log attribute in the configuration entry, **nsTaskLog**. This function adds a line to the **nsTaskLog** value. This value is cumulative, so any new message logged is appended to the existing value.

Syntax

```
void slapi_task_log_notice(Slapi_Task *task, char *format, ...)
```

Parameters

This function takes the following parameters:

<i>task</i>	Points to the task operation which is being performed by the server.
<i>format</i>	A printf -style format string.

See Also

- [slapi_task_log_status\(\)](#)

51.13. SLAPI_TASK_LOG_STATUS()

Updates the status attribute for the task entry.

Directory task entries record their status in the *nsTaskStatus* attribute. This function generates the value for *nsTaskStatus*. Unlike the *nsTaskLog* attribute, this value is overwritten whenever it is updated.

Syntax

```
void slapi_task_log_status(Slapi_Task *task, char *format, ...)
```

Parameters

This function takes the following parameters:

<i>task</i>	Points to the task operation which is being performed by the server.
<i>format</i>	A printf -style format string.

See Also

- [slapi_task_log_notice\(\)](#)

51.14. SLAPI_PLUGIN_TASK_REGISTER_HANDLER()

Registers a new task handler function.

The *name* parameter passed in the function the value of the *cn* attribute of the task container for the new task for which the plug-in is being written. The handler function is called when a new task entry is created inside of the container entry matching the *name* parameter.

Typically, call this from the plug-in start function.

Syntax

```
int slapi_plugin_task_register_handler(const char *name, dseCallbackFn  
func, Slapi_PBlock *plugin_pb)
```

Parameters

This function takes the following parameter:

<i>name</i>	Gives the name of the task handler.
-------------	-------------------------------------

<i>plugin_pb</i>	This is the Slapi_Pblock that is used to initialize the plug-in in the start function of the plug-in. This parameter is used to properly track the task, so that it can be removed later after the plug-in is stopped or closed.
------------------	---

See Also

- [dseCallbackFn](#)

51.15. SLAPI_PLUGIN_TASK_UNREGISTER_HANDLER()

Unregisters a task handler function.

Call `slapi_plugin_task_unregister_handler()` in the plug-in close function.

Syntax

```
int slapi_plugin_task_unregister_handler(const char *name, dseCallbackFn func)
```

Parameters

<i>name</i>	Gives the name of the task handler.
-------------	-------------------------------------

51.16. SLAPI_TASK_SET_DATA()

Adds an opaque data object pointer to your task.

This function is useful for including data-specific information that the handler function can assign to the thread that actually performs the task.

Syntax

```
void slapi_task_set_data(Slapi_Task *task, void *data)
```

Parameters

This function takes the following parameters:

<i>task</i>	Points to the task operation which is being performed by the server.
<i>data</i>	Specific data to use to perform the operation.

Memory Concerns

Whatever is added to the task must be allocated by the caller. To free these data, set a destructor callback function using the [slapi_task_set_destructor_fn\(\)](#) function.

See Also

- [slapi_task_get_data\(\)](#)
- [slapi_task_set_destructor_fn\(\)](#)
- [TaskCallbackFn](#)

51.17. SLAPI_TASK_SET_CANCEL_FN()

Sets a callback to be used when a task is cancelled.

The destruction of the task itself is taken care of automatically, but a callback can be set to perform specific operations related to canceling a task. Any custom destructor function is called after the cancel function.

Syntax

```
void slapi_task_set_cancel_fn(Slapi_Task *task, TaskCallbackFn func)
```

Parameters

This function takes the following parameter:

<i>task</i>	Points to the task operation which is being performed by the server.
-------------	--

See Also

- [slapi_task_set_destructor_fn\(\)](#)
- [TaskCallbackFn](#)

51.18. SLAPI_TASK_SET_DESTRUCTOR_FN()

Sets a callback to be used when a task is destroyed.

The destruction of the task itself is taken care of automatically, but a callback can be set to clean up any data that were passed with [slapi_task_set_data\(\)](#).

Syntax

```
void slapi_task_set_destructor_fn(Slapi_Task *task, TaskCallbackFn func)
```

Parameters

This function takes the following parameter:

<i>task</i>	Points to the task operation which is being performed by the server.
-------------	--

See Also

- [slapi_task_set_data\(\)](#)

- [TaskCallbackFn](#)

51.19. SLAPI_TASK_STATUS_CHANGED()

Updates the task configuration entry with the current status information.

This function *must* be called if the progress or status of the task is updated manually in the **Slapi_Task** structure. However, **slapi_task_status_changed()** is called by the other **slapi_task_*** functions that manage the task status automatically, so it is not necessary to call **slapi_task_status_changed()** if the plug-in is written using the **Slapi_Task** API.

Syntax

```
void slapi_task_status_changed(Slapi_Task *task)
```

Parameters

This function takes the following parameter:

<i>task</i>	Points to the task operation which is being performed by the server.
-------------	--

See Also

- [slapi_task_inc_progress\(\)](#)
- [slapi_task_log_notice\(\)](#)
- [slapi_task_log_status\(\)](#)

PART V. PARAMETER BLOCK REFERENCE

This part describes the parameters available in the [Slapi_PBlock](#) parameter block, the type of data associated with each parameter, and the plug-in functions in which those parameters are accessible. To get the values of these parameters, call the [slapi_pblock_get\(\)](#) function. To set the values of these parameters, call the [slapi_pblock_set\(\)](#) function.

CHAPTER 52. PARAMETERS FOR REGISTERING PLUG-IN FUNCTIONS

The parameters listed in this section identify plug-in functions recognized by the server. To register your plug-in function, set the value of the appropriate parameter to the name of your function.



NOTE

With the exception of the parameters for matching rule plug-in functions, you do not need to get the value of any of these parameters.



NOTE

Database plug-ins are not supported in current releases of Directory Server. Please use the pre-operation, post-operation, and/or extended operation API to register plug-in functions.

52.1. PRE-OPERATION/DATA VALIDATION PLUG-INS

The parameters listed in this section are used to register pre-operation/data validation plug-in functions.

To register your plug-in function, write an initialization function that sets the values of the following parameters to your functions.

Parameter ID	Description
SLAPI_PLUGIN_PRE_BIND_FN	This function is called before an LDAP bind operation is completed.
SLAPI_PLUGIN_PRE_UNBIND_FN	This function is called before an LDAP unbind operation is completed.
SLAPI_PLUGIN_PRE_SEARCH_FN	This function is called before an LDAP search operation is completed.
SLAPI_PLUGIN_PRE_COMPARE_FN	This function is called before an LDAP compare operation is completed.
SLAPI_PLUGIN_PRE_MODIFY_FN	This function is called before an LDAP modify operation is completed.
SLAPI_PLUGIN_PRE_MODRDN_FN	This function is called before an LDAP modify RDN operation is completed.
SLAPI_PLUGIN_PRE_ADD_FN	This function is called before an LDAP add operation is completed.

Parameter ID	Description
SLAPI_PLUGIN_PRE_DELETE_FN	This function is called before an LDAP delete operation is completed.
SLAPI_PLUGIN_PRE_ENTRY_FN	This function is called before an entry is sent back to the client.
SLAPI_PLUGIN_PRE_REFERRAL_FN	This function is called before a set of referrals is sent back to the client.
SLAPI_PLUGIN_PRE_RESULT_FN	This function is called before a set of search results is sent back to the client.
SLAPI_PLUGIN_START_FN	This function is called after the server startup. You can specify a start function for each pre-operation plug-in.
SLAPI_PLUGIN_CLOSE_FN	This function is called before the server shuts down. You can specify a close function for each pre-operation plug-in.
SLAPI_PLUGIN_DESTROY_FN	This function is for freeing a filter function or indexer function.
SLAPI_PLUGIN_INTERNAL_PRE_ADD_FN	This function is called before an internal LDAP add operation is completed.
SLAPI_PLUGIN_INTERNAL_PRE_DELETE_FN	This function is called before an internal LDAP delete operation is completed.
SLAPI_PLUGIN_INTERNAL_PRE_MODIFY_FN	This function is called before an internal LDAP modify operation is completed.
SLAPI_PLUGIN_INTERNAL_PRE_MODRDN_FN	This function is called before an internal LDAP modify RDN operation is completed.

