

Sun Java System Directory Server Enterprise Edition 6.1 Developer's Guide



Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820-0381
June, 2007

Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

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

This distribution may include materials developed by third parties.

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

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

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

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

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

Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs brevets américains ou des applications de brevet en attente aux Etats-Unis et dans d'autres pays.

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

Certains composants de ce produit peuvent être dérivés du logiciel Berkeley BSD, licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays; elle est licenciée exclusivement par X/Open Company, Ltd.

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

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

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

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

Contents

Preface	35
Part I Directory Server Plug-In API Guide	45
1 Before You Start Writing Plug-Ins	47
When to Implement a Server Plug-In	47
Maintaining Plug-Ins	47
Sun Services	48
How Plug-Ins Interact With the Server	48
Example Uses	53
Where to Go From Here	54
Prepare Your Development Environment	54
Learn About Plug-In Development	54
Upgrade Existing Plug-Ins	54
Try a Sample Plug-In	55
Find Details	55
2 Changes to the Plug-In API Since Directory Server 5.2	57
Deprecated and Changed Features Since Directory Server 5.2	57
Attribute Value Handling Changes	57
const Correct Code Changes	58
New Features Since Directory Server 5.2	59
New Distinguished Name Functions	59
New Entry Handling Function	59
New Modification Handling Function	59
New Plug-In Call Ordering Mechanism	59
New Schema Checking Function	59

New Suffix Functions	59
New Syntax Checking Functions	59
New Virtual Attributes Function	60
3 Changes to the Plug-In API From Directory Server 4 to Directory Server 5.2	61
Deprecated and Changed Features From 4 to 5.2	61
Handling Deprecation (4 to 5.2)	62
Plug-In Directive Changes (4 to 5.2)	62
Function Parameter Changes (4 to 5.2)	62
Data Type Changes (4 to 5.2)	62
Plug-In Type Changes (4 to 5.2)	62
Access Control Changes (4 to 5.2)	63
Attribute Handling Changes (4 to 5.2)	63
Control Handling Changes (4 to 5.2)	63
Entry Handling Changes (4 to 5.2)	63
Error Handling Changes (4 to 5.2)	64
Extended Operations Function Changes (4 to 5.2)	64
Filter Handling Changes (4 to 5.2)	64
Internal Operation Changes (4 to 5.2)	64
Logging Function Changes (4 to 5.2)	65
Parameter Block Argument Changes (4 to 5.2)	65
Password Handling Changes (4 to 5.2)	68
SASL Bind Changes (4 to 5.2)	68
New Features From Directory Server 4 to 5.2	68
New Plug-In API Version 3 (4 to 5.2)	68
New Plug-In Types (4 to 5.2)	68
New Plug-In Configuration Entries (4 to 5.2)	68
New NSPR 4 API	69
New Attribute Handling Functions and Flags (4 to 5.2)	69
New Backend Handling Functions (4 to 5.2)	70
New Control Handling Functions (4 to 5.2)	70
New Opaque Data Structures (4 to 5.2)	70
New Distinguished Name Handling Functions (4 to 5.2)	71
New Entry Handling Functions and Flags (4 to 5.2)	71
New Filter Handling Functions (4 to 5.2)	72

New Internal Operation Functions (4 to 5.2)	73
New Memory Management Functions (4 to 5.2)	73
New Modification Structure Functions (4 to 5.2)	74
New Object Extensions (4 to 5.2)	75
New Operation Handling Functions (4 to 5.2)	76
New Parameter Block Arguments (4 to 5.2)	76
New Relative Distinguished Name Handling Functions (4 to 5.2)	77
New UTF8 Encoding Functions (4 to 5.2)	77
New Value Handling Functions (4 to 5.2)	78
New Value Set Handling Functions (4 to 5.2)	78
New Virtual Attribute Functions (4 to 5.2)	79
 4 Getting Started With Directory Server Plug-Ins	81
A Hello World Plug-In	81
Find the Code	81
Review the Plug-In	82
Build the Plug-In	83
Plug In the Plug-In	84
Writing Directory Server Plug-Ins	85
Include the <code>slapi-plugin.h</code> Header File	85
Write Your Plug-In Functions	85
Use Appropriate Return Codes	86
Write an Initialization Function	86
Set Configuration Information Through the Parameter Block	86
Set Pointers to Functions Through the Parameter Block	87
Locate Examples	87
Building Directory Server Plug-Ins	88
Include the Header File for the Plug-In API	88
Link the Plug-In as a Shared Object or Dynamic Link Library	88
Locate the Example Build Rules	89
Plugging Libraries Into Directory Server	90
Specify Plug-In Configuration Settings	90
Modify the Directory Server Configuration	97
Restart Directory Server	97
Logging Plug-In Messages	97

Log Three Levels of Message Severity	98
Set the Appropriate Log Level in the Directory Server Configuration	100
Find Messages in the Log	100
5 Working With Entries Using Plug-Ins	101
Creating Entries	101
Creating New Entries	101
Creating Copies of Entries	102
Converting To and From LDIF Representations	102
Converting an LDIF String to a Slapi_Entry Structure	103
Converting a Slapi_Entry Structure to an LDIF String	104
Getting Entry Attributes and Attribute Values	104
Adding and Removing Attribute Values	106
Adding Attribute Values	106
Removing Attribute Values	108
Verifying Schema Compliance for an Entry	108
Handling Entry Distinguished Names	109
Getting the Parent and Suffix DNs	109
Determining Whether a Suffix Is Served Locally	111
Getting and Setting Entry DNs	111
Normalizing a DN	112
Is the User the Directory Manager?	112
6 Extending Client Request Handling Using Plug-Ins	113
Preoperation and Postoperation Plug-Ins	113
Preoperation Plug-Ins	113
Postoperation Plug-Ins	114
Registration Identifiers	114
Location of Plug-In Examples	116
Extending the Bind Operation	116
▼ To Set Up an Example Suffix	116
Logging the Authentication Method	118
▼ To Register the Plug-In	119
▼ To Generate a Bind Log Message	119
Bypassing Bind Processing in Directory Server	120

Normal Directory Server Bind Behavior	120
Extending the Search Operation	120
Logging Who Requests a Search	120
Breaking Down a Search Filter	122
Extending the Compare Operation	126
Extending the Add Operation	128
Prepending a String to an Attribute	128
Logging the Entry to Add	131
Extending the Modify Operation	133
Extending the Rename Operation	136
Extending the Delete Operation	138
Intercepting Information Sent to the Client	140
7 Handling Authentication Using Plug-Ins	143
How Authentication Works	143
Support for Standard Methods	143
Client Identification During the Bind	144
Bind Processing in Directory Server	144
How a Plug-In Modifies Authentication	146
Bypassing Authentication	146
Using Custom SASL Mechanisms	146
Developing a Simple Authentication Plug-In	146
Locating the Simple Authentication Example	146
Seeing the Plug-In Work	151
▼ To Set Up an Example Suffix	152
▼ To Register the Plug-In	153
▼ To Bypass the Plug-In	153
▼ To Bind as an Example.com User	154
Developing a SASL Authentication Plug-In	155
Locating SASL Examples	155
Registering the SASL Mechanism	155
Developing the SASL Client	158
Trying the SASL Client	160

8	Performing Internal Operations With Plug-Ins	163
	Using Internal Operations	163
	When to Use Internal Operations	163
	Issues With Internal Operations	164
	Finding the Internal Operations Example	164
	Before Using the Internal Operations Example	164
	▼ To Set Up an Example Suffix	164
	Internal Add	166
	Internal Modify	168
	Internal Rename and Move (Modify DN)	170
	Internal Search	171
	Internal Delete	174
9	Writing Entry Store and Entry Fetch Plug-Ins	175
	Calling Entry Store and Entry Fetch Plug-Ins	175
	Using LDIF String Parameters	175
	Locating the Entry Store and Entry Fetch Examples	176
	Writing a Plug-In to Encrypt Entries	177
	Registering Entry Store and Entry Fetch Plug-Ins	178
	Trying the Entry Store and Entry Fetch Examples	179
	▼ To Set Up an Example Suffix	179
10	Writing Extended Operation Plug-Ins	183
	Calling Extended Operation Plug-Ins	183
	Implementing an Extended Operation Plug-In	184
	Locating the Extended Operation Examples	184
	Example Extended Operation Plug-In	185
	Developing the Extended Operation Client	188
	Trying the Extended Operation Example	190
11	Writing Matching Rule Plug-Ins	193
	How Matching Rule Plug-Ins Work	193
	What a Matching Rule Is	194
	Requesting a Matching Rule	194

What a Matching Rule Plug-In Does	194
Example Matching Rule Plug-In	196
Configuring Matching Rule Plug-Ins	196
Registering Matching Rule Plug-Ins	196
Handling Extensible Match Filters	199
How Directory Server Handles Extensible Match Searches	199
Filter Matching Function	200
Filter Index Function	203
Filter Factory Function	206
Filter Object Destructor Function	210
Indexing Entries According to a Matching Rule	211
How Directory Server Handles the Index	211
Indexer Function	213
Indexer Factory Function	215
Indexer Object Destructor Function	218
Enabling Sorting According to a Matching Rule	219
How Directory Server Performs Sorting According to a Matching Rule	219
Handling an Unknown Matching Rule	219
Internal List of Correspondences	219
OIDs Not in the Internal List	220
12 Writing Password Storage Scheme Plug-Ins	223
Calling Password Storage Scheme Plug-Ins	223
Types of Password Storage Scheme Plug-Ins	223
Preinstalled Schemes	224
Effect on Password Attribute Values	224
Invocation for Add and Modify Requests	224
Invocation for Bind Requests	224
Part of a Password Policy	225
Writing a Password Storage Scheme Plug-In	225
Encoding a Password	225
Comparing a Password	227
Registering the Password Storage Scheme Plug-In	227
Setting Up the Password Storage Scheme Plug-In	229
▼ To Register the Plug-In	229

▼ To Set Up an Example Suffix	229
Trying the Password Storage Scheme Example	230
13 Writing Password Quality Check Plug-Ins	233
How Directory Server Uses Password Quality Check Plug-Ins	233
Password Policy to Check Password Quality	233
Whether to Implement a Password Check Plug-In	234
What a Password Check Plug-In Must Do	234
Writing a Custom Password Quality Check Plug-In	235
Checking Password Values	235
Initializing the Password Check Plug-In	236
Testing the Password Check Plug-In	237
▼ To Set Up an Example Suffix	237
▼ To Register the Plug-In	239
▼ To Use the Password Check Plug-In	239
14 Writing Computed Attribute Plug-Ins	241
Computed Attributes and Performance	241
Writing a Computed Attribute Plug-In	242
Initializing the Computed Attribute Plug-In	242
Computing an Attribute Value	242
Testing the Computed Attribute Plug-In	244
▼ To Register the Plug-In	244
▼ To use the Computed Attribute Plug-In	244
Part II Directory Server Plug-In API Reference	245
15 Data Type and Structure Reference	247
Quick Reference Tables	247
Data Types and Structures Alphabetically	249
berval	249
computed_attr_context	249
LDAPControl	250
LDAPMod	250

mrFilterMatchFn	252
plugin_referral_entry_callback	253
plugin_result_callback	254
plugin_search_entry_callback	255
roles_get_scope_fn_type	256
send_ldap_referral_fn_ptr_t	256
send_ldap_result_fn_ptr_t	257
send_ldap_search_entry_fn_ptr_t	258
Slapi_Attr	259
Slapi_Backend	259
Slapi_ComponentId	260
slapi_compute_callback_t	261
slapi_compute_output_t	262
Slapi_CondVar	262
Slapi_Connection	263
Slapi_DN	263
Slapi_Entry	263
slapi_extension_constructor_fnptr	264
slapi_extension_destructor_fnptr	264
Slapi_Filter	265
Slapi_MatchingRuleEntry	265
Slapi_Mod	265
Slapi_Mods	266
Slapi_Mutex	266
Slapi_Operation	267
Slapi_PBlock	267
Slapi_PluginDesc	268
slapi_plugin_init_fnptr	269
Slapi_RDN	269
Slapi_Value	270
Slapi_ValueSet	270
vattr_type_thang	270
16 Function Reference, Part I	273
Functions by Functional Area	273

Function Descriptions, Part I	291
Functions Alphabetically, Part 1	292
slapi_access_allowed()	292
slapi_acl_check_mods()	294
slapi_acl_verify_aci_syntax()	296
slapi_add_entry_internal_set_pb()	297
slapi_add_internal_pb()	298
slapi_add_internal_set_pb()	299
slapi_attr_add_value()	300
slapi_attr_basetype()	301
slapi_attr_dup()	301
slapi_attr_first_value_const()	302
slapi_attr_flag_is_set()	303
slapi_attr_free()	304
slapi_attr_get_bervals_copy()	305
slapi_attr_get_flags()	305
slapi_attr_get_numvalues()	306
slapi_attr_get_oid_copy()	307
slapi_attr_get_plugin()	308
slapi_attr_get_type()	308
slapi_attr_get_valueset()	309
slapi_attr_init()	310
slapi_attr_new()	311
slapi_attr_next_value_const()	311
slapi_attr_syntax_normalize()	312
slapi_attr_type_cmp()	313
slapi_attr_types_equivalent()	314
slapi_attr_value_cmp()	315
slapi_attr_value_find()	315
slapi_be_exist()	316
slapi_be_get_name()	317
slapi_be_get_readonly()	317
slapi_be_getsuffix()	318
slapi_be_gettype()	319
slapi_be_is_flag_set()	319
slapi_be_issuffix()	320

slapi_be_logchanges()	321
slapi_be_private()	321
slapi_be_select()	322
slapi_be_select_by_instance_name()	322
slapi_berval_cmp()	323
slapi_build_control()	324
slapi_build_control_from_berval()	325
slapi_ch_array_free()	326
slapi_ch_bvdup()	327
slapi_ch_bvecdup()	328
slapi_ch_calloc()	328
slapi_ch_free()	329
slapi_ch_free_string()	330
slapi_ch_malloc()	331
slapi_ch_realloc()	332
slapi_ch_strdup()	332
slapi_compute_add_evaluator()	333
slapi_compute_add_search_rewriter_ex()	334
slapi_control_present()	335
slapi_delete_internal_pb()	336
slapi_delete_internal_set_pb()	337
slapi_destroy_condvar()	338
slapi_destroy_mutex()	338
slapi_dn_beparent()	339
slapi_dn_ignore_case()	339
slapi_dn_isbesuffix()	340
slapi_dn_isbesuffix_norm()	341
slapi_dn_isparent()	341
slapi_dn_isroot()	342
slapi_dn_issuffix()	342
slapi_dn_normalize()	343
slapi_dn_normalize_case()	344
slapi_dn_normalize_to_end()	344
slapi_dn_parent()	345
slapi_dn_plus_rdn()	346
slapi_dup_control()	346

slapi_entry2mods()	347
slapi_entry2str()	348
slapi_entry2str_with_options()	349
slapi_entry_add_rdn_values()	351
slapi_entry_add_string()	352
slapi_entry_add_value()	352
slapi_entry_add_values_sv()	353
slapi_entry_add_valueset()	354
slapi_entry_alloc()	355
slapi_entry_attr_delete()	356
slapi_entry_attr_find()	356
slapi_entry_attr_get_charptr()	357
slapi_entry_attr_get_int()	358
slapi_entry_attr_get_long()	358
slapi_entry_attr_get_uint()	359
slapi_entry_attr_get_ulong()	359
slapi_entry_attr_hasvalue()	360
slapi_entry_attr_merge_sv()	360
slapi_entry_attr_replace_sv()	361
slapi_entry_attr_set_charptr()	362
slapi_entry_attr_set_int()	363
slapi_entry_attr_set_long()	363
slapi_entry_attr_set_uint()	364
slapi_entry_attr_set_ulong()	364
slapi_entry_delete_string()	365
slapi_entry_delete_values_sv()	365
slapi_entry_dup()	366
slapi_entry_first_attr()	367
slapi_entry_free()	368
slapi_entry_get_dn()	369
slapi_entry_get_dn_const()	369
slapi_entry_get_ndn()	370
slapi_entry_get_sdn()	370
slapi_entry_get_sdn_const()	371
slapi_entry_get_uniqueid()	372
slapi_entry_has_children()	372

slapi_entry_init()	373
slapi_entry_isroot()	374
slapi_entry_merge_values_sv()	375
slapi_entry_next_attr()	376
slapi_entry_rdn_values_present()	376
slapi_entry_schema_check()	377
slapi_entry_schema_check_ext()	378
slapi_entry_set_dn()	378
slapi_entry_set_sdn()	379
slapi_entry_size()	380
slapi_entry_syntax_check()	381
slapi_entry_vattr_find()	381
slapi_filter_compare()	382
slapi_filter_free()	383
slapi_filter_get_attribute_type()	384
slapi_filter_get_ava()	385
slapi_filter_get_choice()	386
slapi_filter_get_subfilt()	387
slapi_filter_get_type()	388
slapi_filter_join()	389
slapi_filter_list_first()	390
slapi_filter_list_next()	391
slapi_filter_test()	392
slapi_filter_test_ext()	393
slapi_filter_test_simple()	394
slapi_find_matching_paren()	395
slapi_free_search_results_internal()	396
slapi_free_suffix_list()	396
slapi_get_first_backend()	397
slapi_get_object_extension()	398
slapi_get_next_backend()	398
slapi_get_suffix_list()	399
slapi_get_supported_controls_copy()	400
slapi_get_supported_extended_ops_copy()	401
slapi_get_supported_saslmechanisms_copy()	402
slapi_has8thBit()	402

slapi_is_rootdse()	403
slapi_is_root_suffix()	403
slapi_ldap_init()	404
slapi_ldap_unbind()	405
slapi_ldapmods_syntax_check()	406
slapi_lock_mutex()	407
slapi_log_error_ex()	407
slapi_log_info_ex()	409
slapi_log_warning_ex()	411
slapi_matchingrule_free()	413
slapi_matchingrule_get()	414
slapi_matchingrule_new()	415
slapi_matchingrule_register()	415
slapi_matchingrule_set()	416
slapi_mod_add_value()	417
slapi_mod_done()	418
slapi_mod_dump()	418
slapi_mod_free()	419
slapi_mod_get_first_value()	419
slapi_mod_get_ldapmod_byref()	420
slapi_mod_get_ldapmod_passout()	421
slapi_mod_get_next_value()	421
slapi_mod_get_num_values()	422
slapi_mod_get_operation()	423
slapi_mod_get_type()	423
slapi_mod_init()	424
slapi_mod_init_byref()	425
slapi_mod_init_byval()	425
slapi_mod_init_passin()	426
slapi_mod_isvalid()	427
slapi_mod_new()	427
slapi_mod_remove_value()	428
slapi_mod_set_operation()	429
slapi_mod_set_type()	429
slapi_moddn_get_newdn()	430