52.2. POST-OPERATION/DATA NOTIFICATION PLUG-INS

The parameters listed in this section are used to register post-operation/data notification plug-in functions.

Parameter ID	Description
SLAPI_PLUGIN_POST_BIND_FN	This function is called after an LDAP bind operation is completed.

Parameter ID	Description
SLAPI_PLUGIN_POST_UNBIND_FN	This function is called after an LDAP unbind operation is completed.
SLAPI_PLUGIN_POST_SEARCH_FN	This function is called after an LDAP search operation is completed.
SLAPI_PLUGIN_POST_COMPARE_FN	This function is called after an LDAP compare operation is completed.
SLAPI_PLUGIN_POST_MODIFY_FN	This function is called after an LDAP modify operation is completed.
SLAPI_PLUGIN_POST_MODRDN_FN	This function is called after an LDAP modify RDN operation is completed.
SLAPI_PLUGIN_POST_ADD_FN	This function is called after an LDAP add operation is completed.
SLAPI_PLUGIN_POST_DELETE_FN	This function is called after an LDAP delete operation is completed.
SLAPI_PLUGIN_POST_ABANDON_FN	This function is called after an LDAP abandon operation is completed.
SLAPI_PLUGIN_POST_ENTRY_FN	This function is called after an entry is sent back to the client.
SLAPI_PLUGIN_POST_REFERRAL_FN	This function is called after a set of referrals is sent back to the client.
SLAPI_PLUGIN_POST_RESULT_FN	This function is called after a set of search results is sent back to the client.
SLAPI_PLUGIN_START_FN	This function is called after the server starts up. You can specify a start function for each post-operation plug-in.
SLAPI_PLUGIN_INTERNAL_POST_ADD_FN	This function is called after an internal LDAP add operation is completed.
SLAPI_PLUGIN_INTERNAL_POST_DELETE_FN	This function is called after an internal LDAP delete operation is completed.
SLAPI_PLUGIN_INTERNAL_POST_MODIFY_FN	This function is called after an internal LDAP modify operation is completed.

Parameter ID	Description
SLAPI_PLUGIN_INTERNAL_POST_MODRDN_FN	This function is called after an internal LDAP modify RDN operation is completed.

52.3. MATCHING RULE PLUG-INS

The parameters listed below are used with matching rule plug-in functions that can be registered.

Parameter ID	Description
SLAPI_PLUGIN_MR_FILTER_CREATE_FN	This is a factory function for creating filter functions. This function must be thread-safe since the server may call it concurrently with other functions.
SLAPI_PLUGIN_MR_INDEXER_CREATE_FN	This is a factory function for creating indexer functions. This function must be thread-safe since the server may call it concurrently with other functions.
SLAPI_PLUGIN_MR_FILTER_MATCH_FN	This functions uses the ID to set and get a filter function.
SLAPI_PLUGIN_MR_FILTER_INDEX_FN	This is a filter function that uses an index to accelerate the processing of a search request.
SLAPI_PLUGIN_MR_FILTER_RESET_FN	This function resets the filter function.
SLAPI_PLUGIN_MR_INDEX_FN	This function uses the ID to get and set the index function.

52.4. ENTRY PLUG-INS

The parameters listed below are used for entry store and entry fetch plug-in functions. These plug-in functions are called by the server before writing an entry to disk and after reading an entry from disk. Entry store and entry fetch plug-in functions are passed using the string representation (in LDIF) of the entry.



NOTE

The Directory Server caches recently added and retrieved entries in memory. The entry fetch plug-in function is called only when reading the entry from the disk, not when reading the entry from the cache.

Parameter ID	Description
SLAPI_PLUGIN_ENTRY_FETCH_FUNC	This function fetches information that represents an LDAP entry.
SLAPI_PLUGIN_ENTRY_STORE_FUNC	This function stores information about an entry that was fetched by the SLAPI_PLUGIN_ENTRY_FETCH_FUNC function.

CHAPTER 53. PARAMETERS ACCESSIBLE TO ALL PLUG-INS

The parameters listed in this section are accessible to all types of plug-ins.

53.1. INFORMATION ABOUT THE DATABASE

The parameters listed below specify information about the backend database. These parameters are available for all types of plug-ins. These specific parameters cannot be set by calling `slapi_pblock_set()`. You can, however, get these parameters by calling `slapi_pblock_get()`.

Parameter ID	Data Type	Description
SLAPI_BACKEND	Slapi_Backend *	The database backend servicing this operation. The value may be NULL if there is currently no backend associated with the operation.
SLAPI_BE_MONITORDN	char *	This is no longer supported.
SLAPI_BE_TYPE	char *	Type of backend database; this is the type specified by the <i>nsslapd-database</i> directive in the server configuration file.
SLAPI_BE_READONLY	int	Specifies whether the backend database is read-only; this is determined by the <i>nsslapd-readonly</i> directive in the server configuration file. <ul style="list-style-type: none"> • 1 means that the database backend is read-only. • 0 means that the database backend is writeable.
SLAPI_DBSIZE	int	Specifies the size of the backend database.
SLAPI_BE_LASTMOD	int	If 0 (false), the database does not keep track of the last modification time and who modified it. If non-zero (true), the database does keep track. The default is true.

Parameter ID	Data Type	Description
SLAPI_BE_FLAG_REMOTE_DATA	int	Flag that indicates the entries held by the backend are remote.
SLAPI_BE_ALL_BACKENDS	int	Special value that is returned by a distribution plug-in function to indicate that all backends should be searched. Used only for search operations.
SLAPI_BE_MAXNESTLEVEL	int *	Indicates the maximum number of nesting levels allowed within groups for access control evaluation.
SLAPI_CLIENT_DNS	struct berval **	Contains a list of client IP addresses that are registered in DNS. Used to determine the authorization type.
SLAPI_FAIL_DISKFULL	int	Return code for a backend API call that indicates the disk is full and the operation has failed.
SLAPI_FAIL_GENERAL	int	Return code for a backend API call that indicates that the operation has failed due to some cause other than disk full.

53.2. INFORMATION ABOUT THE CONNECTION

The parameters listed below specify information about the connection. These parameters are available for all types of plug-ins.

Table 53.1. Information about the Connection

Parameter ID	Data Type	Description
SLAPI_CONN_SASL_SSF	int *	The security strength factor provided by the SASL layer used by the connection.

Parameter ID	Data Type	Description
SLAPI_CONN_CERT	CERTCertificate * This is an NSS database. See https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS	The client certificate associated with the connection; may be absent.
SLAPI_CONN_IS_REPLICATION_SESSION	char *	Indicates the current connection is a replication session.
SLAPI_CONN_IS_SSL_CONNECTION	char *	Indicates the current connection is through TLS.
SLAPI_CONNECTION	Slapi_Connection *	Information about the current client connection.
SLAPI_CONN_ID	int *	ID identifying the current connection.
SLAPI_CONN_DN	char *	DN of the user authenticated on the current connection. If you call to get this DN, you should call slapi_ch_free_string() to free the resulting DN when done.

Parameter ID	Data Type	Description
SLAPI_CONN_AUTHMETHOD	char *	<p>Method used to authenticate the current user. If you call to get this value, you should call slapi_ch_free_string() to free the resulting value when done. This parameter can have one of the following values:</p> <ul style="list-style-type: none"> • SLAPD_AUTH_NONE specifies that no authentication mechanism was used (for example, in cases of anonymous authentication). • SLAPD_AUTH_SIMPLE specifies that simple authentication (user name and password) was used to authenticate the current user. • SLAPD_AUTH_SSL specifies that TLS (certificate-based authentication) was used to authenticate the current user. • SLAPD_AUTH_SASL specifies that a SASL (Simple Authentication and Security Layer) mechanism was used to authenticate the current user.
SLAPI_CONN_AUTHTYPE	char *	<p>This parameter has been deprecated for current releases. Use SLAPI_CONN_AUTHMETHOD instead.</p>
SLAPI_CONN_CLIENTNETADDR [a]	PRNetAddr	IP address of the client requesting the operation.

Parameter ID	Data Type	Description
SLAPI_CONN_SERVERNETADDR[a]	PRNetAddr	IP address to which the client is connecting. You might want to use this parameter if, for example, your server accepts connections on multiple IP addresses.
[a] These parameters use an NSPR structure. See https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSPR .		

53.3. INFORMATION ABOUT THE OPERATION

The parameters listed below specify information about the current operation. These parameters are available for all types of plug-ins.

Parameter ID	Data Type	Description
SLAPI_OPERATION	Slapi_Operation *	Information about the operation currently in progress.
SLAPI_OPINITIATED_TIME	time_t	Time when the server began processing the operation.
SLAPI_REQUESTOR_ISROOT	int	<p>Specifies whether the user requesting the operation is the root DN.</p> <ul style="list-style-type: none"> • 1 means that the root DN is requesting the operation. • 0 means that the user requesting the operation is not the root DN. <p>The root DN is the superuser of the directory. This DN is specified by the nsslapd-rootdn attribute in the cn=config entry in the server configuration file.</p>
SLAPI_REQUESTOR_ISUPDAT EDN	int	Deprecated.
SLAPI_REQUESTOR_DN	char *	Specifies the DN of the user requesting the operation.

Parameter ID	Data Type	Description
SLAPI_TARGET_DN	char *	Specifies the DN to which the operation applies; for example, the DN of the entry being added or removed.
SLAPI_REQCONTROLS	LDAPControl **	Array of the controls specified in the request.
SLAPI_CONTROLS_ARG	LDAPControl **	Allows control arguments to be passed before an operation object is created.

53.4. INFORMATION ABOUT EXTENDED OPERATIONS

The following table lists the parameters in the parameter block passed to extended operation functions. If you are writing your own plug-in function for performing this work, you can get these values by calling the function.

Parameter ID	Data Type	Description
SLAPI_EXT_OP_REQ_OID	char *	Object ID (OID) of the extended operation specified in the request.
SLAPI_EXT_OP_REQ_VALUE	struct berval*	Value specified in the request.
SLAPI_EXT_OP_RET_OID	char *	Object ID (OID) that you want sent back to the client.
SLAPI_EXT_OP_RET_VALUE	struct berval*	Return value that you want sent back to the client.

53.5. INFORMATION ABOUT THE TRANSACTION

The parameters listed below specify information about the current transaction. These parameters are available for all types of plug-ins.

Parameter ID	Data Type	Description
SLAPI_PARENT_TXN	void *	Parent transaction.
SLAPI_TXN	void *	ID for current transaction.

53.6. INFORMATION ABOUT ACCESS CONTROL LISTS

The parameters listed below are used with the access control list (ACL) plug-in functions to determine access control levels.

Parameter ID	Data Type	Description
SLAPI_PLUGIN_ACL_ALLOW_ACCESS	int	Flag sent to the ACL plug-in when it is called that indicates that ACL access is allowed.
SLAPI_PLUGIN_ACL_INIT	int	Flag that is set when ACL plug-ins are initialized that allows the use of ACL plug-in access functions.
SLAPI_PLUGIN_ACL_MODS_ALLOWED	int	Flag that indicates if the modifications that were made are allowed.
SLAPI_PLUGIN_ACL_MODS_UPDATE	int	Flag that indicates you can modify (remove, add, or change) the access control items (ACIs).
SLAPI_PLUGIN_ACL_SYNTAX_CHECK	int	Flag that verifies the ACI being added for the entry has a valid syntax.

53.7. NOTES IN THE ACCESS LOG

The parameters listed below specify notes that can be appended to access log entries. These parameters are available for all types of plug-ins.

ParameterID	Data Type	Description
-------------	-----------	-------------

ParameterID	Data Type	Description
SLAPI_OPERATION_NOTES	unsigned int	<p>Flags specifying the notes that you want appended to access log entries. You can set this parameter to the following value:</p> <ul style="list-style-type: none"> • SLAPI_OP_NOTE_UNINDEXED specifies that you want the string Notes=U appended to access log entries. You can use this to indicate that a search operation could not use indexes to generate a smaller list of candidates. <p>If no flags are set, no notes are appended to access log entries.</p>

53.8. INFORMATION ABOUT THE PLUG-IN

The parameters listed below specify information about the plug-in that is available to all plug-in functions defined in the current library. These parameters are available for all types of plug-ins.

Parameter ID	Data Type	Description
SLAPI_PLUGIN	void *	Pointer to the internal server representation of this plug-in.
SLAPI_PLUGIN_PRIVATE	void *	Private data that you want passed to your plug-in functions.
SLAPI_PLUGIN_TYPE	int	Specifies the type of plug-in function (refer to Section 53.8.1, “Types of Plug-ins”)
SLAPI_PLUGIN_ARGV	char **	NULL-terminated array of command-line arguments specified for the plugin directive in the server configuration file.

Parameter ID	Data Type	Description
SLAPI_PLUGIN_ARGC	int	Number of command-line arguments specified for the plugin directive in the server configuration file.
SLAPI_PLUGIN_VERSION	char *	Specifies the version of the plug-in function (refer to Section 53.8.2, “Version Information”).
SLAPI_PLUGIN_OPRETURN	int	Specifies the return value of the LDAP operation that has just been processed.
SLAPI_PLUGIN_OBJECT	void *	Reserved for internal use only; used with filter processing.
SLAPI_PLUGIN_DESTROY_FN	void *	Reserved for internal use only; used with filter processing.
SLAPI_PLUGIN_DESCRIPTION	char *	Provides a description of this plug-in function.
SLAPI_PLUGIN_IDENTITY	char *	Identifies this plug-in function.

53.8.1. Types of Plug-ins

The **SLAPI_PLUGIN_TYPE** parameter can have one of the following values, which identifies the type of the current plug-in:

Defined Constant	Description
SLAPI_PLUGIN_DATABASE	Deprecated.
SLAPI_PLUGIN_EXTENDEDOP	Extended operation plug-in.
SLAPI_PLUGIN_BETXNEXTENDEDOP	Transactional extended operation plugin.
SLAPI_PLUGIN_DATABASE	Pre-operation/data validation plug-in.
SLAPI_PLUGIN_POSTOPERATION	Post-operation/data notification plug-in.
SLAPI_PLUGIN_MATCHINGRULE	Matching rule plug-in.

Defined Constant	Description
SLAPI_PLUGIN_SYNTAX	Syntax plug-in.
SLAPI_PLUGIN_ACL	Access control plug-in.
SLAPI_PLUGIN_BEPREOPERATION	Database pre-operation plug-in.
SLAPI_PLUGIN_BEPOSTOPERATION	Database post-operation plug-in.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME	Password storage scheme plug-in.
SLAPI_PLUGIN_REVER_PWD_STORAGE_SCHEME	Reverse password storage scheme plug-in.
SLAPI_PLUGIN_VATTR_SP	Virtual attribute service provider plug-in.
SLAPI_PLUGIN_INDEX	Indexing plug-in.
SLAPI_PLUGIN_TYPE_OBJECT	Object type plug-in.
SLAPI_PLUGIN_LDBM_ENTRY_FETCH_STORE	Plug-in that fetches and stores an entry from the default backend database (ldbm).

53.8.2. Version Information

To set the value of the **SLAPI_PLUGIN_VERSION** parameter, you can specify one of the following values:

Defined Constant	Description
SLAPI_PLUGIN_CURRENT_VERSION	The current version of the Directory Server plug-in.
SLAPI_PLUGIN_VERSION_01	Version 1 of the plug-in interface, which is supported by the Directory Server 3.x and subsequent releases (including 4.0).
SLAPI_PLUGIN_VERSION_02	Version 2 of the plug-in interface, which is supported by the Directory Server 4.x release but not by previous releases.
SLAPI_PLUGIN_VERSION_03	Version 3 of the plug-in interface, which is supported by current releases of Directory Server but not by previous releases.

53.9. INFORMATION ABOUT COMMAND-LINE ARGUMENTS

The parameters listed below are used to determine the command-line arguments with which a plug-in was invoked.