17 Function Reference, Part II	431
Functions Alphabetically, Part 1	431
slapi_modify_internal_pb()	431
slapi_modify_internal_set_pb()	432
slapi_modrdn_internal_pb()	433
slapi_mods2entry()	434
slapi_mods_add()	435
slapi_mods_add_ldapmod()	436
slapi_mods_add_mod_values()	437
slapi_mods_add_modbvps()	438
slapi_mods_add_smod()	439
slapi_mods_add_string()	439
slapi_mods_done()	440
slapi_mods_dump()	441
slapi_mods_free()	442
slapi_mods_get_first_mod()	442
slapi_mods_get_first_smod()	443
slapi_mods_get_ldapmods_byref()	443
slapi_mods_get_ldapmods_passout()	444
slapi_mods_get_next_mod()	445
slapi_mods_get_next_smod()	446
slapi_mods_get_num_mods()	447
slapi_mods_init()	447
slapi_mods_init_byref()	448
slapi_mods_init_passin()	448
slapi_mods_insert_after()	449
slapi_mods_insert_at()	450
slapi_mods_insert_before()	451
slapi_mods_insert_smod_at()	452
slapi_mods_insert_smod_before()	452
slapi_mods_iterator_backone()	453
slapi_mods_new()	454
slapi_mods_remove()	455
slapi_mods_remove_at()	455
slapi_mr_filter_index()	456
slapi_mr_indexer_create()	456

slapi_new_condvar()	457
slapi_new_mutex()	458
slapi_notify_condvar()	459
slapi_op_abandoned()	460
slapi_op_get_type()	460
slapi_pblock_destroy()	461
slapi_pblock_get()	462
slapi_pblock_new()	464
slapi_pblock_set()	464
slapi_pw_find_sv()	466
slapi_pw_find_valueset()	467
slapi_rdn_add()	468
slapi_rdn_compare()	469
slapi_rdn_contains()	469
slapi_rdn_contains_attr()	470
slapi_rdn_done()	471
slapi_rdn_free()	472
slapi_rdn_get_first()	472
slapi_rdn_get_index()	473
slapi_rdn_get_index_attr()	474
slapi_rdn_get_next()	475
slapi_rdn_get_num_components()	476
slapi_rdn_get_rdn()	476
slapi_rdn_init()	477
slapi_rdn_init_dn()	477
slapi_rdn_init_rdn()	478
slapi_rdn_init_sdn()	479
slapi_rdn_isempty()	479
slapi_rdn_new()	480
slapi_rdn_new_dn()	481
slapi_rdn_new_rdn()	481
slapi_rdn_new_sdn()	482
slapi_rdn_remove()	483
slapi_rdn_remove_attr()	484
slapi_rdn_remove_index()	485
slapi_rdn_set_dn()	485

slapi_rdn_set_rdn()	486
slapi_rdn_set_sdn()	487
slapi_rdn_syntax_check()	487
slapi_register_object_extension()	488
slapi_register_plugin()	489
slapi_register_role_get_scope()	490
slapi_register_supported_control()	491
slapi_register_supported_saslmechanism()	492
slapi_rename_internal_set_pb()	493
slapi_role_check()	494
slapi_role_get_scope()	495
slapi_sdn_add_rdn()	495
slapi_sdn_compare()	496
slapi_sdn_copy()	497
slapi_sdn_done()	497
slapi_sdn_dup()	498
slapi_sdn_free()	498
slapi_sdn_get_backend_parent()	499
slapi_sdn_get_dn()	500
slapi_sdn_get_ndn()	500
slapi_sdn_get_ndn_len()	501
slapi_sdn_get_parent()	502
slapi_sdn_get_rdn()	502
slapi_sdn_get_suffix()	503
slapi_sdn_isempty()	504
slapi_sdn_isgrandparent()	504
slapi_sdn_isparent()	505
slapi_sdn_issuffix()	506
slapi_sdn_new()	506
slapi_sdn_new_dn_byref()	507
slapi_sdn_new_dn_byval()	508
slapi_sdn_new_dn_passin()	509
slapi_sdn_new_ndn_byref()	510
slapi_sdn_new_ndn_byval()	511
slapi_sdn_scope_test()	511
slapi_sdn_set_dn_byref()	512

slapi_sdn_set_dn_byval()	513
slapi_sdn_set_dn_passin()	514
slapi_sdn_set_ndn_byref()	515
slapi_sdn_set_ndn_byval()	515
slapi_sdn_set_parent()	516
slapi_sdn_set_rdn()	517
slapi_search_internal_callback_pb()	518
slapi_search_internal_get_entry()	520
slapi_search_internal_pb()	521
slapi_search_internal_set_pb()	521
slapi_send_ldap_referral()	523
slapi_send_ldap_result()	524
slapi_send_ldap_search_entry()	526
slapi_set_object_extension()	527
slapi_str2entry()	528
slapi_str2filter()	529
slapi_unlock_mutex()	530
slapi_UTF-8CASECMP()	531
slapi_UTF-8NCASECMP()	532
slapi_UTF-8ISLOWER()	533
slapi_UTF-8ISUPPER()	533
slapi_UTF-8STRTOLOWER()	534
slapi_UTF-8STRTOUPPER()	534
slapi_UTF-8TOLOWER()	535
slapi_UTF-8TOUPPER()	536
slapi_value_compare()	536
slapi_value_done()	537
slapi_value_dup()	537
slapi_value_free()	538
slapi_valuearray_free()	539
slapi_value_get_berval()	539
slapi_value_get_int()	540
slapi_value_get_length()	541
slapi_value_get_long()	541
slapi_value_get_string()	542
slapi_value_get_uint()	543

slapi_value_get_ulong()	544
slapi_value_init()	544
slapi_value_init_berval()	545
slapi_value_init_string()	546
slapi_value_init_string_passin()	547
slapi_value_new()	547
slapi_value_new_berval()	548
slapi_value_new_string()	549
slapi_value_new_string_passin()	550
slapi_value_new_value()	551
slapi_value_set()	552
slapi_value_set_berval()	553
slapi_value_set_int()	554
slapi_value_set_string()	555
slapi_value_set_string_passin()	556
slapi_value_set_value()	557
slapi_valueset_add_value_optimised()	557
slapi_valueset_count()	558
slapi_valueset_done()	559
slapi_valueset_find_const()	559
slapi_valueset_first_value_const()	560
slapi_valueset_free()	561
slapi_valueset_init()	562
slapi_valueset_new()	562
slapi_valueset_next_value_const()	563
slapi_valueset_set_from_smod()	564
slapi_valueset_set_valueset_optimised()	565
slapi_vattr_attr_free()	566
slapi_vattr_attrs_free()	566
slapi_vattr_filter_test()	567
slapi_vattr_is_registered()	568
slapi_vattr_is_virtual()	569
slapi_vattr_list_attrs()	569
slapi_vattr_value_compare()	570
slapi_vattr_values_free()	571
slapi_vattr_values_get_ex()	572

slapi_vattr_values_type_thang_get()	573
slapi_wait_condvar()	575
18 Parameter Block Reference	577
Parameter Categories	577
Access Log	578
Add	578
Backend Information	579
Bind	579
Compare	580
Connection Information	580
Delete	581
Directory Configuration Information	582
Extended Operations	582
Internal Operations	583
Modify	583
Operation Information	583
Plug-In Registration	584
Password Verification	586
Post-Operation Entry Access	586
Startup and Shutdown	586
Extended Operations	587
Internal Postoperation	587
Internal Preoperation	587
Entry Storage and Retrieval	588
Matching Rules	588
Postoperation	590
Preoperation	591
One-Way and Reversible Password Storage	592
Rename (Modify RDN)	593
Results	593
Search	594
A NameFinder Application	597
Prerequisite Software	597

Deploying NameFinder	598
▼ To Deploy on Application Server	598
▼ To Deploy on Web Server	598
Configuring NameFinder to Access Your Directory	599
▼ To Configure Access When Using Application Server	599
▼ To Configure Access When Using Web Server	600
Customizing NameFinder	602
Connection Properties	602
Search Attribute Properties	602
Other Properties	603
Index	605

Figures

FIGURE 1-1	Client Request Processing	49
FIGURE 1-2	Plug-In Entry Points	51
FIGURE 1-3	Multiple Plug-Ins in a Shared Library	52
FIGURE 11-1	Directory Server Performing an Extensible Match Search	200
FIGURE 11-2	Filter Match Function Context	201
FIGURE 11-3	Filter Factory Function Context	206
FIGURE 11-4	Directory Server Maintaining an Index Using a Matching Rule	212
FIGURE 11-5	Indexer Function Context	213
FIGURE 11-6	Indexer Factory Function Context	216
FIGURE 11-7	Finding a Matching Rule Plug-In for an Unknown OID	221

Tables

TABLE 1-1	Server Plug-In Types	50
TABLE 1-2	Plug-In Example Uses by Type	53
TABLE 2-1	Replacement Functions for Handling Attribute Values	58
TABLE 3-1	Replacement Functions for Handling Attributes	63
TABLE 3-2	Replacement Functions for Handling Entries	63
TABLE 3-3	Replacement Functions for Handling Internal Operations	65
TABLE 3-4	Replacement Functions for Logging	65
TABLE 3-5	Replacement Parameter Block Arguments	65
TABLE 3-6	New Flags for Handling Attributes	69
TABLE 3-7	New Functions for Managing Memory	73
TABLE 4-1	Plug-In Configuration Settings	90
TABLE 11-1	Functions Defined in Matching Rule Plug-Ins	195
TABLE 15-1	Quick Reference to Data Structures	247
TABLE 15-2	Quick Reference to Callbacks	248
TABLE 15-3	berval Fields	249
TABLE 15-4	LDAPControl Fields	250
TABLE 15-5	LDAPMod Fields	251
TABLE 15-6	mrFilterMatchFn Parameters	253
TABLE 15-7	plugin_referral_entry_callback Parameters	254
TABLE 15-8	plugin_result_callback Parameters	255
TABLE 15-9	plugin_search_entry_callback Parameters	255
TABLE 15-10	roles_get_scope_fn_type Parameters	256
TABLE 15-11	send_ldap_referral_fn_ptr_t Parameters	257
TABLE 15-12	send_ldap_result_fn_ptr_t Parameters	258
TABLE 15-13	send_ldap_search_entry_fn_ptr_t Parameters	259
TABLE 15-14	slapi_compute_callback_t Parameters	261
TABLE 15-15	slapi_compute_output_t Parameters	262
TABLE 15-16	slapi_extension_constructor_fnptr Parameters	264

TABLE 15-17	slapi_extension_destructor_fnptr Parameters	265
TABLE 15-18	Slapi_PluginDesc Fields	268
TABLE 15-19	slapi_plugin_init_fnptr Parameter	269
TABLE 16-1	Functions for Handling Parameter Blocks	273
TABLE 16-2	Functions for Handling Memory	273
TABLE 16-3	Functions for Handling Access Control	274
TABLE 16-4	Functions for Handling Attributes	274
TABLE 16-5	Functions for Handling Basic Encoding Rule Values	275
TABLE 16-6	Functions for Handling Controls	275
TABLE 16-7	Functions for Handling Distinguished Name Strings	276
TABLE 16-8	Functions for Handling Entries	276
TABLE 16-9	Functions for Handling Extended Operations	279
TABLE 16-10	Functions for Handling Filters	279
TABLE 16-11	Functions for Handling Internal Operations	279
TABLE 16-12	Functions for Handling Matching Rules	280
TABLE 16-13	Functions for Handling Modifications	281
TABLE 16-14	Functions for Handling Operations	284
TABLE 16-15	Functions for Handling Passwords	284
TABLE 16-16	Functions for Handling Roles	284
TABLE 16-17	Functions for Handling SASL Mechanisms	284
TABLE 16-18	Functions for Handling Slapi_Backend Structures	285
TABLE 16-19	Functions for Handling Slapi_DN Structures	285
TABLE 16-20	Functions for Handling Slapi_RDN Structures	287
TABLE 16-21	Functions for Handling Slapi_Value Structures	288
TABLE 16-22	Functions for Handling Slapi_ValueSet Structures	289
TABLE 16-23	Functions for Handling UTF-8 Strings	290
TABLE 16-24	Functions for Writing Log Messages	291
TABLE 16-25	Functions for Handling Virtual Attributes	291
TABLE 16-26	Functions for Sending Entries, Referrals, and Results to Clients	291
TABLE 16-27	Function for Registering Plug-Ins	291
TABLE 18-1	Slapi_PBlock Parameter Categories	577
TABLE 18-2	Access Log Parameters	578
TABLE 18-3	Add Function Parameters	579
TABLE 18-4	Backend Information Parameters	579
TABLE 18-5	Bind Function Parameters	580
TABLE 18-6	Compare Function Parameters	580

TABLE 18-7	Connection Information Parameters	581
TABLE 18-8	Delete Function Parameters	582
TABLE 18-9	Directory Configuration Information Parameters	582
TABLE 18-10	Extended Operation Parameters	582
TABLE 18-11	Internal Operation Parameters	583
TABLE 18-12	Modify Function Parameters	583
TABLE 18-13	Operation Information Parameters	583
TABLE 18-14	Plug-In Information Parameters	585
TABLE 18-15	Password Verification Parameters	586
TABLE 18-16	Post-Operation Entry Access Parameters	586
TABLE 18-17	Generic Function Registration Parameters	586
TABLE 18-18	Extended Operation Registration Parameters	587
TABLE 18-19	Internal Postoperation Registration Parameters	587
TABLE 18-20	Internal Preoperation Registration Parameters	587
TABLE 18-21	Entry Store/Fetch Function Registration Parameters	588
TABLE 18-22	Matching Rule Function and Argument Registration Parameters	588
TABLE 18-23	Postoperation Function Registration Parameters	590
TABLE 18-24	Preoperation Function Registration Parameters	591
TABLE 18-25	Password Storage Function Registration Parameters	592
TABLE 18-26	Rename (Modify RDN) Function Parameters	593
TABLE 18-27	Result Parameters	594
TABLE 18-28	Search Function Parameters	594

Examples

EXAMPLE 4-1	Hello, World! Plug-In (<code>hello.c</code>)	82
EXAMPLE 4-2	64-bit: Makefile for a Solaris Sample Plug-In Library	88
EXAMPLE 4-3	32-bit: Makefile for a Solaris Sample Plug-In Library	89
EXAMPLE 4-4	Plug-In Using Arguments (<code>testextendedop.c</code>)	95
EXAMPLE 4-5	Logging a Fatal Error Message	98
EXAMPLE 4-6	Logging a Warning Message	99
EXAMPLE 5-1	Creating a New Entry (<code>entries.c</code>)	102
EXAMPLE 5-2	LDIF Syntax Representing an Entry	102
EXAMPLE 5-3	Converting To and From LDIF Strings (<code>entries.c</code>)	103
EXAMPLE 5-4	Iterating Through Attributes of an Entry (<code>testpreop.c</code>)	105
EXAMPLE 5-5	Adding String Attribute Values (<code>entries.c</code>)	106
EXAMPLE 5-6	Merging Attribute Values (<code>entries.c</code>)	107
EXAMPLE 5-7	Checking Schema Compliance (<code>entries.c</code>)	108
EXAMPLE 5-8	Determining the Parent and Suffix of an Entry (<code>dns.c</code>)	109
EXAMPLE 5-9	Determining Whether a Suffix Is Local (<code>dns.c</code>)	111
EXAMPLE 5-10	Normalizing a DN (<code>dns.c</code>)	112
EXAMPLE 6-1	Registering Postoperation Functions (<code>testpostop.c</code>)	115
EXAMPLE 6-2	Logging the Authentication Method (<code>testpreop.c</code>)	118
EXAMPLE 6-3	Getting the DN of the Client Requesting a Search (<code>testbind.c</code>)	121
EXAMPLE 6-4	Logging Base and Scope for a Search (<code>testpreop.c</code>)	122
EXAMPLE 6-5	Retrieving Filter Information (<code>testpreop.c</code>)	124
EXAMPLE 6-6	Search Filter Breakdown	126
EXAMPLE 6-7	Plug-In Comparison Function (<code>testpreop.c</code>)	126
EXAMPLE 6-8	Obtaining the Attribute Value	127
EXAMPLE 6-9	Performing a Comparison	128
EXAMPLE 6-10	LDIF Entry	129
EXAMPLE 6-11	Prepending ADD to the Description of the Entry (<code>testpreop.c</code>)	129
EXAMPLE 6-12	Searching for the Entry	131

EXAMPLE 6-13	Tracking Added Entries (testpostop.c)	132
EXAMPLE 6-14	Example changelog.txt Entry	133
EXAMPLE 6-15	Tracking Modified Entries (testpostop.c)	134
EXAMPLE 6-16	Checking the Directory for an Entry	134
EXAMPLE 6-17	Modifying a Mail Address	135
EXAMPLE 6-18	Example changelog.txt After Modification	135
EXAMPLE 6-19	Tracking Renamed Entries (testpostop.c)	136
EXAMPLE 6-20	Renaming an Entry	137
EXAMPLE 6-21	Example changelog.txt Entry After Rename	138
EXAMPLE 6-22	Tracking Entry Deletion (testpostop.c)	138
EXAMPLE 6-23	Example changelog.txt After Deletion	139
EXAMPLE 6-24	Logging and Responding to the Client (testpreop.c)	140
EXAMPLE 7-1	Preoperation Bind Function (testbind.c)	147
EXAMPLE 7-2	Registering a Custom SASL Plug-In (testsaslbind.c)	155
EXAMPLE 7-3	Handling the my_sasl_mechanism Bind (testsaslbind.c)	156
EXAMPLE 7-4	Client Using my_sasl_mechanism (clients/saslclient.c)	159
EXAMPLE 8-1	Internal Add Operation (internal.c)	166
EXAMPLE 8-2	Internal Modify Operation (internal.c)	168
EXAMPLE 8-3	Internal Rename or Move Operation (internal.c)	170
EXAMPLE 8-4	Search Callback (internal.c)	172
EXAMPLE 8-5	Internal Search Operation (internal.c)	173
EXAMPLE 9-1	Entry Fetch Scrambler (testentry.c)	176
EXAMPLE 9-2	Entry Fetch UnScrambler (testentry.c)	177
EXAMPLE 9-3	Registering Entry Store and Entry Fetch Plug-Ins (testentry.c)	178
EXAMPLE 9-4	LDIF Representation of an Entry	180
EXAMPLE 9-5	Attribute Values in a Database File Before Scrambling	181
EXAMPLE 9-6	Attribute Values in a Database File After Scrambling	181
EXAMPLE 9-7	Unscrambled Search Results	182
EXAMPLE 10-1	Registering Plug-In Functions and OIDs (testextendedop.c)	185
EXAMPLE 10-2	Client Requesting an Extended Operation (clients/reqextop.c)	188
EXAMPLE 11-1	Registering Matching Rule Factory Functions (matchingrule.c)	196
EXAMPLE 11-2	Registering a Matching Rule (matchingrule.c)	197
EXAMPLE 11-3	Filter Match Function (matchingrule.c)	201
EXAMPLE 11-4	Filter Index Function (matchingrule.c)	203
EXAMPLE 11-5	Filter Factory Function (matchingrule.c)	206
EXAMPLE 11-6	Filter Object and Destructor (matchingrule.c)	210

EXAMPLE 11-7	Indexer Function (matchingrule.c)	214
EXAMPLE 11-8	Indexer Factory Function (matchingrule.c)	216
EXAMPLE 12-1	Encoding a userPassword Value (testpwdstore.c)	226
EXAMPLE 12-2	Comparing a userPassword Value (testpwdstore.c)	227
EXAMPLE 12-3	Registering a Password Storage Scheme Plug-In (testpwdstore.c)	228
EXAMPLE 12-4	Testing the Password Storage Scheme	231
EXAMPLE 12-5	Binding With the New Password	232
EXAMPLE 15-1	Preparing Modifications to an Entry	251
EXAMPLE 15-2	Adding Further Modifications	252
EXAMPLE 15-3	Setting a Plug-In Identifier for Use with Internal Operations	260
EXAMPLE 16-1	Setting a Timeout	404
EXAMPLE 16-2	Logging an Error	408
EXAMPLE 16-3	Logging an Informational Message	410
EXAMPLE 16-4	Logging a Warning	412

Preface

This *Developer's Guide* shows you how to develop directory plug-ins using the APIs provided as part of Sun Java™ System Directory Server Enterprise Edition.

Who Should Use This Book

This guide is intended for developers extending Directory Server and Directory Proxy Server functionality.

Before using this guide, you must be familiar with the following products, programming languages, and technologies:

- Directory Server functionality and the C language, if you plan to develop Directory Server plug-ins
- Directory Proxy Server functionality and the Java language, if you plan to develop Directory Proxy Server plug-ins
- Specifications for LDAP and related protocols, such as DSML v2
- Internet and World Wide Web technologies

Before You Read This Book

Before developing server plug-ins, install at least Directory Server or Directory Proxy Server. See the *Sun Java System Directory Server Enterprise Edition 6.1 Installation Guide* for details.

If you are not yet familiar with this version of Directory Server Enterprise Edition, start by evaluating the new features and capabilities of the product. See the *Sun Java System Directory Server Enterprise Edition 6.1 Evaluation Guide* for details.

How This Book Is Organized

[Part I](#) covers how to develop Directory Server plug-ins, libraries that customize and extend server capabilities.

[Part II](#) offers a complete reference to the Directory SDK for C Directory Server plug-in API.

[Appendix A](#) explains how to install and configure the sample NameFinder web application.

Directory Server Enterprise Edition Documentation Set

This Directory Server Enterprise Edition documentation set explains how to use Sun Java System Directory Server Enterprise Edition to evaluate, design, deploy, and administer directory services. In addition, it shows how to develop client applications for Directory Server Enterprise Edition. The Directory Server Enterprise Edition documentation set is available at <http://docs.sun.com/coll/1224.2>.

For an introduction to Directory Server Enterprise Edition, review the following documents in the order in which they are listed.

TABLE P-1 Directory Server Enterprise Edition Documentation

Document Title	Contents
<i>Sun Java System Directory Server Enterprise Edition 6.1 Release Notes</i>	Contains the latest information about Directory Server Enterprise Edition, including known problems.
<i>Sun Java System Directory Server Enterprise Edition 6.1 Documentation Center</i>	Contains links to key areas of the documentation set.
<i>Sun Java System Directory Server Enterprise Edition 6.1 Evaluation Guide</i>	Introduces the key features of this release. Demonstrates how these features work and what they offer in the context of a fictional deployment that you can implement on a single system.
<i>Sun Java System Directory Server Enterprise Edition 6.1 Deployment Planning Guide</i>	Explains how to plan and design highly available, highly scalable directory services based on Directory Server Enterprise Edition. Presents the basic concepts and principles of deployment planning and design. Discusses the solution life cycle, and provides high-level examples and strategies to use when planning solutions based on Directory Server Enterprise Edition.

TABLE P-1 Directory Server Enterprise Edition Documentation (Continued)

Document Title	Contents
<i>Sun Java System Directory Server Enterprise Edition 6.1 Installation Guide</i>	<p>Explains how to install the Directory Server Enterprise Edition software. Shows how to select which components to install, configure those components after installation, and verify that the configured components function properly.</p> <p>For instructions on installing Directory Editor, go to http://docs.sun.com/coll/DirEdit_05q1.</p> <p>Make sure you read the information in <i>Sun Java System Directory Server Enterprise Edition 6.1 Release Notes</i> concerning Directory Editor before you install Directory Editor.</p>
<i>Sun Java System Directory Server Enterprise Edition 6.1 Migration Guide</i>	Provides instructions for upgrading components from earlier versions of Directory Server, Directory Proxy Server, and Identity Synchronization for Windows.
<i>Sun Java System Directory Server Enterprise Edition 6.1 Administration Guide</i>	<p>Provides command-line instructions for administering Directory Server Enterprise Edition.</p> <p>For hints and instructions on using the Directory Service Control Center, DSCC, to administer Directory Server Enterprise Edition, see the online help provided in DSCC.</p> <p>For instructions on administering Directory Editor, go to http://docs.sun.com/coll/DirEdit_05q1.</p> <p>For instructions on installing and configuring Identity Synchronization for Windows, see Part II, “Installing Identity Synchronization for Windows,” in <i>Sun Java System Directory Server Enterprise Edition 6.1 Installation Guide</i>.</p>
<i>Sun Java System Directory Server Enterprise Edition 6.1 Developer's Guide</i>	Shows how to develop directory client applications with the tools and APIs that are provided as part of Directory Server Enterprise Edition.
<i>Sun Java System Directory Server Enterprise Edition 6.1 Reference</i>	Introduces the technical and conceptual foundations of Directory Server Enterprise Edition. Describes its components, architecture, processes, and features. Also provides a reference to the developer APIs.
<i>Sun Java System Directory Server Enterprise Edition 6.1 Man Page Reference</i>	Describes the command-line tools, schema objects, and other public interfaces that are available through Directory Server Enterprise Edition. Individual sections of this document can be installed as online manual pages.
<i>Sun Java System Directory Server Enterprise Edition 6.1 Troubleshooting Guide</i>	Provides information for defining the scope of the problem, gathering data, and troubleshooting the problem areas using various tools.
<i>Sun Java System Identity Synchronization for Windows 6.0 Deployment Planning Guide</i>	Provides general guidelines and best practices for planning and deploying Identity Synchronization for Windows

Related Reading

The SLAMD Distributed Load Generation Engine (SLAMD) is a Java application that is designed to stress test and analyze the performance of network-based applications. It was originally developed by Sun Microsystems, Inc. to benchmark and analyze the performance of LDAP directory servers. SLAMD is available as an open source application under the Sun Public License, an OSI-approved open source license. To obtain information about SLAMD, go to <http://www.slamd.com/>. SLAMD is also available as a java.net project. See <https://slamd.dev.java.net/>.