Parameter ID	Data Type	Description
SLAPI_ARGC	int	Determines the number of command-line arguments with which the Directory Server was invoked.
SLAPI_ARGV	char **	Pointer to an array of character strings that contain the command-line arguments, one per string, with which the Directory Server was invoked.

53.10. INFORMATION ABOUT ATTRIBUTES

The parameters listed below provide the following information about attributes:

- [Section 53.10.1, “Attribute Names”](#)
- [Section 53.10.2, “Attribute Flags”](#)
- [Section 53.10.3, “Attribute Comparisons”](#)

53.10.1. Attribute Names

The parameters listed below are used to check for commonly-used attribute names. These are not pblock parameters but macros that define strings; for example, **SLAPI_ATTR_OBJECTCLASS** is **objectclass**.

Parameter ID	Data Type	Description
SLAPI_ATTR_NSCP_ENTRYDN	int	The nscpEntryDN attribute value.
SLAPI_ATTR_OBJECTCLASS	int	The objectclass attribute value.
SLAPI_ATTR_UNIQUEID	int	The nsuniqueid (unique ID) attribute value.
SLAPI_ATTR_VALUE_PARENT_UNIQUEID	int	The nsParentUniqueID attribute value.
SLAPI_ATTR_VALUE_TOMBSTONE	int	The nsTombstone attribute value.

53.10.2. Attribute Flags

The parameters listed below are used by the **slapi_attr_get_flags()** function to get the flags associated with the specified attribute. These flags can identify an attribute as a single-valued attribute, an operational attribute, or as a read-only attribute.

Parameter ID	Description
SLAPI_ATTR_FLAG_COLLECTIVE	Flag that indicates the optional collective marker has been set. This is not supported.
SLAPI_ATTR_FLAG_NOUSERMOD	Flag that indicates this attribute cannot be modified by a user over LDAP.
SLAPI_ATTR_FLAG_OBSOLETE	Flag that indicates this attribute is obsolete.
SLAPI_ATTR_FLAG_OPATTR	Flag that determines if the attribute is an operational attribute.
SLAPI_ATTR_FLAG_READONLY	Flag that determines if the attribute is read-only.
SLAPI_ATTR_FLAG_SINGLE	Flag that determines if the attribute is single-valued.
SLAPI_ATTR_FLAG_STD_ATTR	Flag that indicates that this is a standard, non-user-defined attribute that is not listed in the user defined schema file, which is typically the schema file named 99user.ldif . Standard attribute types can't be deleted by modifying the subschema subentry (cn=schema) over LDAP.

53.10.3. Attribute Comparisons

The parameters listed below are used with the **slapi_attr_type_cmp()** plug-in function to compare two components of an attribute.

Parameter ID	Description
SLAPI_TYPE_CMP_BASE	Ignores the options on both names and compares the base names only.
SLAPI_TYPE_CMP_EXACT	Compares the base name plus options, as specified.
SLAPI_TYPE_CMP_SUBTYPE	Ignores the options on the second name that are not in the first name.

53.11. INFORMATION ABOUT TARGETS

The parameters listed below provide information about targets. These parameters are available for all types of plug-ins.

Parameter ID	Data Type	Description
SLAPI_TARGET_ADDRESS	void *	Indicates the target address (DN + <i>uniqueid</i>) should be normalized.
SLAPI_TARGET_DN	char *	Indicates the target DN of the operation, which is normalized.
SLAPI_TARGET_UNIQUEID	char *	Indicates the target <i>uniqueid</i> should be normalized.

CHAPTER 54. PARAMETERS FOR THE BIND FUNCTION

The following table lists the parameters in the parameter block passed to the database bind function. If you are writing a pre-operation, database, or post-operation bind function, you can get these values by calling the function.

Parameter ID	Data Type	Description
SLAPI_BIND_TARGET	char *	DN of the entry to which to bind.
SLAPI_BIND_METHOD	int	Authentication method used; for example, LDAP_AUTH_SIMPLE or LDAP_AUTH_SASL .
SLAPI_BIND_CREDENTIALS	struct berval *	Credentials from the bind request.
SLAPI_BIND_RET_SASLCREDENTIALS	struct berval *	Simple Authentication and Security Layer (SASL) credentials that you want to send back to the client. Set this before calling .
SLAPI_BIND_SASLMECHANISM	char *	Simple Authentication and Security Layer (SASL) mechanism that is used (for example, LDAP_SASL_EXTERNAL).

See [Table 7.9, “Table of Information Processed during an LDAP Abandon Operation”](#), for more information on these parameters.

CHAPTER 55. PARAMETERS FOR THE SEARCH FUNCTION

The following parameters are used with the search function:

- [Section 55.1, “Parameters Passed to the Search Function”](#)
- [Section 55.2, “Parameters for Executing the Search”](#)
- [Section 55.3, “Parameters for the Search Results”](#)

55.1. PARAMETERS PASSED TO THE SEARCH FUNCTION

The following table lists the parameters in the parameter block passed to the database search function. If you are writing a pre-operation, database, or post-operation search function, you can get these values by calling the function.

Parameter ID	Data Type	Description
SLAPI_SEARCH_TARGET	char *	DN of the base entry in the search operation; the starting point of the search.
SLAPI_ORIGINAL_TARGET_DN	char *	The original DN sent by the client (this DN is normalized by SLAPI_SEARCH_TARGET); read-only parameter.
SLAPI_SEARCH_SCOPE	int	The scope of the search. The scope can be one of the following values: <ul style="list-style-type: none"> • LDAP_SCOPE_BASE • LDAP_SCOPE_ONELEVEL • LDAP_SCOPE_SUBTREE
SLAPI_SEARCH_DEREF	int	Method for handling aliases in a search. This method can be one of the following values: <ul style="list-style-type: none"> • LDAP_DEREF_NEVER • LDAP_DEREF_SEARCHING • LDAP_DEREF_FINDING • LDAP_DEREF_ALWAYS

Parameter ID	Data Type	Description
SLAPI_SEARCH_SIZELIMIT	int	Maximum number of entries to return in the search results.
SLAPI_SEARCH_TIMELIMIT	int	Maximum amount of time (in seconds) allowed for the search operation.
SLAPI_SEARCH_FILTER	Slapi_Filter *	Structure (an opaque data structure) representing the filter to be used in the search.
SLAPI_SEARCH_STRFILTER	char *	String representation of the filter to be used in the search.
SLAPI_SEARCH_ATTRS	char **	Array of attribute types to be returned in the search results.
SLAPI_SEARCH_ATTRONLY	int	Specifies whether the search results return attribute types only or attribute types and values. 0 means return both attributes and values; 1 means return attribute types only.

55.2. PARAMETERS FOR EXECUTING THE SEARCH

The following parameters are set by the frontend and backend database as part of the process of executing the search.

Parameter ID	Data Type	Description
SLAPI_SEARCH_RESULT_SET	void *	Set of search results.
SLAPI_SEARCH_RESULT_ENTRY	void *	Entry returned from iterating through the results set.
SLAPI_SEARCH_RESULT_ENTRY_EXT	void *	Reserved for future use.
SLAPI_NENTRIES	int	Number of search results found.
SLAPI_SEARCH_REFERRALS	struct berval **	Array of the URLs to other LDAP servers to which the current server is referring the client.

See [Section 7.4, “Processing an LDAP Search Operation”](#), and [Section 7.4.2, “Iterating through Candidates”](#), for more information on these parameters.

55.3. PARAMETERS FOR THE SEARCH RESULTS

The entry and referrals options listed below are set/read by both the frontend and backend database while stepping through the search results.

Parameters that Return Data Types

The parameters listed below return data types.

Parameter ID	Data Type	Description
SLAPI_RESULT_CODE	int *	Result code that was encountered during the search; this corresponds to the resultCode field within an LDAPResult message.
SLAPI_RESULT_MATCHED	char *	The portion of the target DN that was matched; this corresponds to the matchedDN field within an LDAPResult message.
SLAPI_RESULT_TEXT	char *	The textual error message; this corresponds to the errorMessage field within an LDAPResult message.
SLAPI_PB_RESULT_TEXT	char *	A textual error message passed from internal subsystems to a plug-in. Currently used by the slapi_entry_schema_check() function to provide extra explanatory information when it returns a non-zero value, when the schema check fails.