Java Naming and Directory Interface (JNDI) technology supports accessing the Directory Server using LDAP and DSML v2 from Java applications. For information about JNDI, see <http://java.sun.com/products/jndi/>. The *JNDI Tutorial* contains detailed descriptions and examples of how to use JNDI. This tutorial is at <http://java.sun.com/products/jndi/tutorial/>.

Directory Server Enterprise Edition can be licensed as a standalone product, as a component of Sun Java Enterprise System, as part of a suite of Sun products, such as the Sun Java Identity Management Suite, or as an add-on package to other software products from Sun. Java Enterprise System is a software infrastructure that supports enterprise applications distributed across a network or Internet environment. If Directory Server Enterprise Edition was licensed as a component of Java Enterprise System, you should be familiar with the system documentation at <http://docs.sun.com/coll/1286.2>.

Identity Synchronization for Windows uses Message Queue with a restricted license. Message Queue documentation is available at <http://docs.sun.com/coll/1307.2>.

Identity Synchronization for Windows works with Microsoft Windows password policies.

- Information about password policies for Windows 2003 is available in the [Microsoft documentation](#) online.
- Information about changing passwords, and about group policies in Windows 2003 is available the [Microsoft documentation](#) online.
- Information about the Microsoft Certificate Services Enterprise Root certificate authority is available in the [Microsoft support documentation](#) online.
- Information about configuring LDAP over SSL on Microsoft systems is available in the [Microsoft support documentation](#) online.

Redistributable Files

Directory Server Enterprise Edition does not provide any files that you can redistribute.

Default Paths and Command Locations

This section explains the default paths used in the documentation, and gives the locations of commands on different operating systems and deployment types.

Default Paths

The table in this section describes the default paths that are used in this document. For full descriptions of the files installed, see also Chapter 14, “Directory Server File Reference,” in *Sun Java System Directory Server Enterprise Edition 6.1 Reference*, Chapter 25, “Directory Proxy Server File Reference,” in *Sun Java System Directory Server Enterprise Edition 6.1 Reference*, or Appendix A, “Directory Server Resource Kit File Reference,” in *Sun Java System Directory Server Enterprise Edition 6.1 Reference*.

TABLE P-2 Default Paths

Placeholder	Description	Default Value
<i>install-path</i>	<p>Represents the base installation directory for Directory Server Enterprise Edition software.</p> <p>The software is installed in directories below this base <i>install-path</i>. For example, Directory Server software is installed in <i>install-path</i>/ds6/.</p>	<p>When you install from a zip distribution using <code>dsee_deploy(1M)</code>, the default <i>install-path</i> is the current directory. You can set the <i>install-path</i> using the <code>-i</code> option of the <code>dsee_deploy</code> command. When you install from a native package distribution, such as you would using the Java Enterprise System installer, the default <i>install-path</i> is one of the following locations:</p> <ul style="list-style-type: none"> ■ Solaris systems - <code>/opt/SUNWdsee/</code>. ■ HP-UX systems - <code>/opt/sun/</code>. ■ Red Hat systems - <code>/opt/sun/</code>. ■ Windows systems - <code>C:\Program Files\Sun\JavaES5\DSEE</code>.
<i>instance-path</i>	<p>Represents the full path to an instance of Directory Server or Directory Proxy Server.</p> <p>The documentation uses <code>/local/ds/</code> for Directory Server and <code>/local/dps/</code> for Directory Proxy Server.</p>	<p>No default path exists. Instance paths must nevertheless always be found on a <i>local</i> file system.</p> <p>The following directories are recommended:</p> <p><code>/var</code> on Solaris systems</p> <p><code>/global</code> if you are using Sun Cluster</p>

TABLE P-2 Default Paths (Continued)

Placeholder	Description	Default Value
<i>serverroot</i>	Represents the parent directory of the Identity Synchronization for Windows installation location	Depends on your installation. Note the concept of a <i>serverroot</i> no longer exists for Directory Server.
<i>isw-hostname</i>	Represents the Identity Synchronization for Windows instance directory	Depends on your installation
<i>/path/to/cert8.db</i>	Represents the default path and file name of the client's certificate database for Identity Synchronization for Windows	<i>current-working-dir/cert8.db</i>
<i>serverroot/isw-hostname/logs/</i>	Represents the default path to the Identity Synchronization for Windows local logs for the System Manager, each connector, and the Central Logger	Depends on your installation
<i>serverroot/isw-hostname/logs/central/</i>	Represents the default path to the Identity Synchronization for Windows central logs	Depends on your installation

Command Locations

The table in this section provides locations for commands that are used in Directory Server Enterprise Edition documentation. To learn more about each of the commands, see the relevant man pages.

TABLE P-3 Command Locations

Command	Java ES, Native Package Distribution	Zip Distribution
cacaoadm	Solaris - <i>/usr/sbin/cacaoadm</i>	Solaris - <i>install-path/dsee6/cacao_2.0/usr/lib/cacao/bin/cacaoadm</i>
	Red Hat, HP-UX - <i>/opt/sun/cacao/bin/cacaoadm</i>	Red Hat, HP-UX - <i>install-path/dsee6/cacao_2.0/cacao/bin/cacaoadm</i>
	Windows - <i>install-path\share\cacao_2.0\bin\cacoadm.bat</i>	Windows - <i>install-path\dsee6\cacao_2.0\bin\cacoadm.bat</i>

TABLE P-3 Command Locations (Continued)

Command	Java ES, Native Package Distribution	Zip Distribution
certutil	Solaris - /usr/sfw/bin/certutil	install-path/dsee6/bin/certutil
	Red Hat, HP-UX - /opt/sun/private/bin/certutil	
dpadm(1M)	install-path/dps6/bin/dpadm	install-path/dps6/bin/dpadm
dpconf(1M)	install-path/dps6/bin/dpconf	install-path/dps6/bin/dpconf
dsadm(1M)	install-path/ds6/bin/dsadm	install-path/ds6/bin/dsadm
dsccon(1M)	install-path/dscc6/bin/dsccmon	install-path/dscc6/bin/dsccmon
dsccreg(1M)	install-path/dscc6/bin/dsccreg	install-path/dscc6/bin/dsccreg
dscctest(1M)	install-path/dscc6/bin/dsccsetup	install-path/dscc6/bin/dsccsetup
dsconf(1M)	install-path/ds6/bin/dsconf	install-path/ds6/bin/dsconf
dsee_deploy(1M)	Not provided	install-path/dsee6/bin/dsee_deploy
dsmig(1M)	install-path/ds6/bin/dsmig	install-path/ds6/bin/dsmig
entrycmp(1)	install-path/ds6/bin/entrycmp	install-path/ds6/bin/entrycmp
fildif(1)	install-path/ds6/bin/fildif	install-path/ds6/bin/fildif
idsktune(1M)	Not provided	At the root of the unzipped zip distribution
insync(1)	install-path/ds6/bin/insync	install-path/ds6/bin/insync
ns-accountstatus(1M)	install-path/ds6/bin/ns-accountstatus	install-path/ds6/bin/ns-accountstatus
ns-activate(1M)	install-path/ds6/bin/ns-activate	install-path/ds6/bin/ns-activate
ns-inactivate(1M)	install-path/ds6/bin/ns-inactivate	install-path/ds6/bin/ns-inactivate
repldisc(1)	install-path/ds6/bin/repldisc	install-path/ds6/bin/repldisc
schema_push(1M)	install-path/ds6/bin/schema_push	install-path/ds6/bin/schema_push
smcwebserver	Solaris, Linux, HP-UX - /usr/sbin/smcwebserver	This command pertains only to Directory Service Control Center when it is installed using native packages distribution.
	Windows - install-path\share\webconsole\bin\smcwebserver	

TABLE P-3 Command Locations (Continued)

Command	Java ES, Native Package Distribution	Zip Distribution
wadmin	Solaris, Linux, HP-UX - /usr/sbin/wadmin	This command pertains only to Directory Service Control Center when it is installed using native packages distribution.
	Windows - install-path\share\ webconsole\bin\wadmin	

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P-4 Typographic Conventions

Typeface	Meaning	Example
AaBbCc123	The names of commands, files, and directories, and onscreen computer output	Edit your . login file. Use <code>ls -a</code> to list all files. machine_name% you have mail.
AaBbCc123	What you type, contrasted with onscreen computer output	machine_name% su Password:
<i>AaBbCc123</i>	A placeholder to be replaced with a real name or value	The command to remove a file is <i>rm filename</i> .
<i>AaBbCc123</i>	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the <i>User's Guide</i> . A <i>cache</i> is a copy that is stored locally. Do <i>not</i> save the file.

Shell Prompts in Command Examples

The following table shows default system prompts and superuser prompts.

TABLE P-5 Shell Prompts

Shell	Prompt
C shell on UNIX and Linux systems	machine_name%
C shell superuser on UNIX and Linux systems	machine_name#

TABLE P-5 Shell Prompts (Continued)

Shell	Prompt
Bourne shell and Korn shell on UNIX and Linux systems	\$
Bourne shell and Korn shell superuser on UNIX and Linux systems	#
Microsoft Windows command line	C:\

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P-6 Symbol Conventions

Symbol	Description	Example	Meaning
[]	Contains optional arguments and command options.	ls [-l]	The -l option is not required.
{ }	Contains a set of choices for a required command option.	-d {y n}	The -d option requires that you use either the y argument or the n argument.
\${ }	Indicates a variable reference.	\${com.sun.javaRoot}	References the value of the com.sun.javaRoot variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (<http://www.sun.com/documentation/>)
- Support (<http://www.sun.com/support/>)
- Training (<http://www.sun.com/training/>)

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Searching Sun Product Documentation

Besides searching for Sun product documentation from the docs.sun.com web site, you can use a search engine of your choice by typing the following syntax in the search field:

search-term site:docs.sun.com

For example, to search for Directory Server, type the following:

"Directory Server" site:docs.sun.com

To include other Sun web sites in your search, such as java.sun.com, www.sun.com, and developers.sun.com, use sun . com in place of docs . sun . com in the search field.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to <http://docs.sun.com> and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 819-0996.



PART I

Directory Server Plug-In API Guide

This part demonstrates how to develop Directory Server plug-ins. This part also explains what has changed since the last release, so you can upgrade existing plug-ins for use with Directory Server Enterprise Edition 6.1.

- [Chapter 1](#) introduces Directory Server plug-ins, and when to develop plug-ins.
- [Chapter 2](#) describes the changes to the plug-in API since the Directory Server 5.2 release.
- [Chapter 3](#) describes changes to the plug-in API between release 4 and release 5.2.
- [Chapter 4](#) gets you started using the plug-in API.
- [Chapter 5](#) shows how to write plug-in code that handles directory entries.
- [Chapter 6](#) shows how to write plug-in code that the server calls before and after processing client requests.
- [Chapter 7](#) shows how to write plug-in code that changes the way authentication is processed.
- [Chapter 8](#) shows how to write plug-in code that performs internal directory requests.
- [Chapter 9](#) shows how to write plug-in code that is called before and after entries are stored in the database.
- [Chapter 10](#) shows how to write plug-in code to handle LDAP v3 extended operations.
- [Chapter 11](#) shows how to write plug-in code for custom matching rules.

- [Chapter 12](#) shows how to write plug-in code that changes the way passwords are stored.
- [Chapter 13](#) shows how to write plug-in code that is used to check password quality.

Before You Start Writing Plug-Ins

This chapter helps you to decide whether to implement a server plug-in. It also explains what to do next if you choose to implement a plug-in.

This chapter covers the following topics:

- “When to Implement a Server Plug-In” on page 47
- “How Plug-Ins Interact With the Server” on page 48
- “Example Uses” on page 53
- “Where to Go From Here” on page 54

When to Implement a Server Plug-In

Many Sun Java System Directory Server native product features rely on *server plug-ins*. Plug-ins are libraries of functions that are registered with the server to perform key parts of specific directory service operations.

The Directory Server plug-in API provides an interface to add server capabilities that are not in the current releases of the product. If you must add server capabilities because a required feature has not yet been implemented, the plug-in API might render that enhancement possible. If you can instead use standard features of the product, avoid creating a plug-in.

Maintaining Plug-Ins

Be aware that creating your own plug-in links your software solution to a specific product release. The plug-in API can evolve from release to release to accommodate new features. As you choose to upgrade to take advantage of new features, you might need to update your Directory Server plug-ins.

You can use the header files with class libraries solely to create and distribute programs to interface with the Software’s APIs. You can also use libraries to create, then distribute program “plug-ins” to interface with the Software’s plug-in APIs. You cannot modify the header files or

libraries. You acknowledge that Sun makes no direct or implied guarantees that the plug-in APIs will be backward-compatible in future releases of the Software. In addition, you acknowledge that Sun is under no obligation to support or to provide upgrades or error corrections to any derivative plug-ins.

When providing technical support, Sun technical support personnel request that you reproduce the problem *after turning your custom plug-ins off*.

Sun Services

Sun Services (<http://www.sun.com/servicessolutions/>) can help you make the right deployment decisions. Sun Services can also help you to deploy the right solution for your specific situation, whether or not the solution includes custom plug-ins.

How Plug-Ins Interact With the Server

All client requests to Directory Server instances undergo particular processing sequences. The exact sequence depends on the operation that the client has requested. Yet overall, the sequences are similar. The way plug-ins handle requests is independent of how the client encodes the request. In other words, client request data appears essentially the same to plug-ins regardless of the client access protocol used to communicate with Directory Server.

In general, client request processing proceeds as follows. First, the Directory Server front end decodes the request. The core server handles the decoded request, calling the necessary back end functions to find or store entries, for example. The primary function of the Directory Server back end is to retrieve and store entries. Unless abandoned, however, a client request results in a response originating in the back end. The front end formats the response and sends it to the client.

The following figure illustrates how client request data moves through Directory Server.

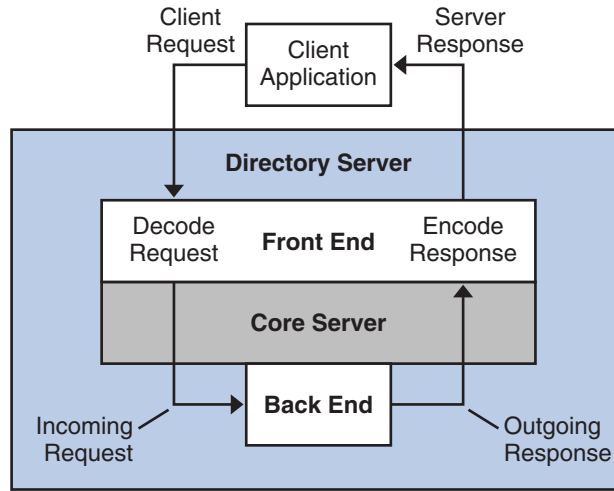


FIGURE 1-1 Client Request Processing

As a rule, plug-ins register functions to be called by Directory Server at specific steps of processing the client request. A plug-in has a specific type, such as preoperation or postoperation. A *preoperation plug-in* registers functions that handle incoming requests before the requests are processed in the server. For example, a preoperation plug-in can register a pre-bind function that is called before the core server begins processing the client bind. The pre-bind function might redefine how the client authenticates to Directory Server, enabling authentication against an external database. A *postoperation plug-in* registers functions that are called after a request is processed. For example, a postoperation plug-in might register a post-modification function that is called to log the information about the modification request. “[Example Uses](#)” on [page 53](#) provides examples for other plug-in types.

The two internal operation plug-in types form an exception to the rule that functions are called to perform client request processing. Internal operation plug-ins implement functions that are triggered for internal Directory Server operations. Internal operations are initiated not by client requests, but by Directory Server or by another plug-in called by Directory Server. Exercise caution when manipulating internal operation data.

The following table summarizes when Directory Server calls documented plug-in function types.

TABLE 1-1 Server Plug-In Types

Plug-In Type	When Called
Entry store-fetch	Immediately before writing a directory entry to or immediately after reading a directory entry from the directory database type: ldbmentryfetchstore
Extended operation	For processing an LDAP v3 extended operation type: extendedop
Internal postoperation	After completing an operation initiated by Directory Server, for which no corresponding client request exists type: internalpostoperation
Internal preoperation	Before performing an operation initiated by Directory Server, for which no corresponding client request exists type: internalpreoperation
Matching rule	When finding search result candidates based on an extensible match filter for search operations only type: matchingrule
Object	Depends on the functions that are registered. (This type is generic and is used to register other types) type: object
Password check	When performing a strong check on a userPassword value to add or modify type: passwordcheck
Password storage scheme	When storing an encoded userPassword value that cannot be decrypted type: pwdstoragescheme
Postoperation	After completing an operation requested by a client type: postoperation
Preoperation	Before performing an operation requested by a client type: preoperation

The following figure illustrates where in the processing sequence Directory Server calls some of the plug-in types described in [Table 1-1](#).

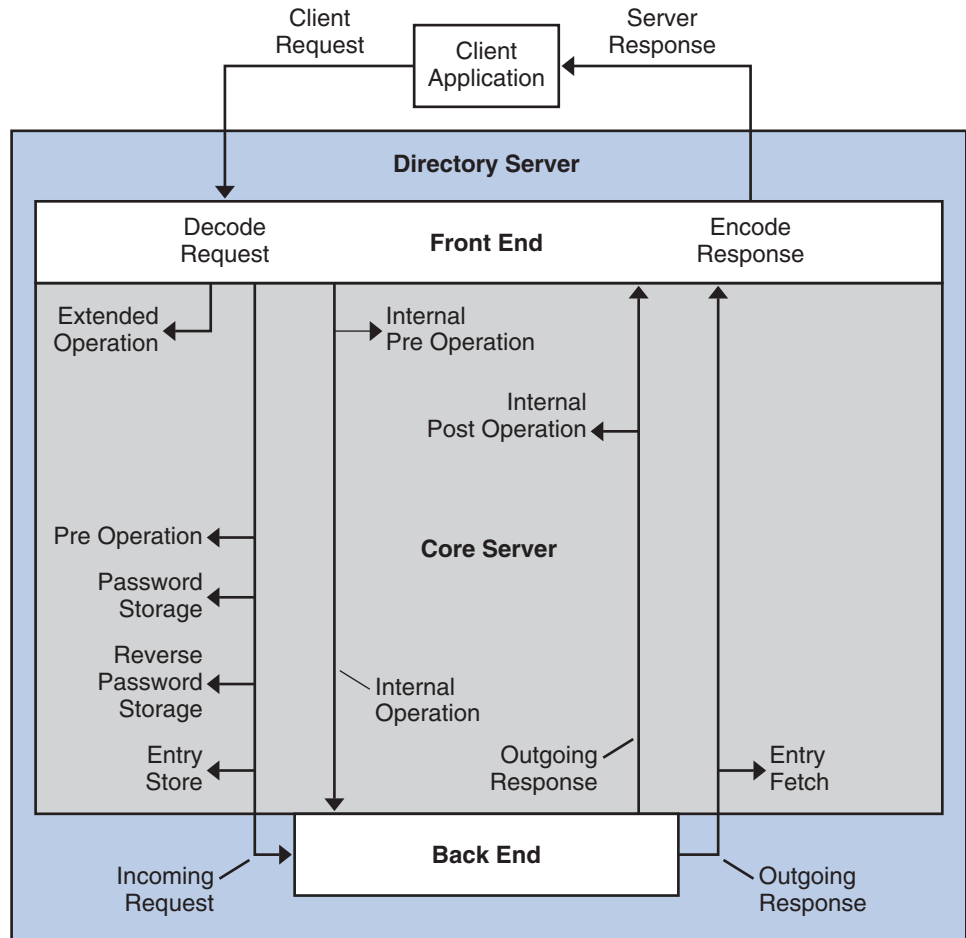


FIGURE 1-2 Plug-In Entry Points

Notice that postoperation return codes do not affect Directory Server processing.

The plug-in type and plug-in configuration information are typically specified statically in a configuration entry.

You can include multiple plug-ins in a single shared library, but you must specify individual, server-specific configuration entries for each plug-in that is registered statically. This means that if you want preoperation and postoperation functions, you typically write two plug-ins. Then you register each plug-in with Directory Server. Both plug-ins can be contained in the same shared library. Both plug-ins can communicate private data across the same operation or connection, as well, using the interface provided. You can also write one plug-in that initializes other plug-ins, if that configuration is required for your deployment.