CHAPTER 56. PARAMETERS THAT CONVERT STRINGS TO ENTRIES

The parameters listed below are pblock parameters; they are flags that can be passed to the `slapi_str2entry()` function.

Parameter ID	Description
SLAPI_STR2ENTRY_ADDRDNVALS	In the conversion from strings to entries, adds the RDN value as an attribute if it is not present.
SLAPI_STR2ENTRY_BIGENTRY	Provides a hint that the entry is large; this enables some optimizations related to large entries.
SLAPI_STR2ENTRY_EXPAND_OBJECT_CLASSES	Adds any missing ancestor values based on the object class hierarchy.
SLAPI_STR2ENTRY_IGNORE_STATE	Ignores entry state information if present.
SLAPI_STR2ENTRY_INCLUDE_VERSION_STR	Returns entries that have a version: 1 line as part of the LDIF representation.
SLAPI_STR2ENTRY_NOT_WELL_FORMED_LDIF	Informs <code>slapi_str2entry()</code> that the LDIF input is not well formed. Well formed LDIF input has no duplicate attribute values, already has the RDN as an attribute of the entry, and has all values for a given attribute type listed contiguously.
SLAPI_STR2ENTRY_REMOVEDUPVALS	Removes duplicate values.
SLAPI_STR2ENTRY_TOMBSTONE_CHECK	Checks to see if the entry is a tombstone; if so, sets the tombstone flag.

CHAPTER 57. PARAMETERS FOR THE ADD FUNCTION

The following table lists the parameters in the parameter block passed to the database add function. If you are writing a pre-operation, database, or post-operation add function, you can get these values by calling the function.

Parameter ID	Data Type	Description
SLAPI_ADD_TARGET	char *	DN of the entry to be added.
SLAPI_ADD_ENTRY	Slapi_Entry *	The entry to be added (specified as the opaque datatype).
SLAPI_ADD_EXISTING_DN_ENTRY	Slapi_Entry *	Internal only; used by the multi-master replication update resolution procedure code. If adding an entry that already exists, this is the entry which has the same DN.
SLAPI_ADD_PARENT_ENTRY	Slapi_Entry *	Internal only; used by the multi-master replication update resolution procedure code. This is the parent entry of the entry to add.
SLAPI_ADD_PARENT_UNIQUE ID	char *	Internal only; used by the multi-master replication update resolution procedure code. This is the unique ID of the parent entry of the entry to add.
SLAPI_ADD_EXISTING_UNIQUEID_ENTRY	Slapi_Entry *	Internal only; used by the multi-master replication resolution procedure code. If adding an entry that already exists, this is the entry which has the same unique ID.

See [Section 7.6, “Processing an LDAP Add Operation”](#), for more information on these parameters.

CHAPTER 58. PARAMETERS FOR THE COMPARE FUNCTION

The following table lists the parameters in the parameter block passed to the database compare function. If you are writing a pre-operation, database, or post-operation compare function, you can get these values by calling the [slapi_pblock_get\(\)](#) function.

Parameter ID	Data Type	Description
SLAPI_COMPARE_TARGET	char *	DN of the entry to be compared.
SLAPI_COMPARE_TYPE	char *	Attribute type to use in the comparison.
SLAPI_COMPARE_VALUE	struct berval *	Attribute value to use in the comparison

See [Section 7.5, “Processing an LDAP Compare Operation”](#), for more information on these parameters.

CHAPTER 59. PARAMETERS FOR THE DELETE FUNCTION

The following table lists the parameters in the parameter block passed to the database delete function. If you are writing a pre-operation, database, or post-operation delete function, you can get these values by calling the function.

Parameter ID	Data Type	Description
SLAPI_DELETE_TARGET	char *	DN of the entry to delete.
SLAPI_DELETE_EXISTING_ENTRY	Slapi_Entry *	Internal only; used by the multi-master replication resolution procedure code.

See [Section 7.9, “Processing an LDAP Delete Operation”](#), for more information on these parameters.

CHAPTER 60. PARAMETERS FOR THE MODIFY FUNCTION

The following table lists the parameters in the parameter block passed to the database modify function. If you are writing a pre-operation, database, or post-operation modify function, you can get these values by calling the function.

Parameter ID	Data Type	Description
SLAPI_MODIFY_TARGET	char *	DN of the entry to be modified.
SLAPI_MODIFY_MODS	LDAPMod **	A NULL-terminated array of LDAPMod structures, which represent the modifications to be performed on the entry.
SLAPI_MODIFY_EXISTING_ENTRY	Slapi_Entry *	Internal only; used by the multi-master replication update resolution procedure code.

See [Section 7.7, “Processing an LDAP Modify Operation”](#), for more information on these parameters.

CHAPTER 61. PARAMETERS FOR THE MODIFY RDN FUNCTION

The following table lists the parameters in the parameter block passed to the database modify RDN function. If you are writing a pre-operation, database, or post-operation modify RDN function, you can get these values by calling the [slapi_pblock_get\(\)](#) function.

Parameter ID	Data Type	Description
SLAPI_MODRDN_TARGET	char *	DN of the entry that you want to rename.
SLAPI_MODRDN_NEWRDN	char *	New RDN to assign to the entry.
SLAPI_MODRDN_DELOLDRDN	int	Specifies whether you want to delete the old RDN. 0 means do not delete the old RDN; 1 means delete the old RDN.
SLAPI_MODRDN_NEWSUPERIOR	char *	DN of the new parent of the entry, if the entry is being moved to a new location in the directory tree.
SLAPI_MODRDN_EXISTING_ENTRY	Slapi_Entry *	Internal only; used by the multi-master replication update resolution code. If the destination RDN of the modrdn already exists, this is that entry.
SLAPI_MODRDN_PARENT_ENTRY	Slapi_Entry *	Internal use only; used by the multi-master replication update resolution procedure code. This is the parent entry.
SLAPI_MODRDN_NEWPARENT_ENTRY	Slapi_Entry *	Internal only; used by the multi-master replication update resolution procedure code. This is the new parent entry.
SLAPI_MODRDN_TARGET_ENTRY	Slapi_Entry *	Internal only; used by the multi-master replication update resolution procedure code.

Parameter ID	Data Type	Description
SLAPI_MODRDN_NEWSUPERIOR_ADDRESS	void *	Internal only; used by the multi-master replication update resolution procedure code.

See [Section 7.8, “Processing an LDAP Modify RDN Operation”](#), for more information on these parameters.

CHAPTER 62. PARAMETERS FOR THE ABANDON FUNCTION

The following table lists the parameters in the parameter block passed to the database abandon function. If you are writing a pre-operation, database, or post-operation abandon function, you can get these values by calling the function.

Parameter ID	Data Type	Description
SLAPI_ABANDON_MSGID	unsigned long	Message ID of the operation to abandon.

See [Section 7.10, “Processing an LDAP Abandon Operation”](#), for more information on these parameters.

CHAPTER 63. PARAMETERS FOR THE MATCHING RULE FUNCTION

The following table lists the parameters in the parameter block passed to the database matching rule function.

Parameter ID	Data Type	Description
SLAPI_PLUGIN_MR_OID	char *	Matching rule OID (if any) specified in the extensible match filter.
SLAPI_PLUGIN_MR_TYPE	char *	Attribute type (if any) specified in the extensible match filter.
SLAPI_PLUGIN_MR_VALUE	struct berval *	Value specified in the extensible match filter.
SLAPI_PLUGIN_MR_VALUES	struct berval ** values	Pointer to an array of berval structures containing the values of the entry's attributes that need to be indexed.
SLAPI_PLUGIN_MR_KEYS	struct berval **	Keys generated for the values specified in the SLAPI_PLUGIN_MR_VALUES parameter. The server creates indexes using these keys.
SLAPI_PLUGIN_MR_FILTER_REUSABLE	unsigned int *	Matching rule filter that is reusable.
SLAPI_PLUGIN_MR_QUERY_OPERATOR	int *	Query operator used by the server to determine how to compare the keys generated from SLAPI_PLUGIN_MR_VALUES and SLAPI_PLUGIN_MR_INDEX_FN against keys in the index.