The sample plug-ins delivered with the product follow the principle of building all plug-ins into one library, then registering plug-ins individually.

The following figure shows multiple plug-ins that reside in the same library.

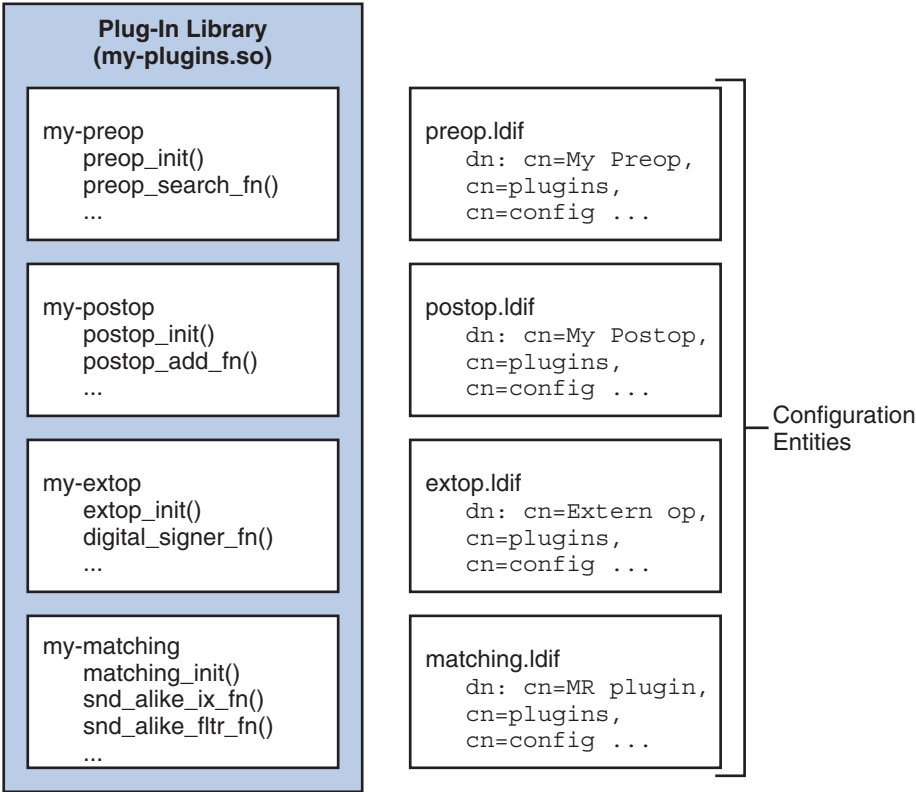


FIGURE 1-3 Multiple Plug-Ins in a Shared Library

Each plug-in includes a registration routine that is called by Directory Server at startup. The registration routine explicitly registers each plug-in function with Directory Server.



Caution – Plug-in libraries are linked with Directory Server. Plug-ins execute in the same memory address space as Directory Server. Plug-ins have direct access to data structures used by Directory Server. As a result, plug-ins can corrupt memory used by Directory Server. Such memory corruption can result in database corruption. Due to this danger, *never* use an untested plug-in on a production Directory Server.

Furthermore, Directory Server runs as a multi threaded process. Code you write for a plug-in must be reentrant. Multiple threads can call your entry point simultaneously. These threads can be rescheduled at any time.

Example Uses

To select the right plug-in type for the job is an art, rather than a science. The following table suggests example uses for documented plug-in types.

TABLE 1-2 Plug-In Example Uses by Type

Plug-In Type	Example Uses
Entry store-fetch	Encoding and decoding entire plug-in entries Auditing or logging each entry as the entry is written to disk
Extended operation	Adding client services that are not available in LDAP v3 such as digital signatures in requests and responses
Internal postoperation	Auditing results of internal operations initiated by another plug-in
Internal preoperation	Preempting internal operations initiated by another plug-in
Matching rule	Offering enhanced sounds-like matching for directory searches
Object	Developing a plug-in that registers a group of other plug-ins with Directory Server
Password check	Forcing new passwords to conform to corporate policy for password syntax
Password storage scheme	Using a custom algorithm for password encryption instead of one of the algorithms supported by the standard product
Postoperation	Associating alerts and alarms sent after particular operations Auditing changes to specific entries Performing cleanup after an operation

TABLE 1-2 Plug-In Example Uses by Type (Continued)

Plug-In Type	Example Uses
Preoperation	Handling custom authentication methods external to the directory
	Forcing syntax checking for attribute values before adding or modifying an entry
	Adding attributes to or deleting attributes from a request
	Preprocessing client request content to translate requests from legacy applications
	Approving or rejecting the content of a client modification request before processing the request

The list of example uses is by no means exhaustive, but is instead intended to help you brainstorm solutions.

Where to Go From Here

Most of the rest of this guide is devoted to examples that demonstrate the specific plug-in types.



Caution – *Never* develop plug-ins on a production server, but instead on a test server used specifically for plug-in development.

Prepare Your Development Environment

You might not have installed Sun Java System Directory Server software. You might not have installed development software for the C language for use during plug-in development. Install the software now. Without development tools and a functioning Directory Server, you cannot use the examples discussed here.

Learn About Plug-In Development

If you have not yet written a plug-in for the current release, refer to [Chapter 4](#). The chapter helps you to build a simple plug-in, and to register the plug-in with Directory Server.

Upgrade Existing Plug-Ins

If you maintain plug-ins developed for a previous release of Directory Server, refer to [Chapter 2](#) and [Chapter 3](#) for information about what has changed.

Try a Sample Plug-In

Examples for several plug-in types are provided with the product in *install-path/examples*. Subsequent chapters demonstrate the use of the sample plug-ins.

Find Details

See [Part II](#) for details about particular data structures, functions, and parameter block semantics.

Changes to the Plug-In API Since Directory Server 5.2

This chapter describes what's new in the Directory Server 6.1 release, that is, the changes to the plug-in API since the Directory Server 5.2 release. If you maintain Directory Server plug-ins originally developed for a previous release, consider upgrade such plug-ins to use the new and updated features.

For reference information, see [Part II](#).

Tip – Consider working with Sun Services consultants to develop and to maintain your Directory Server plug-ins.

This chapter covers the following topics:

- [“Deprecated and Changed Features Since Directory Server 5.2” on page 57](#)
- [“New Features Since Directory Server 5.2” on page 59](#)

Deprecated and Changed Features Since Directory Server 5.2

This section addresses features deprecated or changed since the Directory Server 5.2 release. Where possible, use the replacement functionality.

Attribute Value Handling Changes

The following table shows deprecated functions and their replacements for handling attribute values.

TABLE 2-1 Replacement Functions for Handling Attribute Values

Deprecated Function	Replacement Function
<code>slapi_attr_first_value()</code>	<code>slapi_attr_first_value_const()</code>
<code>slapi_attr_next_value()</code>	<code>slapi_attr_next_value_const()</code>
<code>slapi_valueset_add_value()</code>	<code>slapi_valueset_add_value_optimised()</code>
<code>slapi_valueset_first_value()</code>	<code>slapi_valueset_first_value_const()</code>
<code>slapi_valueset_next_value()</code>	<code>slapi_valueset_next_value_const()</code>
<code>slapi_valueset_set_valueset()</code>	<code>slapi_valueset_set_valueset_optimised()</code>
<code>slapi_valueset_find()</code>	<code>slapi_valueset_find_const()</code>

const Correct Code Changes

The following functions now apply const correctness:

- `slapi_build_control()`
- `slapi_build_control_from_berval()`
- `slapi_control_present()`
- `slapi_dup_control()`
- `slapi_entry2str()`
- `slapi_entry2str_with_options()`
- `slapi_get_account_availability()`
- `slapi_log_error_ex()`
- `slapi_log_info_ex()`
- `slapi_log_warning_ex()`
- `slapi_register_supported_control()`
- `slapi_search_internal_get_entry()`

The following callback data types also now apply const correctness:

- `slapi_pwd_storage_scheme_cmp_fn`
- `slapi_pwd_storage_scheme_dec_fn`
- `slapi_pwd_storage_scheme_enc_fn`

For updated prototypes, see *install-path/ds6/include/slapi-plugin.h*.

New Features Since Directory Server 5.2

This section summarizes features that have been added since the Directory Server 5.2 release. This section does not include features that are reserved for internal use.

New Distinguished Name Functions

The following functions have been added to handle distinguished names (DNs):

```
slapi_dn_is_besuffix_norm()  
slapi_sdn_get_suffix()
```

New Entry Handling Function

The `slapi_entry_isroot()` function has been added to handle entries.

New Modification Handling Function

The `slapi_mods_remove_at()` function has been added to handle modifications.

New Plug-In Call Ordering Mechanism

Directory Server now includes a mechanism for setting the order in which plug-ins are called by the server. See [“Ordering Plug-In Calls” on page 92](#) for details.

New Schema Checking Function

The `slapi_entry_schema_check_ext()` function has been added to handle schema checking.

New Suffix Functions

The following functions have been added to browse supported suffixes:

```
slapi_free_suffix_list()  
slapi_get_suffix_list()
```

New Syntax Checking Functions

The following functions have been added to handle syntax checking:

```
slapi_entry_syntax_check()  
slapi_ldapmods_syntax_check()  
slapi_rdn_syntax_check()
```

New Virtual Attributes Function

The `slapi_vattr_is_virtual()` function has been added to handle virtual attributes.

+

Changes to the Plug-In API From Directory Server 4 to Directory Server 5.2

This chapter describes the changes to the plug-in API from the Directory Server 4 release to and including the Directory Server 5.2 release. If you maintain Directory Server plug-ins originally developed for such releases or for a previous release, upgrade the plug-ins to use the new and updated features.

For reference information, see [Part II](#).

Tip – You can use Sun Services consultants to develop and to maintain your Directory Server plug-ins.

This chapter covers the following topics:

- “Deprecated and Changed Features From 4 to 5.2” on page 61
- “New Features From Directory Server 4 to 5.2” on page 68

Deprecated and Changed Features From 4 to 5.2

This section addresses features deprecated or changed since Directory Server 4 was released up to and including Directory Server 5.2. Where possible, use the replacement functionality.

- “Handling Deprecation (4 to 5.2)” on page 62
- “Plug-In Directive Changes (4 to 5.2)” on page 62
- “Function Parameter Changes (4 to 5.2)” on page 62
- “Data Type Changes (4 to 5.2)” on page 62
- “Plug-In Type Changes (4 to 5.2)” on page 62
- “Access Control Changes (4 to 5.2)” on page 63
- “Attribute Handling Changes (4 to 5.2)” on page 63
- “Control Handling Changes (4 to 5.2)” on page 63
- “Entry Handling Changes (4 to 5.2)” on page 63
- “Error Handling Changes (4 to 5.2)” on page 64

- “Extended Operations Function Changes (4 to 5.2)” on page 64
- “Filter Handling Changes (4 to 5.2)” on page 64
- “Internal Operation Changes (4 to 5.2)” on page 64
- “Logging Function Changes (4 to 5.2)” on page 65
- “Parameter Block Argument Changes (4 to 5.2)” on page 65
- “Password Handling Changes (4 to 5.2)” on page 68
- “SASL Bind Changes (4 to 5.2)” on page 68

For information about a function, see [Part II](#).

Handling Deprecation (4 to 5.2)

SLAPI_DEPRECATED has been defined in `slapi-plugin.h` to highlight what features have been superseded. If you cannot change core plug-in code, you might decide to recompile after including `slapi-plugin-compat4.h` to access deprecated features in this release.

```
#include "slapi-plugin-compat4.h"
```

In general, you must *at minimum* recompile your plug-ins so they work with Directory Server 5. Where possible, replace deprecated code with new alternatives before recompiling and testing.

Plug-In Directive Changes (4 to 5.2)

Plug-in directives have been deprecated. Use configuration entries instead. Refer to “[Plugging Libraries Into Directory Server](#)” on page 90 for details on using configuration entries to load and configure plug-ins.

Function Parameter Changes (4 to 5.2)

Many of the existing function parameters have become `const` to reflect that parameter values are not changed when reading information.

Data Type Changes (4 to 5.2)

Consider using opaque data types wherever possible. For information about data types, see [Chapter 15](#).

Plug-In Type Changes (4 to 5.2)

The `SLAPI_PLUGIN_DATABASE` plug-in type (`database`) has been deprecated.

Access Control Changes (4 to 5.2)

`slapi_acl_verify_aci_syntax()` now takes a pointer to a `Slapi_PBlock` as its first parameter. You must modify plug-in code to account for this change.

Attribute Handling Changes (4 to 5.2)

The following table shows deprecated functions and their replacements for handling attributes.

TABLE 3-1 Replacement Functions for Handling Attributes

Deprecated Function	Replacement Function
<code>slapi_attr_get_values()</code>	<code>slapi_attr_get_bervals_copy()</code>
<code>slapi_attr_get_oid()</code>	<code>slapi_attr_get_oid_copy()</code>
<code>slapi_compute_add_search_rewriter(\$slapi_compute_add_search_rewriter_ex())</code>	

Control Handling Changes (4 to 5.2)

`slapi_get_supported_controls()` is not thread-safe. The function has been deprecated. Use `slapi_get_supported_controls_copy()` instead.

`SLAPI_OPERATION_*` identifiers have changed to unsigned long values.

Entry Handling Changes (4 to 5.2)

The following table shows deprecated functions and their replacements for handling entries.

TABLE 3-2 Replacement Functions for Handling Entries

Deprecated Function	Replacement Function
<code>slapi_entry_add_values()</code>	<code>slapi_entry_add_values_sv</code> <code>()</code> <code>slapi_entry_add_valueset()</code>

TABLE 3-2 Replacement Functions for Handling Entries (Continued)

Deprecated Function	Replacement Function
slapi_entry_attr_hasvalue()	slapi_entry_attr_has_syntax_value()
slapi_entry_attr_merge()	slapi_entry_attr_merge_sv() slapi_entry_merge_values_sv()
slapi_entry_attr_replace()	slapi_entry_attr_replace_sv()
slapi_entry_delete_values()	slapi_entry_delete_values_sv()

Error Handling Changes (4 to 5.2)

slapi-plugin.h now includes ldap_msg.h, which lists unique error numbers used by Directory Server.

Functions for dealing with back ends that return int error codes can now also return SLAPI_FAIL_RETRY. The old values are also still used.

Directory Server now distinguishes between two results. SLAPI_BIND_FAIL indicates that the bind failed and the server prepared a result to send. SLAPI_BIND_ANONYMOUS indicates that the bind succeeded as an anonymous bind.

Extended Operations Function Changes (4 to 5.2)

slapi_get_supported_extended_ops() is not thread-safe. The function has been deprecated. Use slapi_get_supported_extended_ops_copy() instead.

Filter Handling Changes (4 to 5.2)

slapi_filter_get_type() has been deprecated. Use slapi_filter_get_attribute_type() instead. The replacement function works for all simple filter choices.

Internal Operation Changes (4 to 5.2)

The following table shows deprecated functions and their replacements for handling internal operations.

TABLE 3-3 Replacement Functions for Handling Internal Operations

Deprecated Function	Replacement Function
<code>slapi_add_entry_internal()</code>	<code>slapi_add_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_modify_internal()</code>	<code>slapi_modify_internal_pb()</code>
<code>slapi_modrdn_internal()</code>	<code>slapi_modrdn_internal_pb()</code>
<code>slapi_search_internal()</code>	<code>slapi_search_internal_pb()</code>
<code>slapi_search_internal_callback()</code>	<code>slapi_search_internal_callback_pb()</code>

Logging Function Changes (4 to 5.2)

The following table shows deprecated functions and their replacements for logging.

TABLE 3-4 Replacement Functions for Logging

Deprecated Function	Replacement Function
<code>slapi_log_error()</code>	<code>slapi_log_error_ex()</code> <code>slapi_log_info_ex()</code> <code>slapi_log_warning_ex()</code>
<code>slapi_is_loglevel_set()</code>	No replacement is available

Refer to “[Logging Plug-In Messages](#)” on page 97 and all example code delivered with the product for demonstrations of how to use the new logging interface.

Parameter Block Argument Changes (4 to 5.2)

The following table shows deprecated parameter block arguments and their replacement arguments.

TABLE 3-5 Replacement Parameter Block Arguments

Deprecated Argument	Replacement Argument
<code>SLAPI_CHANGENUMBER</code>	No replacement is available

TABLE 3-5 Replacement Parameter Block Arguments (Continued)

Deprecated Argument	Replacement Argument
SLAPI_CONN_AUTHTYPE	SLAPI_CONN_AUTHMETHOD
SLAPI_CONN_CLIENTIP	SLAPI_CONN_CLIENTNETADDR
SLAPI_CONN_SERVERIP	SLAPI_CONN_SERVERNETADDR
SLAPI_LOG_OPERATION	No replacement is available

TABLE 3-5 Replacement Parameter Block Arguments (Continued)

Deprecated Argument	Replacement Argument
SLAPI_PLUGIN_DB_ABANDON_FN	No replacement, as the database plug-in type is deprecated
SLAPI_PLUGIN_DB_ABORT_FN	
SLAPI_PLUGIN_DB_ADD_FN	
SLAPI_PLUGIN_DB_ARCHIVE2DB_FN	
SLAPI_PLUGIN_DB_BEGIN_FN	
SLAPI_PLUGIN_DB_BIND_FN	
SLAPI_PLUGIN_DB_COMMIT_FN	
SLAPI_PLUGIN_DB_COMPARE_FN	
SLAPI_PLUGIN_DB_CONFIG_FN	
SLAPI_PLUGIN_DB_DB2ARCHIVE_FN	
SLAPI_PLUGIN_DB_DB2INDEX_FN	
SLAPI_PLUGIN_DB_DB2LDIF_FN	
SLAPI_PLUGIN_DB_DELETE_FN	
SLAPI_PLUGIN_DB_ENTRY_FN	
SLAPI_PLUGIN_DB_ENTRY_RELEASE_FN	
SLAPI_PLUGIN_DB_FLUSH_FN	
SLAPI_PLUGIN_DB_FREE_RESULT_SET_FN	
SLAPI_PLUGIN_DB_LDIF2DB_FN	
SLAPI_PLUGIN_DB_MODIFY_FN	
SLAPI_PLUGIN_DB_MODRDN_FN	
SLAPI_PLUGIN_DB_NEXT_SEARCH_ENTRY_EXT_FN	
SLAPI_PLUGIN_DB_NEXT_SEARCH_ENTRY_FN	
SLAPI_PLUGIN_DB_NO_ACL	
SLAPI_PLUGIN_DB_REFERRAL_FN	
SLAPI_PLUGIN_DB_RESULT_FN	
SLAPI_PLUGIN_DB_SEARCH_FN	
SLAPI_PLUGIN_DB_SEQ_FN	
SLAPI_PLUGIN_DB_SIZE_FN	
SLAPI_PLUGIN_DB_TEST_FN	
SLAPI_PLUGIN_DB_UNBIND_FN	

TABLE 3-5 Replacement Parameter Block Arguments (Continued)

Deprecated Argument	Replacement Argument
SLAPI_REQUESTOR_ISUPDATEDN	No replacement is available

Password Handling Changes (4 to 5.2)

`slapi_pw_find()` has been deprecated. Use `slapi_pw_find_valueset()` instead.

SASL Bind Changes (4 to 5.2)

`slapi_get_supported_saslmechanisms()` has been deprecated. Use `slapi_get_supported_saslmechanisms_copy()` instead.

New Features From Directory Server 4 to 5.2

This section summarizes features that were added since the Directory Server 4 was released up to and including Directory Server 5. This section does not include features that are reserved for internal use.

For information about a function, see [Chapter 16](#).

New Plug-In API Version 3 (4 to 5.2)

The header file `slapi-plugin.h` identifies the current plug-in API as version 3. Plug-ins supporting API version 3 indicate the version in their description with `SLAPI_PLUGIN_VERSION_03` or currently `SLAPI_PLUGIN_CURRENT_VERSION`.

New Plug-In Types (4 to 5.2)

New plug-in types have been added since the version 4 releases. For a list of supported plug-in types, refer to “[How Plug-Ins Interact With the Server](#)” on [page 48](#).

New Plug-In Configuration Entries (4 to 5.2)

Configuration entries rather than directives are now used to configure Directory Server plug-ins. Refer to “[Plugging Libraries Into Directory Server](#)” on [page 90](#) for details on using configuration entries to load and configure plug-ins.

New NSPR 4 API

The Netscape Portable Runtime (NSPR) API allows compliant applications to use system facilities. The system facilities include threads, thread synchronization, I/O, interval timing, atomic operations, and other low-level services. NSPR allows access to these facilities in a platform-independent manner.

For information about the version of NSPR used currently, refer to <http://www.mozilla.org/projects/nspr/release-notes/>.

New Attribute Handling Functions and Flags (4 to 5.2)

The following table shows new flags defined for use with existing functions for handling attributes.

TABLE 3-6 New Flags for Handling Attributes

Function	Flags Added
<code>slapi_attr_get_flags()</code>	<code>SLAPI_ATTR_FLAG_COLLECTIVE</code> <code>SLAPI_ATTR_FLAG_NOUSERMOD</code> <code>SLAPI_ATTR_FLAG_OBSOLETE</code> <code>SLAPI_ATTR_FLAG_STD_ATTR</code>
<code>slapi_attr_type_cmp()</code>	<code>SLAPI_TYPE_CMP_BASE</code> <code>SLAPI_TYPE_CMP_EXACT</code> <code>SLAPI_TYPE_CMP_SUBTYPE</code>

The old flags are still available. Refer to comments in `slapi-plugin.h` for hints on how to use the flags.