Parameter ID	Data Type	Description
SLAPI_PLUGIN_MR_USAGE	unsigned int *	<p>Specifies the intended use of the indexer object. This parameter can have one of the following values:</p> <ul style="list-style-type: none"> • SLAPI_PLUGIN_MR_USAGE_INDEX specifies that the indexer object should be used to index entries. • SLAPI_PLUGIN_MR_USAGE_SORT specifies that the indexer object should be used to sort entries. <p>You can use this to specify different information in the indexer object or different indexer functions, based on whether the plug-in is used for indexing or sorting.</p>

The following extended filter argument parameters are used with LDAPv3 only:

- **SLAPI_MR_FILTER_ENTRY**
- **SLAPI_MR_FILTER_TYPE**
- **SLAPI_MR_FILTER_VALUE**
- **SLAPI_MR_FILTER_OID**
- **SLAPI_MR_FILTER_DNATTRS**

The following function sets all three parameters:

- **SLAPI_SEARCH_INTERNAL_PB()**

63.1. QUERY OPERATORS IN EXTENSIBLE MATCH FILTERS

The server checks the value of the **SLAPI_PLUGIN_MR_QUERY_OPERATOR** parameter to determine which operator is specified. The following parameters are defined values for the **SLAPI_PLUGIN_MR_QUERY_OPERATOR**:

Parameter ID	Description
SLAPI_OP_LESS	Less than (<) operator.
SLAPI_OP_LESS_OR_EQUAL	Less than or equal to (<=) operator.

Parameter ID	Description
SLAPI_OP_EQUAL	Equal to (=) operator.
SLAPI_OP_GREATER_OR_EQUAL	Greater than or equal to (>=) operator.
SLAPI_OP_GREATER	Greater than (>) operator.
SLAPI_OP_SUBSTRING	Allows an operation to use a wildcard (*) in a search filter. When used in a table it can be stated as cn=a* , cn=*a , or cn = *a* .

CHAPTER 64. PARAMETERS FOR LDBM BACKEND PRE- AND POST-OPERATION FUNCTIONS

The section describes the parameters that are used with the LDBM Backend plug-in functions:

- [Section 64.1, “Pre-Operation Plug-ins”](#)
- [Section 64.2, “Post-Operation Plug-ins”](#)

These functions are called by the LDBM Backend, for example, the **SLAPI_PLUGIN_BE_PRE_DELETE_FN** is called by the LDBM Backend before a delete operation is carried out but after the all of the more general **SLAPI_PLUGIN_PRE_DELETE_FN** functions have been called.

64.1. PRE-OPERATION PLUG-INS

The parameters listed in this section are used with pre-operation database plug-in functions.

Parameter ID	Description
SLAPI_PLUGIN_BE_PRE_ADD_FN	This function is called before a database add operation is completed.
SLAPI_PLUGIN_BE_PRE_DELETE_FN	This function is called before a database delete operation is completed.
SLAPI_PLUGIN_BE_PRE_MODIFY_FN	This function is called before a database modify operation is completed.
SLAPI_PLUGIN_BE_PRE_MODRDN_FN	This function is called before a database modify RDN operation is completed.

64.2. POST-OPERATION PLUG-INS

The parameters listed in this section are used with post-operation database plug-in functions.

Parameter ID	Description
SLAPI_PLUGIN_BE_POST_ADD_FN	This function is called after a database add operation is completed.
SLAPI_PLUGIN_BE_POST_DELETE_FN	This function is called after a database delete operation is completed.
SLAPI_PLUGIN_BE_POST_MODIFY_FN	This function is called after a database modify operation is completed.

Parameter ID	Description
SLAPI_PLUGIN_BE_POST_MODRDN_FN	This function is called after a database modify RDN operation is completed.

CHAPTER 65. PARAMETERS FOR LDBM BACK END TRANSACTION PRE- AND POST-OPERATION FUNCTIONS

The section describes the parameters that the LDBM back end transaction plug-in functions use:

- [Section 64.1, “Pre-Operation Plug-ins”](#)
- [Section 64.2, “Post-Operation Plug-ins”](#)

These functions are called inside the database transaction and perform atomic operations. For example, if an error occurs during a pre- or post-operation, you can revert the complete database transaction.

An updated operation, that is performed inside of a transaction plug-in, runs as a child transaction of the enclosing parent transaction. This means:

- The plug-in uses data that other child transactions of the parent transaction already committed. For example:
 - A **modify** operation can use data that another child transaction of the enclosing parent transaction modified and committed.
 - A **search** operation returns data that another child transaction of the enclosing parent transaction modified and committed.
- If the plug-in writes to the database, but a subsequent operation fails in the same enclosing parent transaction, the plug-in's write operation is removed. Any aborted transaction at any child level is reverted up to the original update operation's transaction.

To correctly abort the changes in case of an error, plug-ins using the pre- or post-operation transaction type must return a non-zero value if the plug-in operation failed. In case of an error, Red Hat recommended to additionally set the result code (**SALPI_RESULT_CODE**) or the return code **opreturn** (**SLAPI_PLUGIN_OPRETURN**) in the **pblock** parameter.

Due to the nature of the LDBM database, deadlocks can occur in database operations. Other database errors cause the transaction to abort and to return the error. However, in case of a deadlock the operation is retried up to 50 times. The back end transaction pre- and post-plug-ins are called as many times as the retry is attempted, while the back end pre- and post-plug-ins are not.

65.1. PRE-OPERATION PLUG-INS

The parameters listed in this section are used with back end transaction pre-operation database plug-in functions.

Parameter ID	Description
SLAPI_PLUGIN_BE_TXN_PRE_ADD_FN	This function is called in a database transaction before a database add operation is completed.

Parameter ID	Description
SLAPI_PLUGIN_BE_TXN_PRE_DELETE_FN	This function is called in a database transaction before a database delete operation is completed.
SLAPI_PLUGIN_BE_TXN_PRE_MODIFY_FN	This function is called in a database transaction before a database modify operation is completed.
SLAPI_PLUGIN_BE_TXN_PRE_MODRDN_FN	This function is called in a database transaction before a database modify RDN operation is completed.

65.2. POST-OPERATION PLUG-INS

The parameters listed in this section are used with back end transaction post-operation database plug-in functions.

Parameter ID	Description
SLAPI_PLUGIN_BE_TXN_POST_ADD_FN	This function is called in a database transaction after a database add operation is completed.
SLAPI_PLUGIN_BE_TXN_POST_DELETE_FN	This function is called in a database transaction after a database delete operation is completed.
SLAPI_PLUGIN_BE_TXN_POST_MODIFY_FN	This function is called in a database transaction after a database modify operation is completed.
SLAPI_PLUGIN_BE_TXN_POST_MODRDN_FN	This function is called in a database transaction after a database modify RDN operation is completed.

CHAPTER 66. PARAMETERS FOR THE DATABASE

The parameters listed in this section can be used to get and set information about the database itself, database connections, and database operations.

- [Section 66.1, “Information about the Database”](#)
- [Section 66.2, “Information about Operations”](#)
- [Section 66.3, “Information about Backend State Change”](#)

66.1. INFORMATION ABOUT THE DATABASE

The following parameters can be used as the second argument to the [slapi_pblock_get\(\)](#) and [slapi_pblock_set\(\)](#) functions to get and set information about the database.

Parameter ID	Data Type	Description
SLAPI_BACKEND	Slapi_Backend *	A pointer to the backend database that is handling the operation.
SLAPI_BE_LASTMOD	int *	A value that indicates whether the backend database is tracking modifiersName and modifyTimeStamp ; true if the value is not zero.
SLAPI_BE_READONLY	int *	A value that indicates whether the backend database is accepting updates; not accepting updates if the value is not zero.
SLAPI_BE_TYPE	char *	The database type name; for example, ldbm database.
SLAPI_REQUESTOR_ISROOT	int *	Indicates the requestor is root.

66.2. INFORMATION ABOUT OPERATIONS

The following parameters can be used as the second argument to the [slapi_pblock_get\(\)](#) and [slapi_pblock_set\(\)](#) functions to get and set information about operations.