The following functions have been added to handle attributes and their values:

```
slapi_attr_add_value()
slapi_attr_first_value()
slapi_attr_get_numvalues()
slapi_attr_get_valueset()
slapi_attr_init()
slapi_attr_next_value()
slapi_attr_set_valueset()
slapi_attr_syntax_normalize()
slapi_attr_types_equivalent()
```

New Backend Handling Functions (4 to 5.2)

The following functions have been added to handle Slapi_Backend structures:

```
slapi_be_exist()
slapi_be_get_name()
slapi_be_get_readonly()
slapi_be_getsuffix()
slapi_be_gettype()
slapi_be_is_flag_set()
slapi_be_issuffix()
slapi_be_logchanges()
slapi_be_private()
slapi_be_select()
slapi_be_select_by_instance_name()
slapi_free_suffix_list()
slapi_get_first_backend()
slapi_get_next_backend()
slapi_get_suffix_list()
slapi_is_root_suffix()
```

New Control Handling Functions (4 to 5.2)

```
slapi_build_control()
slapi_build_control_from_berval()
slapi_dup_control()
```

New Opaque Data Structures (4 to 5.2)

The following data structures are now defined in `slapi-plugin.h` to wrap key objects used by Directory Server plug-ins:

```
Slapi_Attr
Slapi_Backend
Slapi_ComponentId
Slapi_Connection
Slapi_DN
Slapi_Entry
Slapi_Filter
Slapi_MatchingRuleEntry
Slapi_Mod
Slapi_Mods
Slapi_Operation
Slapi_PBlock
Slapi_RDN
```

Slapi_Value
Slapi_ValueSet

New Distinguished Name Handling Functions (4 to 5.2)

The following functions have been added to handle Slapi_DN structures:

```
slapi_dn_normalize_to_end()
slapi_dn_plus_rdn()
slapi_moddn_get_newdn()
slapi_sdn_compare()
slapi_sdn_copy()
slapi_sdn_done()
slapi_sdn_dup()
slapi_sdn_free()
slapi_sdn_get_backend_parent()
slapi_sdn_get_dn()
slapi_sdn_get_ndn()
slapi_sdn_get_ndn_len()
slapi_sdn_get_parent()
slapi_sdn_get_rdn()
slapi_sdn_is_rdn_component()
slapi_sdn_isempty()
slapi_sdn_isgrandparent()
slapi_sdn_isparent()
slapi_sdn_issuffix()
slapi_sdn_new()
slapi_sdn_new_dn_byref()
slapi_sdn_new_dn_byval()
slapi_sdn_new_dn_passin()
slapi_sdn_new_ndn_byref()
slapi_sdn_new_ndn_byval()
slapi_sdn_scope_test()
slapi_sdn_set_dn_byref()
slapi_sdn_set_dn_byval()
slapi_sdn_set_dn_passin()
slapi_sdn_set_ndn_byref()
slapi_sdn_set_ndn_byval()
slapi_sdn_set_parent()
slapi_sdn_set_rdn()
```

New Entry Handling Functions and Flags (4 to 5.2)

The `slapi_str2entry()` function contains the following new flags defined for handling entries:

```
SLAPI_STR2ENTRY_EXPAND_OBJECTCLASSES
SLAPI_STR2ENTRY_IGNORE_STATE
SLAPI_STR2ENTRY_INCLUDE_VERSION_STR
SLAPI_STR2ENTRY_NOT_WELL_FORMED_LDIF
SLAPI_STR2ENTRY_TOMBSTONE_CHECK
```

The old flags are still available. Refer to comments in `slapi-plugin.h` for hints on how to use the flags.

The following functions have been added to handle entries:

```
slapi_entry2str_with_options()
slapi_entry_add_string()
slapi_entry_add_value()
slapi_entry_attr_add()
slapi_entry_attr_get_long()
slapi_entry_attr_get_uint()
slapi_entry_attr_get_ulong()
slapi_entry_attr_remove()
slapi_entry_attr_set_int()
slapi_entry_attr_set_uint()
slapi_entry_attr_set_ulong()
slapi_entry_delete_string()
slapi_entry_get_dn_const()
slapi_entry_get_ndn()
slapi_entry_get_sdn()
slapi_entry_get_sdn_const()
slapi_entry_get_uniqueid()
slapi_entry_has_children()
slapi_entry_init()
slapi_entry_set_sdn()
slapi_entry_size()
slapi_is_rootdse()
```

New Filter Handling Functions (4 to 5.2)

The following functions have been added to handle `Slapi_Filter` structures:

```
slapi_filter_apply()
slapi_filter_compare()
slapi_filter_get_attribute_type()
slapi_filter_has_extension()
slapi_filter_test_simple()
slapi_find_matching_paren()
slapi_vattr_filter_test()
```


New Internal Operation Functions (4 to 5.2)

The following new functions are intended for use with the updated interface:

```
slapi_add_entry_internal_set_pb()
slapi_add_internal_set_pb()
slapi_delete_internal_set_pb()
slapi_modify_internal_set_pb()
slapi_rename_internal_set_pb()
slapi_search_internal_get_entry()
slapi_search_internal_set_pb()
```

Refer to [Chapter 1](#) to see how to use the updated interface.

New Memory Management Functions (4 to 5.2)

The following table shows functions that have been added to manage memory.

TABLE 3-7 New Functions for Managing Memory

Data Structure	Memory Management Functions
char *	slapi_ch_array_free() slapi_ch_free_string()
Slapi_Attr	slapi_attr_dup() slapi_attr_free() slapi_attr_new()
Slapi_DN	slapi_sdn_dup() slapi_sdn_free() slapi_sdn_new() slapi_sdn_new_dn_byref() slapi_sdn_new_dn_byval() slapi_sdn_new_dn_passin() slapi_sdn_new_ndn_byref() slapi_sdn_new_ndn_byval()
Slapi_Filter	slapi_filter_free()
Slapi_MatchingRuleEntry	slapi_matchingrule_free() slapi_matchingrule_new()
Slapi_Mod	slapi_mod_free() slapi_mod_new()

TABLE 3-7 New Functions for Managing Memory (Continued)

Data Structure	Memory Management Functions
Slapi_Mods	slapi_mods_free()
	slapi_mods_new()
Slapi_PBlock	slapi_pblock_destroy()
	slapi_pblock_new()
Slapi_RDN	slapi_rdn_free()
	slapi_rdn_new()
	slapi_rdn_new_dn()
	slapi_rdn_new_rdn()
	slapi_rdn_new_sdn()
Slapi_Value	slapi_valuearray_free()
	slapi_value_dup()
	slapi_value_free()
	slapi_value_new()
	slapi_value_new_berval()
	slapi_value_new_string()
	slapi_value_new_string_passin()
	slapi_value_new_value()
Slapi_ValueSet	slapi_valueset_free()
	slapi_valueset_new()

New Modification Structure Functions (4 to 5.2)

The following functions have been added to handle Slapi_Mod structures:

```
slapi_mod_add_value()
slapi_mod_done()
slapi_mod_dump()
slapi_mod_free()
slapi_mod_get_first_value()
slapi_mod_get_next_value()
slapi_mod_get_num_values()
slapi_mod_get_operation()
slapi_mod_get_type()
slapi_mod_init()
```

```

slapi_mod_init_byref()
slapi_mod_init_byval()
slapi_mod_init_passin()
slapi_mod_isvalid()
slapi_mod_new()
slapi_mod_remove_value()
slapi_mod_set_operation()
slapi_mod_set_type()

```

The following functions have been added to handle Slapi_Mods structures:

```

slapi_entry2mods()
slapi_mods2entry()
slapi_mods_add()
slapi_mods_add_ldapmod()
slapi_mods_add_mod_values()
slapi_mods_add_modbvps()
slapi_mods_add_smod()
slapi_mods_add_string()
slapi_mods_done()
slapi_mods_dump()
slapi_mods_free()
slapi_mods_get_first_smod()
slapi_mods_get_next_mod()
slapi_mods_get_next_smod()
slapi_mods_get_num_mods()
slapi_mods_init()
slapi_mods_init_byref()
slapi_mods_init_passin()
slapi_mods_insert_after()
slapi_mods_insert_at()
slapi_mods_insert_before()
slapi_mods_insert_smod_at()
slapi_mods_insert_smod_before()
slapi_mods_iterator_backone()
slapi_mods_new()
slapi_mods_remove()

```

New Object Extensions (4 to 5.2)

Object extensions offer a new way of passing data through Directory Server from plug-in to plug-in. The following macros define extensible objects:

```

SLAPI_EXT_CONNECTION
SLAPI_EXT_ENTRY
SLAPI_EXT_MTNODE
SLAPI_EXT_OPERATION

```

The following new functions and associated constructor and destructor callbacks are part of the object extension interface:

```
slapi_extension_constructor_fnptr  
slapi_extension_destructor_fnptr  
slapi_get_object_extension()  
slapi_register_object_extension()  
slapi_set_object_extension()
```

New Operation Handling Functions (4 to 5.2)

```
slapi_op_get_type()  
  
slapi_op_is_flag_set()
```

New Parameter Block Arguments (4 to 5.2)

```
SLAPI_CLIENT_DNS  
SLAPI_CONFIG_DIRECTORY  
SLAPI_CONN_CERT  
SLAPI_CONN_CLIENTNETADDR  
SLAPI_CONN_IS_REPLICATION_SESSION  
SLAPI_CONN_IS_SSL_SESSION  
SLAPI_CONN_SERVERNETADDR  
SLAPI_CONTROLS_ARG  
SLAPI_DESTROY_CONTENT  
SLAPI_IS_INTERNAL_OPERATION  
SLAPI_IS_REPLICATED_OPERATION  
SLAPI_ORIGINAL_TARGET  
SLAPI_ORIGINAL_TARGET_DN  
SLAPI_PLUGIN_ENTRY_FETCH_FUNC  
SLAPI_PLUGIN_ENTRY_STORE_FUNC  
SLAPI_PLUGIN_IDENTITY  
SLAPI_PLUGIN_INTERNAL_POST_ADD_FN  
SLAPI_PLUGIN_INTERNAL_POST_DELETE_FN  
SLAPI_PLUGIN_INTERNAL_POST_MODIFY_FN  
SLAPI_PLUGIN_INTERNAL_POST_MODRDN_FN  
SLAPI_PLUGIN_INTERNAL_PRE_ADD_FN  
SLAPI_PLUGIN_INTERNAL_PRE_DELETE_FN  
SLAPI_PLUGIN_INTERNAL_PRE_MODIFY_FN  
SLAPI_PLUGIN_INTERNAL_PRE_MODRDN_FN  
SLAPI_PLUGIN_POSTSTART_FN  
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_CMP_FN  
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DB_PWD  
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DEC_FN
```

```

SLAPI_PLUGIN_PWD_STORAGE_SCHEME_ENC_FN
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_NAME
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_USER_PWD
SLAPI_RESULT_CODE
SLAPI_RESULT_MATCHED
SLAPI_RESULT_TEXT

```

New Relative Distinguished Name Handling Functions (4 to 5.2)

The following functions have been added to handle Slapi_RDN structures:

```

slapi_rdn_add()
slapi_rdn_compare()
slapi_rdn_contains()
slapi_rdn_contains_attr()
slapi_rdn_done()
slapi_rdn_free()
slapi_rdn_get_first()
slapi_rdn_get_index()
slapi_rdn_get_index_attr()
slapi_rdn_get_next()
slapi_rdn_get_nrdn()
slapi_rdn_get_num_components()
slapi_rdn_get_rdn()
slapi_rdn_init()
slapi_rdn_init_dn()
slapi_rdn_init_rdn()
slapi_rdn_init_sdn()
slapi_rdn_isempty()
slapi_rdn_new()
slapi_rdn_new_dn()
slapi_rdn_new_rdn()
slapi_rdn_new_sdn()
slapi_rdn_remove()
slapi_rdn_remove_attr()
slapi_rdn_remove_index()
slapi_rdn_set_dn()
slapi_rdn_set_rdn()
slapi_rdn_set_sdn()
slapi_sdn_add_rdn()

```

New UTF8 Encoding Functions (4 to 5.2)

The following functions have been added to handle UTF8 encoding and decoding:

```
slapi_has8thBit()  
slapi_UTF8CASECMP()  
slapi_UTF8ISLOWER()  
slapi_UTF8ISUPPER()  
slapi_UTF8NCASECMP()  
slapi_UTF8STRTOLOWER()  
slapi_UTF8STRTOUPPER()  
slapi_UTF8TOLOWER()  
slapi_UTF8TOUPPER()
```

New Value Handling Functions (4 to 5.2)

The following functions have been added to handle `Slapi_Value` structures:

```
slapi_value_compare()  
slapi_value_dup()  
slapi_value_free()  
slapi_value_get_berval()  
slapi_value_get_int()  
slapi_value_get_length()  
slapi_value_get_long()  
slapi_value_get_string()  
slapi_value_get_uint()  
slapi_value_get_ulong()  
slapi_value_init()  
slapi_value_init_berval()  
slapi_value_init_string()  
slapi_value_init_string_passin()  
slapi_value_new()  
slapi_value_new_berval()  
slapi_value_new_string()  
slapi_value_new_string_passin()  
slapi_value_new_value()  
slapi_value_set()  
slapi_value_set_berval()  
slapi_value_set_int()  
slapi_value_set_string()  
slapi_value_set_string_passin()  
slapi_value_set_value()  
slapi_valuearray_free()
```

New Value Set Handling Functions (4 to 5.2)

The following functions have been added to handle `Slapi_ValueSet` structures:

```
slapi_valueset_add_value()  
slapi_valueset_count()  
slapi_valueset_done()  
slapi_valueset_find()  
slapi_valueset_first_value()  
slapi_valueset_free()  
slapi_valueset_init()  
slapi_valueset_new()  
slapi_valueset_next_value()  
slapi_valueset_set_from_smod()  
slapi_valueset_set_valueset()
```

New Virtual Attribute Functions (4 to 5.2)

```
slapi_entry_vattr_find()  
slapi_vattr_attr_free()  
slapi_vattr_attrs_free()  
slapi_vattr_is_registered()  
slapi_vattr_list_attrs()  
slapi_vattr_values_free()  
slapi_vattr_values_get_ex()
```


Getting Started With Directory Server Plug-Ins

This chapter provides an introduction to creating Directory Server plug-ins including a minimal but complete plug-in example. This chapter also explains how to enable the server to use plug-ins and how to generate log entries.

Tip – You can find complete code examples, including all code covered in this chapter, *install-path/examples*.

If you maintain plug-ins developed for a previous release of Directory Server, refer to [Chapter 2](#) and [Chapter 3](#) for information about what has changed.

This chapter covers the following topics:

- “A Hello World Plug-In” on page 81
- “Writing Directory Server Plug-Ins” on page 85
- “Building Directory Server Plug-Ins” on page 88
- “Plugging Libraries Into Directory Server” on page 90
- “Logging Plug-In Messages” on page 97

A Hello World Plug-In

This section demonstrates a plug-in that logs a famous greeting. You can follow the process outlined in this section to understand the concepts that are covered in this chapter.

Find the Code

On the directory host, look in the *install-path/examples/* directory. The example code that is covered in this section is `hello.c`.

Review the Plug-In

The following example logs Hello, World! at Directory Server startup.

EXAMPLE 4-1 Hello, World! Plug-In (hello.c)

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

Slapi_PluginDesc desc = {
    "Hello, World",                /* plug-in identifier */
    "Sun Microsystems, Inc.",      /* vendor name */
    "6.0",                        /* plug-in revision number */
    "My first plug-in"            /* plug-in description */
};

/* Log a greeting at server startup if info logging is on for plug-ins */
int
hello()
{
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN, /* Log if info logging is */
        SLAPI_LOG_INFO_LEVEL_DEFAULT, /* set for plug-ins. */
        SLAPI_LOG_NO_MSGID, /* No client at startup */
        SLAPI_LOG_NO_CONNID, /* No conn. at startup */
        SLAPI_LOG_NO_OPID, /* No op. at startup */
        "hello() in My first plug-in", /* Origin of this message */
        "Hello, World!\n" /* Informational message */
    );
    return 0;
}

/* Register the plug-in with the server */
#ifdef _WIN32
__declspec(dllexport)
#endif
int
hello_init(Slapi_PBlock * pb)
{
    int rc = 0; /* 0 means success */
    rc |= slapi_pblock_set( /* Plug-in API version */
        pb,
        SLAPI_PLUGIN_VERSION,
        SLAPI_PLUGIN_CURRENT_VERSION
    );
    rc |= slapi_pblock_set( /* Plug-in description */
        pb,
        SLAPI_PLUGIN_DESCRIPTION,
        (void *) &desc;
    );
}
```

EXAMPLE 4-1 Hello, World! Plug-In (hello.c) (Continued)

```

    );
    rc |= slapi_pblock_set(          /* Startup function      */
        pb,
        SLAPI_PLUGIN_START_FN,
        (void *) hello
    );
    return rc;
}

```

To log the greeting, the plug-in includes a function to be called at Directory Server startup. The plug-in also includes an initialization function to register the plug-in description, supported API version, and startup function with Directory Server.

The startup function specifies that the message is from a plug-in, `SLAPI_LOG_INFO_AREA_PLUGIN`. The startup function also specifies that the message should be logged when informational logging is activated, `SLAPI_LOG_INFO_LEVEL_DEFAULT`. For this log message, no client connection information is available, `SLAPI_LOG_NO_MSGID`, `SLAPI_LOG_NO_CONNID`, `SLAPI_LOG_NO_OPID`. Directory Server calls the function at startup before any clients have connected. The function specifies where the log message originates, "hello() in My first plug-in". Finally, the function provides the famous log message.

The initialization function is named `hello_init()`. This function modifies the parameter block, `pb`, with the function `slapi_pblock_set()`. The function thus registers the plug-in API version supported, the plug-in description, and the functions offered to Directory Server by this plug-in. As required for all plug-in initialization functions, `hello_init()` returns 0 on success, -1 on failure. The function `slapi_pblock_set()` returns 0 if successful, -1 otherwise. Therefore, you do not need to set the return code to -1 explicitly in [Example 4-1](#).

Build the Plug-In

Build the plug-in as a shared object, `libtest-plugin.so` or `libtest-plugin.sl`, or dynamic link library, `testplugin.dll`, depending on your platform. On Solaris™ SPARC systems, you do the following.

```
$ gmake -f Makefile
```

On Solaris x64 systems where you boot a 64-bit kernel, you do the following.

```
$ gmake -f Makefile64
```

Use the `Makefile` or `Makefile64` in `install-path/examples/` to compile and to link the code. This file builds and links all plug-in examples into the same library. Refer to “[Searching Plug-In Libraries](#)” on page 96 for details when building a plug-in for use with a 64-bit Directory Server instance.

Plug In the Plug-In

Use the `dsconf(1M)` command to add information about the plug-in to the server configuration.

Updating Directory Server Configuration

Configure the server to log plug-in messages:

```
$ dsconf set-log-prop -h localhost -p 1389 error level:err-plugins
```

Configure the server to load the plug-in:

```
$ dsconf create-plugin -h localhost -p 1389 \  
-H lib-path -F hello_init -Y object "Hello World"  
$ dsconf set-plugin-prop "Hello World" feature:"Hello, World" version:6.0  
  vendor:"Sun Microsystems, Inc." desc:"My first plug-in"  
$ dsconf enable-plugin -h localhost -p 1389 "Hello World"  
Directory Server needs to be restarted for changes to take effect
```

Here, *lib-path* must correspond to the absolute path to the plug-in library, such as `/local/myplugins/examples/libtest-plugin.so`. Directory Server requires an absolute path, not a relative path. Before setting *lib-path* on a 64-bit system, read [“Searching Plug-In Libraries” on page 96](#).

Note – Plug-ins delivered with Directory Server have signatures, which are stored as `ds-pluginSignature` attribute values. Plug-ins also have digests, which are stored as `ds-pluginDigest` attribute values. The values allow you to differentiate plug-ins that are delivered with Directory Server from custom plug-ins.

Restarting Directory Server

After changing the Directory Server configuration, you must restart Directory Server for the server to register the plug-in. Use the `dsadm(1M)` command.

For example, to restart a server instance in `/local/ds/`, type the following:

```
$ dsadm restart /local/ds  
Waiting for server to stop...  
Server stopped  
Server started: pid=4362
```

Checking the Log

After Directory Server has started, search the error log, *instance-path/logs/errors*, for Hello, World! You should find an entry similar to the following line, which is wrapped for the printed page:

```
[02/Jan/2006:16:56:07 +0100] - INFORMATION -
hello() in My first plug-in - conn=-1 op=-1 msgId=-1 - Hello, World!
```

At this point, reset the log level to the default to avoid logging unnecessary messages:

```
$ dsconf set-log-prop -h localhost -p 1389 error level:default
```

Writing Directory Server Plug-Ins

This section focuses on the basics of coding a Directory Server plug-in. This section covers the key tasks. These tasks include specifying the header file, writing an initialization function, setting parameter block values, and registering plug-in functions.

Include the `slapi-plugin.h` Header File

The plug-in API is defined in *install-path/include/slapi-plugin.h*. Observe that the header file includes `ldap.h`, the entry point for the standard and extended LDAP C APIs, and `ldap_msg.h`, the list of error message identifiers used by Directory Server.

In general, interfaces that Directory Server exposes are specified in *install-path/include/*. For details about specific features of the API, refer to [Part II](#).

To use the API, include `slapi-plugin.h` in the declaration section of your plug-in source:

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

As a rule, use appropriate macros in your Makefile or project file to tell the linker to look for header files in *install-path/include/*.

Write Your Plug-In Functions

Directory Server calls plug-in functions in the context of a Directory Server operation, for example when a bind, add, search, modify, or delete is performed. Do not export these functions. The functions become available in the appropriate scope when registered with Directory Server at startup. This guide covers how to write such functions.

A plug-in function prototype looks like the prototype of any other locally used function. Many plug-in functions are passed a parameter block.