Parameter ID	Data Type	Description
SLAPI_OPERATION_AUTHTYPE	char *	The authorization type for the operation.

Parameter ID	Data Type	Description
SLAPI_OPERATION_ID	int	The operation ID.
SLAPI_OPERATION_TYPE	int	The operation type; the type is one of the SLAPI_OPERATION_xxx values.
SLAPI_OPINITIATED_TIME	time_t	The time in seconds since 00:00:00 UTC, January 1, 1970, when the Directory Server started processing the operation.
SLAPI_REQUESTOR_DN	char *	The bind DN at the time processing of the operation began.
SLAPI_IS_LEGACY_REPLICATED_OPERATION	int	Flag that indicates this is a legacy replicated operation.
SLAPI_IS_MMR_REPLICATED_OPERATION	int	Flag that indicates this is an MMR replicated operation.
SLAPI_IS_REPLICATED_OPERATION	int	Flag that indicates this is a replicated operation.

66.3. INFORMATION ABOUT BACKEND STATE CHANGE

The following parameters can be used in the [slapi_register_backend_state_change\(\)](#) and [slapi_unregister_backend_state_change\(\)](#) functions to register and unregister callbacks when a backend state changes.

Parameter ID	Data Type	Description
MTN_BE_ON	int	The backend is on .
MTN_BE_OFFLINE	int	The backend is offline (import process).
MTN_BE_DELETE	int	The backend has been deleted.

CHAPTER 67. PARAMETERS FOR LDAP FUNCTIONS

The parameters listed in this section can be used with functions that you can call to perform LDAP operations from a plug-in. These internal operations do not return any data to a client.

- [Section 67.1, “Parameters for LDAP Operations”](#)
- [Section 67.2, “Parameters for LDAP Control”](#)
- [Section 67.3, “Parameters for Generating LDIF Strings”](#)

67.1. PARAMETERS FOR LDAP OPERATIONS

Parameter ID	Data Type	Description
SLAPI_PLUGIN_INTOP_RESULT	int	Result code of the internal LDAP operation.
SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES	Slapi_Entry ** Section 14.22, “Slapi_Entry”	Array of entries found by an internal LDAP search operation. See slapi_search_internal_pb() for details.
SLAPI_PLUGIN_INTOP_SEARCH_REFERRALS	char **	Array of referrals (in the form of LDAP URLs) found by an internal LDAP search operation. See slapi_search_internal_pb() for details.

The following function sets all three parameters: [slapi_search_internal_pb\(\)](#)

The following functions set only the **SLAPI_PLUGIN_INTOP_RESULT** parameter:

- [slapi_add_internal_pb\(\)](#)
- [slapi_delete_internal_pb\(\)](#)
- [slapi_modify_internal_pb\(\)](#)
- [slapi_modrdn_internal_pb\(\)](#)

67.2. PARAMETERS FOR LDAP CONTROL

The parameters listed below provide information about LDAP controls that are used in LDAP operations.

Parameter ID	Data Type	Description
SLAPI_OPERATION_ABANDON	LDAPControl *	This control applies to the LDAP abandon operation.
SLAPI_OPERATION_ADD	LDAPControl *	This control applies to the LDAP add operation.
SLAPI_OPERATION_ANY	LDAPControl *	This control applies to any LDAP operation.
SLAPI_OPERATION_BIND	LDAPControl *	This control applies to the LDAP bind operation.
SLAPI_OPERATION_COMPARE	LDAPControl *	This control applies to the LDAP compare operation.
SLAPI_OPERATION_DELETE	LDAPControl *	This control applies to the LDAP delete operation.
SLAPI_OPERATION_EXTENDED	LDAPControl *	This control applies to the LDAPv3 extended operation.
SLAPI_OPERATION_MODDN	LDAPControl *	This control applies to the LDAP modify DN operation.
SLAPI_OPERATION_MODIFY	LDAPControl *	This control applies to the LDAP modify operation.
SLAPI_OPERATION_MODRDN	LDAPControl *	This control applies to the LDAPv3 modify RDN operation.
SLAPI_OPERATION_NONE	LDAPControl *	This control applies to none of the LDAP operations.
SLAPI_OPERATION_SEARCH	LDAPControl *	This control applies to the LDAP search operation.
SLAPI_OPERATION_UNBIND	LDAPControl *	This control applies to the LDAP unbind operation.
SLAPI_RESCONTROLS	LDAPControl *	The complete set of LDAPv3 controls that will be sent with the LDAP result.
SLAPI_ADD_RESCONTROL	LDAPControl *	Add one LDAPv3 controls to the set that will be sent with the LDAP result.

67.3. PARAMETERS FOR GENERATING LDIF STRINGS

The parameters listed below are used to generate a description of an entry as an LDIF string. These are not pblock parameters but flags that are passed to the `slapi_entry2str_with_options()` function.

Parameter ID	Description
SLAPI_DUMP_MINIMAL_ENCODING	Use the base-64 encoding as little as possible, only when it is required to produce an LDIF fragment that can be parsed. This option is useful for some international data to avoid excessive base-64 encoding. Using this option may produce LDIF that does not conform to the standard.
SLAPI_DUMP_NOOPATTRS	A flag used to suppress the operational attributes. Refer to Section 67.3, “Parameters for Generating LDIF Strings” .
SLAPI_DUMP_NOWRAP	By default, lines will be wrapped as defined in the LDIF specification. This flag disables line wrapping.
SLAPI_DUMP_STATEINFO	This flag is only used internally by replication. This flag allows access to the internal data used by multi-master replication.
SLAPI_DUMP_UNIQUEID	This flag is used when creating an LDIF file that will be used to initialize a replica. Each entry will contain the nsUniqueID operational attribute.

CHAPTER 68. PARAMETERS FOR ERROR LOGGING

The parameters listed below are used with the function to write error messages to the error log, which is by located by default in `/var/log/dirsrv/slapd-instance/errors`.

The severity level of the message is determined by the administrator and determines whether the message is written to the error log. The severity level can have one of the following values:

Parameter ID	Description
SLAPI_LOG_FATAL	This message is always written to the error log. This severity level indicates that a fatal error has occurred in the server.
SLAPI_LOG_TRACE	This message is written to the error log if the log level setting “Trace function calls” is selected. This severity level is typically used to indicate what function is being called.
SLAPI_LOG_PACKETS	This message is written to the error log if the log level setting “Packet handling” is selected.
SLAPI_LOG_ARGS	This message is written to the error log if the log level setting “Heavy trace output” is selected.
SLAPI_LOG_CONNS	This message is written to the error log if the log level setting “Connection management” is selected.
SLAPI_LOG_BER	This message is written to the error log if the log level setting “Packets sent/received” is selected.
SLAPI_LOG_FILTER	This message is written to the error log if the log level setting “Search filter processing” is selected.
SLAPI_LOG_CONFIG	This message is written to the error log if the log level setting “Config file processing” is selected.
SLAPI_LOG_ACL	This message is written to the error log if the log level setting “Access control list processing” is selected.
SLAPI_LOG_SHELL	This message is written to the error log if the log level setting “Log communications with shell backends” is selected.

Parameter ID	Description
SLAPI_LOG_PARSE	This message is written to the error log if the log level setting “Log entry parsing” is selected.
SLAPI_LOG_HOUSE	This message is written to the error log if the log level setting “Housekeeping” is selected.
SLAPI_LOG_REPL	This message is written to the error log if the log level setting “Replication” is selected.
SLAPI_LOG_CACHE	This message is written to the error log if the log level setting “Entry cache” is selected.
SLAPI_LOG_PLUGIN	This message is written to the error log if the log level setting “Plug-ins” is selected. This severity level is typically used to identify messages from server plug-ins.
SLAPI_LOG_TIMING	This message is written to the error log if the log level setting “Log timing” is selected.
SLAPI_LOG_ACLSUMMARY	This message is written to the error log if the log level setting “Log ACL summary” is selected.

CHAPTER 69. PARAMETERS FOR FILTERS

This section lists the parameters used for manipulating LDAP filters, including with functions such as `slapi_filter_join()`. These are not block parameters.