```
int prebind_auth(Slapi_PBlock * pb); /* External authentication. */
```

You can also use additional helper functions that are not registered with Directory Server. This guide does not cover additional helper functions, but some of the sample plug-ins delivered with the product include such functions.

Use Appropriate Return Codes

In general, return 0 from your plug-in functions when they complete successfully. For some functions that search for matches, 0 means a successful match, and -1 means that no match is found. For preoperation functions, 0 means that Directory Server should continue processing the operation. When you want the server to stop processing the operation after your preoperation function completes, return a positive value such as 1.

When plug-in functions do not complete successfully, send a result to the client if appropriate, log an error message, and return a non-zero value, such as the LDAP result code from *install-path/include/ldap-standard.h*, for example.

Write an Initialization Function

All plug-ins must include an initialization function. The initialization function registers the plug-in version compatibility, the plug-in description, the plug-in functions, and any other data required by the plug-in. The initialization function takes a pointer to a `Slapi_PBlock` structure as a parameter. Refer to [Example 4-1](#).

The initialization function returns 0 if everything registers successfully. The initialization function returns -1 if registration fails for any configuration information or function. A return code of -1 from a plug-in initialization function prevents the server from starting.

Directory Server can call the initialization function more than once during startup.

Set Configuration Information Through the Parameter Block

Directory Server passes a parameter block pointer to the plug-in initialization function. This parameter block holds configuration information. The parameter block can hold other data from Directory Server. The parameter block is the structure into which you add configuration information and pointers to plug-in functions using calls to `slapi_pblock_set()`.

Tip – To read parameter values, use `slapi_pblock_get()`. To write parameter values, use `slapi_pblock_set()`. Using other functions can cause the server to crash.

Specifying Compatibility

You specify compatibility with the plug-in API version as defined in `slapi-plugin.h`, which is described in [Part II](#). If you are creating a new plug-in, for example, use `SLAPI_PLUGIN_CURRENT_VERSION` or `SLAPI_PLUGIN_VERSION_03`. For example, to specify that a plug-in supports the current version, version 3, of the plug-in API, use the following:

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

Here, `pb` is the parameter block pointer passed to the initialization function. `SLAPI_PLUGIN_VERSION` signifies that you are setting plug-in API version compatibility in the parameter block.

Specifying the Plug-In Description

Specify the plug-in description in a `Slapi_PluginDesc` structure, as specified in `slapi-plugin.h` and described in [Part II](#). Call `slapi_pblock_set()` to register the description:

```
slapi_pblock_set(pb, SLAPI_PLUGIN_DESCRIPTION, (void *) &desc);
```

Here, `desc` is a `Slapi_PluginDesc` structure that contains the plug-in description. See [Example 4-1](#).

Set Pointers to Functions Through the Parameter Block

Register plug-in functions according to what operation Directory Server performs. Also register plug-in functions according to when the operation is performed. Directory Server typically calls plug-in functions before, during, or after a standard operation.

Macros that specify when Directory Server should call plug-ins have the form `SLAPI_PLUGIN_*_FN`. As the sample plug-ins demonstrate, the macro used to register a plug-in function depends entirely on what the function is supposed to do. For example, to register a plug-in function `foo()` to be called at Directory Server startup, do the following:

```
slapi_pblock_set(pb, SLAPI_PLUGIN_START_FN, (void *) foo);
```

Here, `pb` is the parameter block pointer passed to the initialization function. `SLAPI_PLUGIN_START_FN` signifies that Directory Server calls `foo()` at startup. Note that `foo()` returns 0 on success.

Locate Examples

Sample plug-ins are located in *install-path/examples/*.

Building Directory Server Plug-Ins

This section explains how to build plug-in binaries. This section deals with compilation requirements and with linking requirements for Directory Server plug-ins.

Include the Header File for the Plug-In API

The compiler needs to be able to locate the header files such as `slapi-plugin.h`. Find header files in *install-path/include/*.

Link the Plug-In as a Shared Object or Dynamic Link Library

Link plug-ins so the plug-in library is reentrant and portable and can be shared. The following example shows one way of building a plug-in library. The example works with the Sun compiler that is used with a 64-bit Directory Server running on Solaris platforms.

EXAMPLE 4-2 64-bit: Makefile for a Solaris Sample Plug-In Library

```
INCLUDE_FLAGS = -I../include
CFLAGS = $(INCLUDE_FLAGS) -D_REENTRANT -KPIC -xarch=v9 -DUSE_64
LDFLAGS = -G
DIR64 = 64

OBJS = dns.o entries.o hello.o internal.o testpwdstore.o \
      testsaslbind.o testextendedop.o testpreop.o testpostop.o \
      testentry.o testbind.o testgetip.o

all: MKDIR64 $(DIR64)/libtest-plugin.so

MKDIR64:
    @if [ ! -d $(DIR64) ]; then mkdir $(DIR64); fi

$(DIR64)/libtest-plugin.so: $(OBJS)
    $(LD) $(LDFLAGS) -o $@ $(OBJS)

.c.o:
    $(CC) $(CFLAGS) -c $<

clean:
    -rm -f $(OBJS) libtest-plugin.so $(DIR64)/libtest-plugin.so
```


The CFLAGS `-xarch=v9` and `-DUSE_64` specify that the compiler builds the library for use with a 64-bit server. Notice also that the 64-bit library is placed in a directory that is named `64/`. This extra directory is required for 64-bit plug-ins. A 64-bit server adds the name of the extra directory to the path name when searching for the library. Refer to “[Searching Plug-In Libraries](#)” on page 96 for further information.

The following example shows a 32-bit equivalent of the Makefile.

EXAMPLE 4-3 32-bit: Makefile for a Solaris Sample Plug-In Library

```
INCLUDE_FLAGS = -I../include
CFLAGS = $(INCLUDE_FLAGS) -D_REENTRANT -KPIC
LDFLAGS = -G

OBJS = dns.o entries.o hello.o internal.o testpwdstore.o \
      testsaslbind.o testextendedop.o testpreop.o testpostop.o \
      testentry.o testbind.o testgetip.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
```

Notice that both 32-bit versions and 64-bit versions of the plug-in library can coexist on the same host.

Locate the Example Build Rules

Build rules are in *install-path/examples/*. The rules are also in *install-path/examples/Makefile64*. Refer to these files for the recommended setup for compiling and linking on your platform.

Plugging Libraries Into Directory Server

This section covers server plug-in configuration. This section also explains how plug-ins are loaded. When plug-ins are properly loaded, Directory Server can initialize and register the plug-ins correctly. Dependencies between Directory Server plug-ins are also discussed.

Specify Plug-In Configuration Settings

Plug-in configuration settings specify information that Directory Server requires. The server uses the information to load a plug-in at startup, to identify the plug-in, and to pass parameters to the plug-in at load time.

You access configuration settings through the `dsconf(1M)` command, using the subcommands `create-plugin`, `set-plugin-prop`, and `enable-plugin`.

The `dsconf create-plugin` command forces you to set required plug-in configuration settings. The settings include the name, plug-in initialization function, path to the library that contains the plug-in, and plug-in type.

The `dsconf set-plugin-prop` command allows you to set optional configuration settings.

The `dsconf enable-plugin` command allows you to turn the plug-in on or off. When you change this setting, you must restart the server.

The following table identifies the configuration settings and indicates whether the settings are required, set with `dsconf create-plugin`, or optional, set with `dsconf set-plugin-prop`.

TABLE 4-1 Plug-In Configuration Settings

Setting	Description	Required or Optional
Plug-in name	Common name for the plug-in, corresponding to the name registered by the plug-in	Required
Initialization function (-F)	Name of the initialization function that is called during Directory Server startup as the name appears in the plug-in code	Required
Library path (-H)	Full path to the library that contains the plug-in, not including 64-bit directory for 64-bit plug-in libraries Refer to “Searching Plug-In Libraries” on page 96 for details.	Required

TABLE 4-1 Plug-In Configuration Settings (Continued)

Setting	Description	Required or Optional
Plug-in type (-Y)	Plug-in type that defines the types of functions the plug-in implements Refer to “Understanding Plug-In Types and Dependencies” on page 91 for details.	Required
argument	A parameter passed to the plug-in at Directory Server startup such as <code>suffix=dc=example,dc=com</code> Refer to “Retrieving Arguments Passed to Plug-Ins” on page 94 .	Optional
depends-on-named	One or more plug-in dependencies on plug-in names (RDN) such as State Change Plugin or Multimaster Replication Plugin Refer to “Understanding Plug-In Types and Dependencies” on page 91 for details.	Optional
depends-on-type	One or more plug-in dependencies on plug-in types such as <code>preoperation</code> or <code>database</code> Refer to “Understanding Plug-In Types and Dependencies” on page 91 for details.	Optional
desc	One-line description as specified in the <code>spd_description</code> field of the plug-in <code>Slapi_PluginDesc</code> structure	Optional
feature	Identifier as specified in the <code>spd_id</code> field of the plug-in <code>Slapi_PluginDesc</code> structure	Optional
vendor	Vendor name as specified in the <code>spd_vendor</code> field of the plug-in <code>Slapi_PluginDesc</code> structure	Optional
version	Version number as specified in the <code>spd_version</code> field of the plug-in <code>Slapi_PluginDesc</code> structure	Optional

Understanding Plug-In Types and Dependencies

The plug-in *type* tells Directory Server which type of plug-in functions can be registered by a plug-in. A plug-in type corresponds to the principal type of operation that the plug-in performs. Refer to [“How Plug-Ins Interact With the Server” on page 48](#) and [“Example Uses” on page 53](#) for explanations of plug-in types. [Part II](#) describes plug-in types and lists type property settings that correspond to the type identifiers specified in `slapi-plugin.h`.

Determining the plug-in type is clear-cut when you know what the plug-in must do. For example, if you write a plug-in to handle authentication of a request before Directory Server processes the bind for that request, the type is `preoperation`. In other words, the plug-in does

something to the request *prior to* the bind operation. Such a plug-in therefore registers at least a pre-bind plug-in function to process the request prior to the bind. If, on the other hand, you write a plug-in to encode passwords, and also compare incoming passwords with encoded passwords, the type is `pwdstoragescheme`. Such a plug-in registers at least password encoding, and also compares functions. In both cases, plug-in type follows plug-in function.

Plug-in *dependencies* tell Directory Server that a plug-in requires one or more other plug-ins in order to function properly. Directory Server resolves dependencies as the server loads the plug-ins. Therefore, Directory Server fails to register a plug-in if a plug-in that the plug-in depends on cannot be registered.

Specify plug-in dependencies by name, using `depends-on-named`, or by type, using `depends-on-type`. Here, plug-in name refers to the plug-in relative distinguished name (RDN) from the configuration entry. Plug-in type is a type identifier from the list of types in [Chapter 18](#). Both `depends-on-named` and `nsslapdepends-on-type` can take multiple values.

A plug-in configuration entry can specify either kind or both kinds of dependencies. Use the configuration entry to identify dependencies on specific plug-ins by *name*. Identify dependencies on a type of plug-in by *type*. If a dependency identifies a plug-in by name, Directory Server only registers the plug-in after the server registers the named plug-in. If a dependency identifies a plug-in by type, Directory Server only registers the plug-in after the server registers *all* plug-ins of the specified type.

Do not confuse plug-in registration dependencies with a mechanism to specify the order in which Directory Server calls plug-ins.

Ordering Plug-In Calls

Directory Server allows you to define the order in which plug-ins are called. This mechanism is independent of the mechanism that defines plug-in load dependencies.

This mechanism uses a set of attributes on the entry with DN `cn=plugins,cn=monitor`. Plug-in call ordering is defined for each type of plug-in on attributes whose values are comma-separated lists of plug-in names. Each attribute type name starts with `plugin-list-` and identifies a point in the server request processing where plug-ins can be called. In the comma-separated list of plug-ins, the first plug-in in the list is called first, the second is called second, and so forth.

For example, six postoperation plug-ins are called after a modify operation. The `plugin-list-postoperation-modify` attribute on `cn=plugins,cn=monitor` shows the order in which these six plug-ins are called by the server.

```
plugin-list-postoperation-modify:  
  Class of Service,  
  Legacy replication postoperation plugin,  
  Multimaster replication postoperation plugin,  
  Retrocl postoperation plugin,  
  Roles Plugin,  
  State Change Plugin
```

You can also retrieve these attributes by performing a search on the entry with DN `cn=plugins,cn=monitor`.

To change the order in which plug-ins are called, you must not modify the value of the attribute type starting with `plugin-list-`. Instead, you must modify the value of a corresponding attribute type starting with `plugin-order-`, and having the same ending as the `plugin-list-` attribute.

The full list of plug-in call ordering attribute types is as follows:

- `plugin-order-bepostoperation-add`
- `plugin-order-bepostoperation-delete`
- `plugin-order-bepostoperation-modify`
- `plugin-order-bepostoperation-modrdn`
- `plugin-order-beprecommit-add`
- `plugin-order-beprecommit-delete`
- `plugin-order-beprecommit-modify`
- `plugin-order-beprecommit-modrdn`
- `plugin-order-bepreoperation-add`
- `plugin-order-bepreoperation-delete`
- `plugin-order-bepreoperation-modify`
- `plugin-order-bepreoperation-modrdn`
- `plugin-order-internalpostoperation-add`
- `plugin-order-internalpostoperation-delete`
- `plugin-order-internalpostoperation-modify`
- `plugin-order-internalpostoperation-modrdn`
- `plugin-order-internalpreoperation-add`
- `plugin-order-internalpreoperation-delete`
- `plugin-order-internalpreoperation-modify`
- `plugin-order-internalpreoperation-modrdn`
- `plugin-order-postoperation-abandon`
- `plugin-order-postoperation-add`
- `plugin-order-postoperation-bind`
- `plugin-order-postoperation-compare`
- `plugin-order-postoperation-delete`
- `plugin-order-postoperation-entry`
- `plugin-order-postoperation-modify`
- `plugin-order-postoperation-modrdn`
- `plugin-order-postoperation-referral`
- `plugin-order-postoperation-result`
- `plugin-order-postoperation-search`
- `plugin-order-postoperation-unbind`
- `plugin-order-preoperation-abandon`
- `plugin-order-preoperation-add`
- `plugin-order-preoperation-attr-encode-result`

- `plugin-order-preoperation-bind`
- `plugin-order-preoperation-compare`
- `plugin-order-preoperation-delete`
- `plugin-order-preoperation-entry`
- `plugin-order-preoperation-finish-entry-encode-result`
- `plugin-order-preoperation-modify`
- `plugin-order-preoperation-modrdn`
- `plugin-order-preoperation-referral`
- `plugin-order-preoperation-result`
- `plugin-order-preoperation-search`
- `plugin-order-preoperation-unbind`
- `plugin-order-pwdpolicy-bind-fail`
- `plugin-order-pwdpolicy-bind-success`

For example, if you want to change the order in which postoperation plug-ins are called after a modify, first read `plugin-list-postoperation-modify`. Next set `plugin-order-postoperation-modify`, copying plug-in names from the value of `plugin-list-postoperation-modify`. Then restart Directory Server. Plug-in names are not case—sensitive, but white spaces are taken into account. Unrecognized names are ignored.

With a single asterisk, *, as an item in the comma-separated list, you let Directory Server define the call order for plug-ins you do not identify explicitly.

If you have two postoperation modify plug-ins, the first, `Call Me First`, and the second, `Call Me Last`, you could set `plugin-list-postoperation-modify` as follows:

```
plugin-order-postoperation-modify: Call Me First,*,Call Me Last
```

The plug-in name that you specify must be identical to the name argument that is passed to the `slapi_register_plugin` function. There must also be no white space between the comma delimiter and the adjacent plug-in names, and no white space at the end of the names list.

Plug-ins can remain that you do not explicitly identify by name or implicitly identify using *. Directory Server calls such plug-ins *before* calling the plug-ins that you listed.

Retrieving Arguments Passed to Plug-Ins

Your plug-in can retrieve parameters specified in the configuration settings property argument, which is multivalued. Directory Server makes the argument available to the plug-in through the parameter block passed to the plug-in initialization function.

If you include parameters in the configuration settings, the following apply:

- `SLAPI_PLUGIN_ARGC` specifies the number of parameters
- `SLAPI_PLUGIN_ARGV` contains the parameters

The following example uses `slapi_pblock_get()` to retrieve the arguments from `install-path/examples/testextendedop.c`.

EXAMPLE 4-4 Plug-In Using Arguments (testextendedop.c)

```

#include "slapi-plugin.h"

Slapi_PluginDesc expdesc = {
    "test-extendedop",          /* plug-in identifier */
    "Sun Microsystems, Inc.",    /* vendor name */
    "6.0",                      /* plug-in revision number */
    "Sample extended operation plug-in" /* plug-in description */
};

/* Register the plug-in with the server. */
#ifdef _WIN32
__declspec(dllexport)
#endif
int
testexop_init(Slapi_PBlock * pb)
{
    char ** argv;                /* Args from configuration */
    int     argc;                /* entry for plug-in. */
    char ** oid_list;            /* OIDs supported */
    int     rc = 0;              /* 0 means success */
    int     i;

    /* Get the arguments from the configuration entry. */
    rc |= slapi_pblock_get(pb, SLAPI_PLUGIN_ARGV, &argv);
    rc |= slapi_pblock_get(pb, SLAPI_PLUGIN_ARGC, &argc);
    if (rc != 0) {
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            SLAPI_LOG_NO_MSGID,
            SLAPI_LOG_NO_CONNID,
            SLAPI_LOG_NO_OPID,
            "testexop_init in test-extendedop plug-in",
            "Could not get plug-in arguments.\n"
        );
        return (rc);
    }

    /* Extended operation plug-ins may handle a range of OIDs. */
    oid_list = (char **)slapi_ch_malloc((argc + 1) * sizeof(char *));
    for (i = 0; i < argc; ++i) {
        oid_list[i] = slapi_ch_strdup(argv[i]);
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            SLAPI_LOG_NO_MSGID,

```

EXAMPLE 4-4 Plug-In Using Arguments (testextendedop.c) *(Continued)*

```
        SLAPI_LOG_NO_CONNID,  
        SLAPI_LOG_NO_OPID,  
        "testexop_init in test-extendedop plug-in",  
        "Registering plug-in for extended operation %s.\n",  
        oid_list[i]  
    );  
}  
oid_list[argc] = NULL;  
  
rc |= slapi_pblock_set(          /* Plug-in API version      */  
    pb,  
    SLAPI_PLUGIN_VERSION,  
    SLAPI_PLUGIN_CURRENT_VERSION  
);  
rc |= slapi_pblock_set(          /* Plug-in description  */  
    pb,  
    SLAPI_PLUGIN_DESCRIPTION,  
    (void *) &expdesc  
);  
rc |= slapi_pblock_set(          /* Extended op. handler */  
    pb,  
    SLAPI_PLUGIN_EXT_OP_FN,  
    (void *) test_extendedop  
);  
rc |= slapi_pblock_set(          /* List of OIDs handled */  
    pb,  
    SLAPI_PLUGIN_EXT_OP_OIDLIST,  
    oid_list  
);  
return (rc);  
}
```

You can specify multiple arguments by using multiple properties in the command line. The `dsconf` command causes the server to provide the arguments to the plug-in in the order you specify on the command line. When you change the list of arguments by using the `dsconf` command, the new arguments replace the existing arguments. The arguments are not appended to the list.

Searching Plug-In Libraries

Directory Server searches for plug-in libraries with the library path that you specify with the `-H` option to the `dsconf create-plugin` command.

- 64-bit servers search for 64-bit libraries by adding a directory name to the end of the path name.

For example, if you specify `-H /local/myplugins/mylib.so`, a 64-bit Directory Server running on Solaris platforms searches for a 64-bit plug-in library that is named as follows:

```
/local/myplugins/64/mylib.so
```

- A 32-bit Directory Server searches for 32-bit plug-in libraries with the exact path specified. A 32-bit Directory Server using the same configuration entry would search for the plug-in library named:

```
/local/myplugins/mylib.so
```

Directory Server fails to start when the server cannot find a plug-in library specified in the configuration entry.

Modify the Directory Server Configuration

With Directory Server running, add your plug-in configuration settings with the `dsconf(1M)` command. Enable the plug-in with the `dsconf enable-plugin name` command.

If everything works as expected, Directory Server adds a plug-in configuration entry to the Directory Server configuration, which is stored in *instance-path*/config/dse.ldif. Do not modify the configuration file directly.

Restart Directory Server

Directory Server does not load plug-ins dynamically. Instead, you must restart the server. Directory Server then registers your plug-in at startup.

Set logging so Directory Server logs plug-in informational messages with the `dsconf set-log-prop error level:err-plugins` command. After setting the log level appropriately, restart the server to ensure that it loads the new plug-ins that you configured:

```
$ dsadm restart instance-path
```

You can adjust log levels while the server is running.

Logging Plug-In Messages

This section shows how to do the following:

- Generate messages in the Directory Server logs
- Set the log level such that Directory Server writes particular types of messages to the log
- Find the messages that the server has logged

Log Three Levels of Message Severity

The plug-in API provides log functions for logging fatal error messages, warnings, and information. Error messages and warnings are always logged. Informational logging is turned off by default. You can adjust log levels while Directory Server is running.

Tip – Directory Server, as Directory Server waits for the system to write every log message to disk, slowing the server down.

Use logging when you need logging. Turn logging off when you do not need to use it.

Refer to [Chapter 16](#) for details about each of the logging functions.

Fatal Error Messages

For fatal errors, use `slapi_log_error_ex()`. In many cases, you might return `-1` after you log the message to indicate to Directory Server that a serious problem has occurred.

EXAMPLE 4-5 Logging a Fatal Error Message

```
#include "slapi-plugin.h"
#include "example-com-error-ids.h" /* example.com unique
                                   error IDs file      */

int
foobar(Slapi_PBlock * pb)
{
    char * error_cause;
    int    apocalypse = 1;          /* Expect the worst.    */

    /* ... */

    if (apocalypse) {               /* Server to crash soon */
        slapi_log_error_ex(
            EXCOM_SERVER_MORIBUND, /* Unique error ID      */
            SLAPI_LOG_NO_MSGID,
            SLAPI_LOG_NO_CONNID,
            SLAPI_LOG_NO_OPID,
            "example.com: foobar in baz plug-in",
            "cannot write to file system: %s\n",
            error_cause
        );
        return -1;
    }
    return 0;
}
```

In this example, `foobar()` logs an error as Directory Server is about to crash.

Tip – If the plug-ins are internationalized, use macros, not literal strings, for the last two arguments to `slapi_log*_ex()` functions.

Warning Messages

For serious situations that require attention and in which messages should always be logged, use `slapi_log_warning_ex()`.

In the following example, `foobar()` logs a warning because the disk is nearly full.

EXAMPLE 4-6 Logging a Warning Message

```
#include "slapi-plugin.h"
#include "example-com-warning-ids.h" /* example.com unique
                                     warning IDs file */

int
foobar()
{
    int disk_use_percentage;

    /* ... */

    if (disk_use_percentage >= 95) {
        slapi_log_warning_ex(
            EXCOM_DISK_FULL_WARN, /* unique warning ID */
            SLAPI_LOG_NO_MSGID,
            SLAPI_LOG_NO_CONNID,
            SLAPI_LOG_NO_OPID,
            "example.com: foobar in baz plug-in",
            "disk %.0f%% full, find more space\n",
            disk_use_percentage
        );
    }
    return 0;
}
```

Informational Messages

For informational or debug messages, use `slapi_log_info_ex()`. This function can be set for default logging, which means that the message is not logged when logging is turned off for plug-ins. Informational logging can also be set to occur only when heavy logging is used. Thus, the message is logged when plug-in logging is both turned on and set to log all informational messages.

Refer to [“Review the Plug-In” on page 82](#) for an example of how to log an informational message from a plug-in.

Set the Appropriate Log Level in the Directory Server Configuration

Log levels can be set for plug-in informational messages, as well as for a number of other Directory Server subsystems. You can set the log level through Directory Service Control Center. You can also set the log level from the command line by using the `dsconf set-log-prop` command.

When you set `dsconf set-log-prop error level:err-plugins`, the server turns on plug-in informational logging.

Find Messages in the Log

The log file for errors, warnings, and informational messages is *instance-path/logs/errors*. All messages go to the same log so you can easily determine the order in which events occurred.

As shown in this example, messages consist of a timestamp, followed by an identifier, followed by the other information in the log message. Note that this log message is broken for the printed page.

```
[02/Jan/2006:16:54:57 +0100] - INFORMATION -  
start_tls - conn=-1 op=-1 msgId=-1 -  
Start TLS extended operation request confirmed.
```

Use the identifiers to filter what you do not need.

Working With Entries Using Plug-Ins

This chapter covers plug-in API features for handling directory entries, attributes, attribute values, and distinguished names (DNs). This chapter also deals with converting entries to and from LDAP Data Interchange Format (LDIF) strings, and checking whether entries comply with LDAP schema.

Code excerpts used in this chapter can be found in the *install-path/examples/entries.c* sample plug-in and the *install-path/examples/dns.c* sample plug-in.

See [Chapter 16](#) for information about the plug-in API functions used in this chapter.

This chapter covers the following topics:

- “Creating Entries” on page 101
- “Converting To and From LDIF Representations” on page 102
- “Getting Entry Attributes and Attribute Values” on page 104
- “Adding and Removing Attribute Values” on page 106
- “Verifying Schema Compliance for an Entry” on page 108
- “Handling Entry Distinguished Names” on page 109

Creating Entries

This section demonstrates how to create an entry in the directory. Create a new entry by allocating space, and creating a completely new entry. Alternatively, create a new entry by duplicating an existing entry and modifying its values.

Creating New Entries

Create a completely new entry by using `slapi_entry_alloc()` to allocate the memory that is required, as shown in the following example.

EXAMPLE 5-1 Creating a New Entry (`entries.c`)

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

int
test_ldif()
{
    Slapi_Entry * entry = NULL;          /* Entry to hold LDIF      */

    /* Allocate the Slapi_Entry structure.      */
    entry = slapi_entry_alloc();

    /* Add code that fills the Slapi_Entry structure.      */

    /* Add code that uses the Slapi_Entry structure.      */

    /* Release memory allocated for the entry.      */
    slapi_entry_free(entry);

    return (0);
}
```

Creating Copies of Entries

Create a copy of an entry with `slapi_entry_dup()`.

Converting To and From LDIF Representations

This section shows how to use the plug-in API to convert entries that are represented in LDIF to `Slapi_Entry` structures. This section also shows how to convert `Slapi_Entry` structures to LDIF strings.

LDIF files appear as a series of human-readable representations of directory entries, and optionally access control instructions. Entries that are represented in LDIF start with a line for the distinguished name. The LDIF representation continues with optional lines for the attributes. The syntax is shown in the following example.

EXAMPLE 5-2 LDIF Syntax Representing an Entry

```
dn:[:] dn-value\n
[attribute:[:] value\n]
[attribute:[:] value\n]
[single-space continued-value\n]*
```

A double colon, ::, indicates that the *value* is base64-encoded. Base64-encoded values can, for example, have a line break in the midst of an attribute value.

As shown in the preceding example, LDIF lines can be folded by leaving a single space at the beginning of the continued line.

Sample LDIF files can be found in *install-path/ldif/*.

Refer to Chapter 13, “Directory Server LDIF and Search Filters,” in *Sun Java System Directory Server Enterprise Edition 6.1 Reference* for details on LDIF syntax.

Converting an LDIF String to a Slapi_Entry Structure

You can achieve this type of conversion by using `slapi_str2entry()`. The function takes as its two arguments the string to convert and an `int` holding flag of the form `SLAPI_STR2ENTRY_*` in `slapi-plugin.h`. The function returns a pointer to a `Slapi_Entry` if successful, `NULL` otherwise, as shown in the following example.

EXAMPLE 5-3 Converting To and From LDIF Strings (`entries.c`)

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

#define LDIF_STR "dn: dc=example,dc=com\nobjectclass: \
    top\nobjectclass: domain\ndc: example\n"

int
test_ldif()
{
    char          * ldif  = NULL;          /* Example LDIF string */
    Slapi_Entry * entry = NULL;          /* Entry to hold LDIF */
    char          * str   = NULL;          /* String to hold entry */
    int           len;                    /* Length of entry as LDIF */

    /* LDIF to Slapi_Entry */
    entry = slapi_entry_alloc();
    ldif = slapi_ch_strdup(LDIF_STR);
    entry = slapi_str2entry(ldif, SLAPI_STR2ENTRY_ADDDRNVALS);
    slapi_ch_free_string(&ldif);
    if (entry == NULL) return (-1);

    /* Slapi_Entry to LDIF */
    str = slapi_entry2str(entry, &len);
    if (str == NULL) return (-1);
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
```

EXAMPLE 5-3 Converting To and From LDIF Strings (`entries.c`) *(Continued)*

```
        SLAPI_LOG_NO_MSGID,  
        SLAPI_LOG_NO_CONNID,  
        SLAPI_LOG_NO_OPID,  
        "test_ldif in test-entries plug-in",  
        "\nOriginal entry:\n%sEntry length: %d\n", str, len  
    );  
  
    slapi_entry_free(entry);  
  
    return (0);  
}
```

Here, `SLAPI_STR2ENTRY_ADDRDNVALS` adds any missing relative distinguished name (RDN) values, as specified in `slapi-plugin.h` where supported flags for `slapi_str2entry()` are listed.

Converting a `Slapi_Entry` Structure to an LDIF String

You can achieve this type of conversion by using `slapi_entry2str()`. This function takes as its two arguments the entry to convert and an `int` to hold the length of the string that is returned. The function returns a `char *` to the LDIF if successful, `NULL` otherwise, as shown in Example 38-3.

Getting Entry Attributes and Attribute Values

This section demonstrates how to iterate through real attributes of an entry. You can use this technique when looking through the list of attributes that belong to an entry. If you know that the entry contains a particular attribute, `slapi_entry_attr_find()` returns a pointer to the attribute directly.

Note – Real attributes contrast with virtual attributes generated by Directory Server for advanced entry management with roles and Class of Service (CoS).

When working with virtual attributes such as attributes for CoS, you can use `slapi_vattr_list_attrs()`. This function provides the complete list of real attributes as well as virtual attributes that belong to an entry.

Iteration through attribute values can be done using `slapi_attr_first_value_const()` and `slapi_attr_next_value_const()` together as shown in the following example.

EXAMPLE 5-4 Iterating Through Attributes of an Entry (testpreop.c)

```

#include "slapi-plugin.h"

int
testpreop_add(Slapi_PBlock * pb)
{
    Slapi_Entry * entry;           /* Entry to add */
    Slapi_Attr * attribute;        /* Entry attributes */
    Slapi_Value ** values;         /* Modified, duplicate vals*/
    int connId, opId, rc = 0;
    long msgId;

    /* Get the entry. */
    rc |= slapi_pblock_get(pb, SLAPI_ADD_ENTRY, &entry);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID, &msgId);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID, &connId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID, &opId);

    /* Prepend ADD to values of description attribute. */
    rc |= slapi_entry_attr_find(entry, "description", &attribute);
    if (rc == 0) { /* Found a description, so... */
        int nvals, i, count = 0;
        const Slapi_Value * value;

        slapi_attr_get_numvalues(attribute, &nvals);
        values = (Slapi_Value **)
            slapi_ch_malloc((nvals+1)*sizeof(Slapi_Value *));

        for ( /* ...loop for value... */
            i = slapi_attr_first_value_const(attribute, &value);
            i != -1;
            i = slapi_attr_next_value_const(attribute, i, &value)
        ) { /* ...prepend "ADD ". */
            const char * string;
            char * tmp;
            values[count] = slapi_value_dup(value);
            string = slapi_value_get_string(values[count]);
            tmp = slapi_ch_malloc(5+strlen(string));
            strcpy(tmp, "ADD ");
            strcpy(tmp+4, string);
            slapi_value_set_string(values[count], tmp);
            slapi_ch_free((void **)&tmp);
            ++count;
        }

        values[count] = NULL;
    }
}

```

EXAMPLE 5-4 Iterating Through Attributes of an Entry (testpreop.c) (Continued)

```
    } else {
        /* entry has no desc. */
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            msgId,
            connId,
            opId,
            "testpreop_add in test-preop plug-in",
            "Entry has no description attribute.\n"
        );
    }

    rc = slapi_entry_attr_replace_sv(entry, "description", values);
    if (rc != 0) {
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            msgId,
            connId,
            opId,
            "testpreop_add in test-preop plug-in",
            "Description attribute(s) not modified.\n"
        );
    }
    slapi_valuearray_free(&values);

    return 0;
}
```

Adding and Removing Attribute Values

This section demonstrates how to add and remove attributes and their values.

Adding Attribute Values

If the attribute name and value are strings, you might be able to use `slapi_entry_add_string()`. The following example builds an entry first by setting the DN, then by adding attributes with string values.

EXAMPLE 5-5 Adding String Attribute Values (entries.c)

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

int
```

EXAMPLE 5-5 Adding String Attribute Values (entries.c) *(Continued)*

```

test_create_entry()
{
    Slapi_Entry * entry = NULL;          /* Original entry          */

    entry = slapi_entry_alloc();
    slapi_entry_set_dn(entry, slapi_ch_strdup("dc=example,dc=com"));
    slapi_entry_add_string(entry, "objectclass", "top");
    slapi_entry_add_string(entry, "objectclass", "domain");
    slapi_entry_add_string(entry, "dc", "example");

    /* Add code using the entry you created.....          */

    slapi_entry_free(entry);

    return (0);
}

```

When working with values that already exist or with values that are not strings, you can use the following functions

- Use `slapi_entry_add_values_sv()` to add new values to an attribute of an entry. This function returns an error when duplicates of the new values already exist.
- Use `slapi_entry_attr_merge_sv()` to add new values to the existing values of a multivalued attribute. This function does not return an error when duplicates of the new values already exist.

The following example demonstrates `slapi_entry_attr_merge_sv()`.

EXAMPLE 5-6 Merging Attribute Values (entries.c)

```

#include "slapi-plugin.h"

int
test_ldif()
{
    Slapi_Entry * entry = NULL;          /* Entry to hold LDIF          */
    Slapi_Value * values[2];             /* Attribute values            */

    entry = slapi_entry_alloc();

    /* Add code to transform an LDIF representation into an entry.*/

    /* Add a description by setting the value of the attribute.
     * Although this is overkill when manipulating string values,
     * it can be handy when manipulating binary values.          */
}

```

EXAMPLE 5-6 Merging Attribute Values (entries.c) (Continued)

```
values[0] = slapi_value_new_string("Description for the entry.");
values[1] = NULL;
if (slapi_entry_attr_merge_sv(entry, "description", values) != 0)
    /* Merge did not work if processing arrives here. */ ;

slapi_entry_free(entry);

return (0);
}
```

If the attribute exists in the entry, `slapi_entry_attr_merge_sv()` merges the specified values into the set of existing values. If the attribute does not exist, `slapi_entry_attr_merge_sv()` adds the attribute with the specified values.

Removing Attribute Values

Remove attribute values with `slapi_entry_delete_string()` or `slapi_entry_delete_values_sv()`.

Verifying Schema Compliance for an Entry

This section demonstrates how to check that an entry is valid with respect to the directory schema known to Directory Server. Verify schema compliance for an entry with `slapi_entry_check()`. The two arguments of the function are a pointer to a parameter block and a pointer to the entry, as shown in the following example.

EXAMPLE 5-7 Checking Schema Compliance (entries.c)

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

int
test_create_entry()
{
    Slapi_Entry * entry = NULL;          /* Original entry          */
    Slapi_Entry * ecopy = NULL;          /* Copy entry              */

    entry = slapi_entry_alloc();

    /* Add code to fill the entry, setting the DN and attributes. */

    ecopy = slapi_entry_dup(entry);
```

EXAMPLE 5-7 Checking Schema Compliance (entries.c) (Continued)

```

slapi_entry_set_dn(ecopy, slapi_ch_strdup("dc=example,dc=org"));
slapi_entry_add_string(ecopy, "description", "A copy of the orig.");

/* Does the resulting copy comply with the schema? */
if (slapi_entry_schema_check(NULL, ecopy) == 0) {
    /* Resulting entry does comply. */ ;
} else {
    /* Resulting entry does not comply. */ ;
}

slapi_entry_free(entry);
slapi_entry_free(ecopy);

return (0);
}

```

Notice that the parameter block pointer argument is NULL. Leave the parameter block pointer argument NULL in most cases. When the plug-in is used in a replicated environment, you can use the argument to prevent schema compliance verification for replicated operations.

Handling Entry Distinguished Names

This section demonstrates how to work with distinguished names (DNs) to do the following:

- Determine the parent DN of a given DN
- Determine the suffix DN with which the entry DN is associated
- Identify whether a root suffix is served by the back end
- Set the DN of the entry
- Normalize the DN of the entry for comparison with other entries
- Determine whether the entry DN is the DN of the directory superuser

Getting the Parent and Suffix DNs

Use the `slapi_dn_parent()` function to return the parent and `slapi_dn_beparent()` to return the suffix associated with the entry, as shown in the following example.

EXAMPLE 5-8 Determining the Parent and Suffix of an Entry (dns.c)

```

#include "slapi-plugin.h"

int
test_dns(Slapi_PBlock * pb)

```

EXAMPLE 5-8 Determining the Parent and Suffix of an Entry (dns.c) *(Continued)*

```
{
    char * bind_DN;                /* DN being used to bind */
    char * parent_DN;              /* DN of parent entry */
    char * suffix_DN;              /* DN of suffix entry */
    int connId, opId, rc = 0;
    long msgId;

    rc |= slapi_pblock_get(pb, SLAPI_BIND_TARGET, &bind_DN);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID, &msgId);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID, &connId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID, &opId);
    if (rc != 0) return (-1);

    /* Get the parent DN of the DN being used to bind. */
    parent_DN = slapi_dn_parent(bind_DN);
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "test_dns in test-dns plug-in",
        "Parent DN of %s: %s\n", bind_DN, parent_DN
    );

    /* Get the suffix DN of the DN being used to bind. */
    suffix_DN = slapi_dn_beparent(pb, bind_DN);
    if (suffix_DN != NULL) {
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            msgId,
            connId,
            opId,
            "test_dns in test-dns plug-in",
            "Suffix for user (%s) is (%s).\n", bind_DN, suffix_DN
        );
    }

    return rc;
}
```

Notice that the suffix and parent DNs can be the same if the tree is not complex. For example, if the bind_DN is uid=bjensen,ou=People,dc=example,dc=com, the parent is ou=People,dc=example,dc=com. In this case, the parent is the same as the suffix.

Determining Whether a Suffix Is Served Locally

Use `slapi_dn_isbesuffix()` function which takes as its two arguments a parameter block and a `char *` to the root suffix DN, as demonstrated in the following example.

EXAMPLE 5-9 Determining Whether a Suffix Is Local (`dns.c`)

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

int
test_dns(Slapi_PBlock * pb)
{
    char * bind_DN;                /* DN being used to bind */
    char * parent_DN;             /* DN of parent entry */
    char * suffix_DN;             /* DN of suffix entry */

    slapi_pblock_get(pb, SLAPI_BIND_TARGET, &bind_DN);

    /* Get the suffix DN of the DN being used to bind. */
    suffix_DN = slapi_dn_beparent(pb, bind_DN);

    /* Climb the tree to the top suffix and check if it is local. */
    while (suffix_DN != NULL &&
           !slapi_dn_isbesuffix(pb, suffix_DN)) {
        suffix_DN = slapi_dn_parent(suffix_DN);
    }
    if (suffix_DN != NULL) {
        /* Suffix is served locally. */ ;
    } else {
        /* Suffix is not served locally. */ ;
    }

    return 0;
}
```

Notice the use of `slapi_dn_isbesuffix()` to move up the tree. Notice also that `slapi_dn_parent()` keeps working away at the DN until the DN that is left is `NULL`. The parent does not necessarily correspond to an actual entry.

Getting and Setting Entry DNs

To get and set entry DNs, use `slapi_entry_get_dn()` and `slapi_entry_set_dn()`, respectively. [“Verifying Schema Compliance for an Entry” on page 108](#) demonstrates how to use `slapi_entry_set_dn()` to change the DN of a copy of an entry. Notice the use of `slapi_ch_strdup()` to ensure that the old DN in the copy is not used.

Normalizing a DN

Before comparing DNs, you can normalize the DNs to prevent the comparison from failing due to extra white space or different capitalization. You can normalize a DN with `slapi_dn_normalize()` and `slapi_dn_normalize_case()`, as shown in the following example.