- [Section 69.1, “Parameters for Comparison Filters”](#)
- [Section 69.2, “Parameters for Filter Operations”](#)

69.1. PARAMETERS FOR COMPARISON FILTERS

The parameters listed below are filters that are used to compare a value against an attribute.

Parameter ID	Description
<code>LDAP_FILTER_AND</code>	AND filter. For example: <pre>(&(ou=Accounting)(l=Sunnyvale))</pre>
<code>LDAP_FILTER_APPROX</code>	Approximation filter. For example: <pre>(ou~=Sales)</pre>
<code>LDAP_FILTER_EQUALITY</code>	Equals filter. For example: <pre>(ou=Accounting)</pre>
<code>LDAP_FILTER_EXTENDED</code>	Extensible filter. For example: <pre>(o:dn:=Example)</pre>
<code>LDAP_FILTER_GE</code>	Greater than or equal to filter. For example: <pre>(supportedLDAPVersion>=3)</pre>
<code>LDAP_FILTER_LE</code>	Less than or equal to filter. For example: <pre>(supportedLDAPVersion<=2)</pre>
<code>LDAP_FILTER_OR</code>	OR filter. For example: <pre>((ou=Accounting)(l=Sunnyvale))</pre>
<code>LDAP_FILTER_NOT</code>	NOT filter. For example: <pre>(!(l=Sunnyvale))</pre>

Parameter ID	Description
LDAP_FILTER_PRESENT	Presence filter. For example: (mail=*)
LDAP_FILTER_SUBSTRINGS	Substringfilter. For example: (ou=Account*Department)

69.2. PARAMETERS FOR FILTER OPERATIONS

The parameters listed below return status information about a filter operation. These are values that **slapi_filter_apply()** and programmer-defined filter apply functions may return.

Parameter ID	Description
SLAPI_FILTER_SCAN_NOMORE	Indicates success in traversing the entire filter.
SLAPI_FILTER_SCAN_STOP	Indicates a premature abort.
SLAPI_FILTER_SCAN_CONTINUE	Indicates to continue scanning.
SLAPI_FILTER_SCAN_ERROR	Indicates an error occurred during the traverse and the scan aborted.
SLAPI_FILTER_UNKNOWN_FILTER_TYPE	Error code that SLAPI_FILTER_SCAN_ERROR can set.

CHAPTER 70. PARAMETERS FOR PASSWORD STORAGE

The following plug-in functions and parameters access password storage schemes to encode, decode, and compare passwords:

- [Section 70.1, “Password Storage Plug-ins”](#)
- [Section 70.2, “Parameters for Password Storage”](#)

70.1. PASSWORD STORAGE PLUG-INS

The parameters listed below are used with functions that you can call to store passwords.

Parameter ID	Description
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_CMP_FN	This function accesses a password storage scheme to compare passwords.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DEC_FN	This function accesses a password storage scheme to decode passwords.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_ENC_FN	This function accesses a password storage scheme to encode passwords.

70.2. PARAMETERS FOR PASSWORD STORAGE

The parameters listed below apply to password storage schemes.

Parameter ID	Data Type	Description
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DB_PWD	char *	Value from the database password storage scheme.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_NAME	char *	Name of the password storage scheme.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_USER_PWD	char *	Value of the LDAP user password.
SLAPI_USERPWD_ATTR userpassword	Slapi_Attr *	Attributes for the user password that are used for password handling.

CHAPTER 71. PARAMETERS FOR RESOURCE LIMITS

The following parameters are used to provide information about resource limits:

- [Section 71.1, “Parameter for Binder-Based Resource Limits”](#)
- [Section 71.2, “Status Codes for Resource Limits”](#)

71.1. PARAMETER FOR BINDER-BASED RESOURCE LIMITS

The following parameter is a valid value for the `slapi_reslimit_register()` function.

Parameter ID	Data Type	Description
<code>SLAPI_RESLIMIT_TYPE_INT</code>	int	Valid values for the type parameter for <code>slapi_reslimit_register()</code> .

71.2. STATUS CODES FOR RESOURCE LIMITS

The status codes are used with functions that extract attribute values from a binder entry that corresponds to resource limits. Typically, operational attributes are used to hold binder-specific search size limits.

Any resource limits found in the binder entry are cached in the connection structure by a connection object extension. This means that if the attributes that correspond to the binder entry is changed, the resource limit is not effected until the next bind occurs as that entry.

The status codes are the possible return values for the `slapi_reslimit_register()` and `slapi_reslimit_get_integer_limit()` functions. A plug-in or server subsystem that wants to use the resource limit subsystem should call the `slapi_reslimit_register()` plug-in function once for each limit to be tracked. `slapi_reslimit_register()` should be called before any client connections are accepted.

CHAPTER 72. PARAMETERS FOR THE VIRTUAL ATTRIBUTE SERVICE

The parameters listed in the tables below are used with the virtual attribute service to return information about virtual and real attributes.

These identifiers are flags that can be passed to various `slapi_vattr_values_XXX()` functions in the `flags` parameter:

Parameter	Data Type	Description
SLAPI_REALATTRS_ONLY	int	Flag that indicates only real attributes are used.
SLAPI_VIRTUALATTRS_ONLY	int	Flag that indicates only virtual attributes are used.
SLAPI_VIRTUALATTRS_LIST_OPERATIONAL_ATTRS	int	Flag that indicates the operational attributes should be listed.
SLAPI_VIRTUALATTRS_REQUEST_POINTERS	int	Flag that indicates you wish to receive pointers into the entry, if possible.

These are buffer disposition flags that are returned in the `buffer_flags` parameter:

Parameter	Data Type	Description
SLAPI_VIRTUALATTRS_REAL_ATTRS_ONLY	int	Buffer disposition flag that indicates these are real attributes.
SLAPI_VIRTUALATTRS_RETURNED_COPIES	int	Buffer disposition flag that indicates the virtual attributes returned copies.
SLAPI_VIRTUALATTRS_RETURNED_POINTERS	int	Buffer disposition flag that indicates the virtual attributes returned pointers.

These are attribute type name disposition values that are returned in the ***type_name_disposition*** parameter:

Parameter	Data Type	Description
-----------	-----------	-------------

Parameter	Data Type	Description
SLAPI_VIRTUALATTRS_LOOP_DETECTED	int	Flag that indicates a failure in evaluating the virtual attributes because a loop was detected while locating and calling virtual attribute providers.
SLAPI_VIRTUALATTRS_NOT_FOUND	int	Flag that indicates the attribute type was not recognized by any virtual attribute and is not a real attribute in the entry.
SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_EXACTLY_OR_ALIAS	int	Flag that indicates the attribute name disposition value indicates a matching result.
SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_SUBTYPE	int	Flag that indicates the attribute name matched the subtype.

APPENDIX A. REVISION HISTORY

Note that revision numbers relate to the edition of this manual, not to version numbers of Red Hat Directory Server.

Revision 10.3-1	Wed Oct 24 2018	Marc Muehlfeld
Red Hat Directory Server 10.3 release of the guide.		
Revision 10.2-1	Tue Apr 10 2018	Marc Muehlfeld
Red Hat Directory Server 10.2 release of the guide.		
Revision 10.1-6	Wed Jan 17 2018	Marc Muehlfeld
Updated parameters of the <code>slapi_mr_indexer_create()</code> function.		
Revision 10.1-5	Mon Nov 06 2017	Marc Muehlfeld
Moved the <i>Setting Replication Session Hooks</i> section from the <i>Admin Guide</i> to this guide.		
Revision 10.1-4	Tue Aug 01 2017	Marc Muehlfeld
Red Hat Directory Server 10.1.1 release of the guide.		
Revision 10.1-3	Fri Feb 24 2017	Marc Muehlfeld
Added "Parameters for LDBM Back End Transaction Pre- and Post-Operation Functions" chapter, enhanced "Registering Extended Operation Functions" section.		
Revision 10.1-2	Wed Dec 14 2016	Marc Muehlfeld
Red Hat Directory Server 10.1 release of the guide.		
Revision 10.0-0	Tue Jun 09 2015	Tomáš Čapek
Red Hat Directory Server 10 release of the guide.		