EXAMPLE 5-10 Normalizing a DN (`dns.c`)

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

int
test_norm()
{
    char * test_DN;                /* Original, not normalized */
    char * copy_DN;                /* Copy that is normalized. */

    test_DN = "dc=Example,      dc=COM"; /* Prior to normalization... */

    /* When normalizing the DN with slapi_dn_normalize() and
     * slapi_dn_normalize_case(), the DN is changed in place.
     * Use slapi_ch_strdup() to work on a copy. */
    copy_DN = slapi_ch_strdup(test_DN);
    copy_DN = slapi_dn_normalize_case(copy_DN);

    return 0;
}
```

The function `slapi_dn_normalize_case()` works *directly* on the `char *` DN passed as an argument. This prepares the string for “case ignore matching” as specified, but not defined, for X.500.

Note – Use `slapi_ch_strdup()` to make a copy first if you do not want to modify the original.

Is the User the Directory Manager?

Answer this question with `slapi_dn_isroot()`. The bind for the directory superuser, however, as for anonymous users, is handled as a special case. For such users, the server front end handles the bind *before* calling preoperation bind plug-ins.

Extending Client Request Handling Using Plug-Ins

This chapter covers support in the plug-in API for modifying what Directory Server does before or after carrying out operations for clients.

Although related to the bind operation, authentication is described independently in [Chapter 7](#).

This chapter cover the following topics:

- “Preoperation and Postoperation Plug-Ins” on page 113
- “Extending the Bind Operation” on page 116
- “Extending the Search Operation” on page 120
- “Extending the Compare Operation” on page 126
- “Extending the Add Operation” on page 128
- “Extending the Modify Operation” on page 133
- “Extending the Rename Operation” on page 136
- “Extending the Delete Operation” on page 138
- “Intercepting Information Sent to the Client” on page 140

Preoperation and Postoperation Plug-Ins

When processing client requests and when sending information to clients, Directory Server calls *preoperation* and *postoperation* plug-in functions.

Preoperation Plug-Ins

Preoperation plug-ins are called before a client request is processed. Use preoperation plug-ins to validate attribute values, and to add to and delete from attributes that are provided in a request, for example.

Postoperation Plug-Ins

Postoperation plug-ins are called after a client request has been processed and all results have been sent to the client, whether or not the operation completes successfully. Use postoperation plug-ins to send alerts, or to send alarms. Also use postoperation plug-ins to perform auditing work, or to perform cleanup work.

Note – Directory Server calls preoperation and postoperation plug-ins only when an external client is involved. Internally, Directory Server also performs operations that no external client has requested.

To make this possible, Directory Server distinguishes between external operations and internal operations. External operations respond to incoming client requests. Internal operations are actions that are performed without a corresponding client request.

Several operations support the use of preoperation and postoperation plug-in functions. Operations include abandon, add, bind, compare, delete, modify, modify RDN, search, send entry, send referral, send result, and unbind. The plug-ins function regardless of the Directory Server front end contacted by the client application.

As is the case for other plug-in functions, `slapi_pblock_set()` is used in the plug-in initialization function to register preoperation and postoperation plug-in functions. `slapi_pblock_set()` is described in [Part II](#).

Registration Identifiers

Preoperation plug-in registrations use IDs of the form `SLAPI_PLUGIN_PRE_operation_FN`. Postoperation registrations use IDs of the form `SLAPI_PLUGIN_POST_operation_FN`. Here, *operation* is one of ABANDON, ADD, BIND, COMPARE, DELETE, ENTRY (sending entries to the client), MODIFY, MODRDN, REFERRAL (sending referrals to the client), RESULT (sending results to the client), SEARCH, or UNBIND.

Note – Preoperation and postoperation plug-ins can also register functions to run at Directory Server startup, using `SLAPI_PLUGIN_START_FN`, and at Directory Server shutdown, using `SLAPI_PLUGIN_STOP_FN`.

[Part II](#) describes these IDs. Refer also to *install-path/include/slapi-plugin.h* for the complete list of IDs used at build time.

The following example demonstrates how the functions are registered with Directory Server by using the appropriate registration IDs.

EXAMPLE 6-1 Registering Postoperation Functions(testpostop.c)

```

#include "slapi-plugin.h"

Slapi_PluginDesc postop_desc = {
    "test-postop",          /* plug-in identifier      */
    "Sun Microsystems, Inc.", /* vendor name             */
    "6.0",                  /* plug-in revision number */
    "Sample post-operation plug-in" /* plug-in description    */
};

static Slapi_ComponentId * postop_id; /* Used to set log */

/* Register the plug-in with the server. */
#ifdef _WIN32
__declspec(dllexport)
#endif
int
testpostop_init(Slapi_PBlock * pb)
{
    int rc = 0; /* 0 means success */
    rc |= slapi_pblock_set( /* Plug-in API version */
        pb,
        SLAPI_PLUGIN_VERSION,
        SLAPI_PLUGIN_CURRENT_VERSION
    );
    rc |= slapi_pblock_set( /* Plug-in description */
        pb,
        SLAPI_PLUGIN_DESCRIPTION,
        (void *) &postop_desc
    );
    rc |= slapi_pblock_set( /* Open log at startup */
        pb,
        SLAPI_PLUGIN_START_FN,
        (void *) testpostop_set_log
    );
    rc |= slapi_pblock_set( /* Post-op add function */
        pb,
        SLAPI_PLUGIN_POST_ADD_FN,
        (void *) testpostop_add
    );
    rc |= slapi_pblock_set( /* Post-op modify function */
        pb,
        SLAPI_PLUGIN_POST_MODIFY_FN,
        (void *) testpostop_mod
    );
    rc |= slapi_pblock_set( /* Post-op delete function */
        pb,

```

EXAMPLE 6-1 Registering Postoperation Functions (testpostop.c) (Continued)

```
        SLAPI_PLUGIN_POST_DELETE_FN,  
        (void *) testpostop_del  
    );  
    rc |= slapi_pblock_set(          /* Post-op modrdn function */  
        pb,  
        SLAPI_PLUGIN_POST_MODRDN_FN,  
        (void *) testpostop_modrdn  
    );  
    rc |= slapi_pblock_set(          /* Close log on shutdown */  
        pb,  
        SLAPI_PLUGIN_CLOSE_FN,  
        (void *) testpostop_free_log  
    );  
    /* Plug-in identifier is required for internal search. */  
    rc |= slapi_pblock_get(pb, SLAPI_PLUGIN_IDENTITY, &postop_id);  
    return (rc);  
}
```

In addition to its postoperation functions, the plug-in that is shown in the preceding example registers a function. The plug-in opens a log file at startup. The plug-in closes the log file at shutdown. For details, refer to *install-path/examples/testpostop.c*.

Location of Plug-In Examples

Plug-in examples are located in *install-path/examples/*.

Extending the Bind Operation

This section shows how to develop functions called by Directory Server before client bind operations.

Note – Pre-bind plug-in functions are often used to handle extensions to authentication. Yet, you might have to account for special cases such as binds by the directory superuser and anonymous users. Sometimes, you have to account for multiple calls to the same preoperation or postoperation plug-in function.

▼ To Set Up an Example Suffix

If you have not done so already, set up a directory instance with a suffix, `dc=example,dc=com`, containing data loaded from a sample LDIF file, *install-path/ds6/ldif/Example.ldif*.

1 Create a new Directory Server instance.

For example:

```
$ dsadm create /local/ds
Choose the Directory Manager password:
Confirm the Directory Manager password:
$
```

2 Start the new Directory Server instance.

For example:

```
$ dsadm start /local/ds
Server started: pid=4705
$
```

3 Create a suffix called dc=example,dc=com.

For example, with long lines folded for the printed page:

```
$ dsconf create-suffix -h localhost -p 1389 dc=example,dc=com
Enter "cn=directory manager" password:
Certificate "CN=defaultCert, CN=hostname:1636" presented by the
server is not trusted.
Type "Y" to accept, "y" to accept just once,
"n" to refuse, "d" for more details: Y
$
```

4 Load the sample LDIF.

For example, with long lines folded for the printed page:

```
$ dsconf import -h localhost -p 1389 \
/opt/SUNWdsee/ds6/ldif/Example.ldif dc=example,dc=com
Enter "cn=directory manager" password:
New data will override existing data of the suffix
"dc=example,dc=com".
Initialization will have to be performed on replicated suffixes.
Do you want to continue [y/n] ? y

## Index buffering enabled with bucket size 16
## Beginning import job...
## Processing file "/opt/SUNWdsee/ds6/ldif/Example.ldif"
## Finished scanning file "/opt/SUNWdsee/ds6/ldif/Example.ldif" (160 entries)
## Workers finished; cleaning up...
## Workers cleaned up.
## Cleaning up producer thread...
## Indexing complete.
## Starting numsubordinates attribute generation.
This may take a while, please wait for further activity reports.
## Numsubordinates attribute generation complete. Flushing caches...
## Closing files...
```

```
## Import complete. Processed 160 entries in 5 seconds.  
(32.00 entries/sec)
```

```
Task completed (slapd exit code: 0).  
$
```

See Also You can use Directory Service Control Center to perform this task. For more information, see the Directory Service Control Center online help.

Logging the Authentication Method

The following example logs the bind authentication method. Refer to *install-path/examples/testpreop.c* for complete example code.

EXAMPLE 6-2 Logging the Authentication Method (testpreop.c)

```
#include "slapi-plugin.h"  
  
int  
testpreop_bind(Slapi_PBlock * pb)  
{  
    char * auth;                /* Authentication type */  
    char * dn;                  /* Target DN */  
    int method;                 /* Authentication method */  
    int connId, opId, rc = 0;  
    long msgId;  
  
    /* Get target DN for bind and authentication method used. */  
    rc |= slapi_pblock_get(pb, SLAPI_BIND_TARGET, &dn);  
    rc |= slapi_pblock_get(pb, SLAPI_BIND_METHOD, &method);  
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID, &msgId);  
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID, &connId);  
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID, &opId);  
    if (rc == 0) {  
        switch (method) {  
            case LDAP_AUTH_NONE: auth = "No authentication";  
                break;  
            case LDAP_AUTH_SIMPLE: auth = "Simple authentication";  
                break;  
            case LDAP_AUTH_SASL: auth = "SASL authentication";  
                break;  
            default: auth = "Unknown authentication method";  
                break;  
        }  
    } else {  
        return (rc);  
    }  
}
```

EXAMPLE 6-2 Logging the Authentication Method (testpreop.c) (Continued)

```

    }

    /* Log target DN and authentication method info.          */
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "testpreop_bind in test-preop plug-in",
        "Target DN: %s\tAuthentication method: %s\n", dn, auth
    );
    return (rc);
}

```

This plug-in function sets the auth message based on the authentication method. The function does nothing to affect the way Directory Server processes the bind.

▼ To Register the Plug-In

If you have not already done so, build the example plug-in library and activate both plug-in informational logging and the example plug-in.

1 Build the plug-in.

Hint Use *install-path/examples/Makefile* or *install-path/examples/Makefile64*.

2 Configure Directory Server to log plug-in informational messages and load the plug-in.

Hint Use the commands specified in the comments at the outset of the plug-in source file.

3 Restart Directory Server.

```
$ dsadm restart instance-path
```

▼ To Generate a Bind Log Message

1 Bind as Kirsten Vaughan (for example).

```
$ ldapsearch -h localhost -p 1389 -b "dc=example,dc=com" \
-D "uid=kvaughan,ou=people,dc=example,dc=com" -w bribery "(uid=*)"
```

- 2 **Search** *install-path/logs/errors* for the resulting message from the `testpreop_bind()` function.

If you ignore housekeeping information for the entry, output similar to this appears:

```
Target DN: uid=kvaughan,ou=people,dc=example,dc=com
Authentication method: Simple authentication
```

For a discussion of less trivial pre-bind plug-in functions, refer to [Chapter 7](#).

Bypassing Bind Processing in Directory Server

When the plug-in returns 0, Directory Server continues to process the bind. To bypass Directory Server bind processing, set `SLAPI_CONN_DN` in the parameter block, and return a positive value, such as 1.

Normal Directory Server Bind Behavior

Directory Server follows the LDAP bind model. At minimum, the server authenticates the client. The server also sends a bind response to indicate the status of authentication. Refer to [RFC 1777](http://www.ietf.org/rfc/rfc1777.txt) (<http://www.ietf.org/rfc/rfc1777.txt>), *Lightweight Directory Access Protocol*, and [RFC 4511](http://www.ietf.org/rfc/rfc4511.txt) (<http://www.ietf.org/rfc/rfc4511.txt>), *Lightweight Directory Access Protocol (v3)*, for details.

Note – Lightweight Directory Access Protocol (v3) is the preferred protocol because Lightweight Directory Access Protocol (v2) is obsolete.

Extending the Search Operation

This section shows how to develop functionality called by Directory Server before LDAP search operations.

Logging Who Requests a Search

The following example logs the DN of the client that requests the search. Refer to *install-path/examples/testbind.c* for complete example code.

Before using the plug-in function as shown in this section, set up the example suffix and register the plug-in. See “[Extending the Bind Operation](#)” on [page 116](#) and “To register the Plug-in”, as described previously. The plug-in, Test Bind, also includes the pre-search function.

The `test_search()` function logs the request, as shown in the following example.

EXAMPLE 6-3 Getting the DN of the Client Requesting a Search (testbind.c)

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

int
test_search(Slapi_PBlock * pb)
{
    char * requestor_dn;          /* DN of client searching */
    int    is_repl;               /* Is this replication? */
    int    is_intl;              /* Is this an internal op? */
    int    connId, opId, rc = 0;
    long   msgId;

    rc |=slapi_pblock_get(pb, SLAPI_OPERATION_MSGID,      &msgId);
    rc |=slapi_pblock_get(pb, SLAPI_CONN_ID,              &connId);
    rc |=slapi_pblock_get(pb, SLAPI_OPERATION_ID,         &opId);
    rc |=slapi_pblock_get(pb, SLAPI_REQUESTOR_DN,         &requestor_dn);
    rc |=slapi_pblock_get(pb, SLAPI_IS_REPLICATED_OPERATION, &is_repl);
    rc |=slapi_pblock_get(pb, SLAPI_IS_INTERNAL_OPERATION, &is_intl);
    if (rc != 0) return (rc);

    /* Do not interfere with replication and internal operations. */
    if (is_repl || is_intl) return 0;

    if (requestor_dn != NULL && *requestor_dn != '\0') {
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            msgId,
            connId,
            opId,
            "test_search in test-bind plug-in",
            "Search requested by %s\n", requestor_dn
        );
    } else {
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            msgId,
            connId,
            opId,
            "test_search in test-bind plug-in",
            "Search requested by anonymous client.\n"
        );
    }
    return (rc);
}
```

After activating the plug-in in the server, perform a search.

```
$ ldapsearch -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery \
-h localhost -p 1389 -b dc=example,dc=com uid=bjensen
```

Search *instance-path/logs/errors* for the resulting message. The last field of the log entry shows the following:

With the plug-in activated in Directory Server, perform a search as Kirsten Vaughan:

Authenticated: uid=kvaughan,ou=people,dc=example,dc=com

Breaking Down a Search Filter

“[Breaking Down a Search Filter](#)” on page 122 breaks down a search filter, logging the component parts of the filter. For complete example code, refer to *install-path/examples/testpreop.c*.

LOG Macros for Compact Code

The code for the `SLAPI_PLUGIN_PRE_SEARCH_FN` function, `testpreop_search()`, is preceded by three macros. Search *testpreop.c* for `#define LOG1(format)`. The purpose of these macros is only to render the subsequent logging statements more compact, making the code easier to decipher. The digits in `LOG1`, `LOG2`, and `LOG3` reflect the number of parameters each macro takes. The actual number of parameters that you pass to the log functions varies, as the log functions let you format strings in the style of `printf()`.

Parameter Block Contents

The parameter block passed to the pre-search function contains information about the target and scope of the search. The following example shows how to obtain this information. The scope, as specified in [RFC 4511](http://www.ietf.org/rfc/rfc4511.txt) (<http://www.ietf.org/rfc/rfc4511.txt>), *Lightweight Directory Access Protocol (v3)*, can include one of the following:

- The base object, which is the base DN entry
- single level under the base DN
- The whole subtree under the base DN

EXAMPLE 6-4 Logging Base and Scope for a Search (*testpreop.c*)

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

int
testpreop_search(Slapi_PBlock * pb)
{
```

EXAMPLE 6-4 Logging Base and Scope for a Search (testpreop.c) (Continued)

```

char          * base          = NULL; /* Base DN for search */
int           scope;          /* Base, 1 level, subtree */
int           rc = 0;

rc |= slapi_pblock_get(pb, SLAPI_SEARCH_TARGET,    &base);
rc |= slapi_pblock_get(pb, SLAPI_SEARCH_SCOPE,     &scope);
if (rc == 0) {
    switch (scope) {
    case LDAP_SCOPE_BASE:
        LOG2("Target DN: %s\tScope: LDAP_SCOPE_BASE\n", base);
        break;
    case LDAP_SCOPE_ONELEVEL:
        LOG2("Target DN: %s\tScope: LDAP_SCOPE_ONELEVEL\n", base);
        break;
    case LDAP_SCOPE_SUBTREE:
        LOG2("Target DN: %s\tScope: LDAP_SCOPE_SUBTREE\n", base);
        break;
    default:
        LOG3("Target DN: %s\tScope: unknown value %d\n", base, scope);
        break;
    }
} /* Continue processing... */

return 0;
}

```

In writing a plug-in mapping X.500 search targets to LDAP search targets, you could choose to parse and transform the base DN for searches.

The parameter block also contains the following information:

- When aliases are resolved during the search
- The Directory Server resources that the search is set to consume
- Which attribute types are to be returned

Search Filter Info

The search filter that you obtain from the parameter block must be considered. `testpreop_search()` gets the filter from the parameter block both as a `Slapi_Filter` structure, `filter`, and as a string, `filter_str`. Depending on what kind of search filter is passed to Directory Server and retrieved in the `Slapi_Filter` structure, the plug-in breaks the filter down in different ways. If the function were post-search rather than pre-search, you would likely want to access the results that are returned through the parameter block. See [Part II](#) for instructions on accessing the results.

The plug-in API offers several functions for manipulating the search filter. `testpreop_search()` uses `slapi_filter_get_choice()` to determine the kind of filter used. The preoperation search function also uses `slapi_filter_get_subfilt()` to break down substrings that are used in the filter. The preoperation search function further uses `slapi_filter_get_type()` to find the attribute type when searching for the presence of an attribute. The preoperation search function uses `slapi_filter_get_ava()` to obtain attribute types and values used for comparisons.

The following example shows a code excerpt that retrieves information from the search filter.

EXAMPLE 6-5 Retrieving Filter Information (`testpreop.c`)

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

int
testpreop_search(Slapi_PBlock * pb)
{
    char      * attr_type = NULL; /* For substr and compare */
    char      * substr_init= NULL; /* For substring filters */
    char      * substr_final=NULL;
    char      ** substr_any = NULL;
    int        i, rc =0;
    Slapi_Filter * filter;

    rc |= slapi_pblock_get(pb, SLAPI_SEARCH_FILTER,    &filter);
    if (rc == 0) {
        filter_type = slapi_filter_get_choice(filter);
        switch (filter_type) {
            case LDAP_FILTER_AND:
            case LDAP_FILTER_OR:
            case LDAP_FILTER_NOT:
                LOG1("Search filter: complex boolean\n");
                break;
            case LDAP_FILTER_EQUALITY:
                LOG1("Search filter: LDAP_FILTER_EQUALITY\n");
                break;
            case LDAP_FILTER_GE:
                LOG1("Search filter: LDAP_FILTER_GE\n");
                break;
            case LDAP_FILTER_LE:
                LOG1("Search filter: LDAP_FILTER_LE\n");
                break;
            case LDAP_FILTER_APPROX:
                LOG1("Search filter: LDAP_FILTER_APPROX\n");
                break;
            case LDAP_FILTER_SUBSTRINGS:
                LOG1("Search filter: LDAP_FILTER_SUBSTRINGS\n");
                slapi_filter_get_subfilt(
```

EXAMPLE 6-5 Retrieving Filter Information (testpreop.c) (Continued)

```

        filter,
        &attr_type,
        &substr_init,
        &substr_any,
        &substr_final
    );
    if (attr_type != NULL)
        LOG2("\tAttribute type: %s\n", attr_type);
    if (substr_init != NULL)
        LOG2("\tInitial substring: %s\n", substr_init);
    if (substr_any != NULL)
        for (i = 0; substr_any[i] != NULL; ++i) {
            LOG3("\tSubstring# %d: %s\n", i, substr_any[i]);
        }
    if (substr_final != NULL)
        LOG2("\tFinal substring: %s\n", substr_final);
    break;
case LDAP_FILTER_PRESENT:
    LOG1("Search filter: LDAP_FILTER_PRESENT\n");
    slapi_filter_get_attribute_type(filter, &attr_type);
    LOG2("\tAttribute type: %s\n", attr_type);
    break;
case LDAP_FILTER_EXTENDED:
    LOG1("Search filter: LDAP_FILTER_EXTENDED\n");
    break;
default:
    LOG2("Search filter: unknown type %d\n", filter_type);
    break;
}
}
return (rc);
}

```

Before using the plug-in function as described here, set up the example suffix and register the plug-in. See [“Extending the Bind Operation” on page 116](#) and [“To register the Plug-in”](#), as described previously.

After the example suffix and plug-in have been loaded into the directory, run a search to generate output in *instance-path/logs/errors*.

```
$ ldapsearch -h localhost -p 1389 -b "dc=example,dc=com" "(uid=*go*iv*)"
```

When you ignore the housekeeping information in the error log, the search filter breakdown occurs as shown in the following example.

EXAMPLE 6-6 Search Filter Breakdown

```
*** PREOPERATION SEARCH PLUG-IN - START ***
Target DN: dc=example,dc=com      Scope: LDAP_SCOPE_SUBTREE
Dereference setting: LDAP_DEREF_NEVER
Search filter: LDAP_FILTER_SUBSTRINGS
    Attribute type: uid
    Substring# 0: go
    Substring# 1: iv
String representation of search filter: (uid=*go*iv*)
*** PREOPERATION SEARCH PLUG-IN - END ***
```

Building a Filter Info

When building a filter, notice that the plug-in API provides `slapi_str2filter()` to convert strings to `Slapi_Filter` data types, and `slapi_filter_join()` to join simple filters with boolean operators AND, OR, or NOT. Free the filter with `slapi_filter_free()`. Refer to [Part II](#) for details.

Normal Directory Server Search Behavior

Directory Server gets a list of candidate entries, then iterates through the list to check which entries match the search criteria. The plug-in then puts the set of results in the parameter block. In most cases, searching within a plug-in is typically performed with `slapi_search_internal*()` calls.

Extending the Compare Operation

This section shows how to develop functionality called by Directory Server before a client compare operation. The following example logs the target DN and attribute of the entry with which to compare values. For complete example code, refer to *install-path/examples/testpreop.c*.

EXAMPLE 6-7 Plug-In Comparison Function (testpreop.c)

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

int
testpreop_cmp(Slapi_PBlock * pb)
{
    char * dn;                /* Target DN */
    char * attr_type;         /* Type of attr to compare */
    /* Attribute value could be lots of things, even a binary file.
     * Here, do not try to retrieve the value to compare. */
    int connId, opId, rc = 0;
```

EXAMPLE 6-7 Plug-In Comparison Function (testpreop.c) *(Continued)*

```

long    msgId;

rc |= slapi_pblock_get(pb, SLAPI_COMPARE_TARGET, &dn);
rc |= slapi_pblock_get(pb, SLAPI_COMPARE_TYPE, &attr_type);
rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID, &msgId);
rc |= slapi_pblock_get(pb, SLAPI_CONN_ID, &connId);
rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID, &opId);
if (rc == 0) {
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "testpreop_cmp in test-preop plug-in",
        "Target DN: %s\tAttribute type: %s\n", dn, attr_type
    );
}

return (rc);
}

```

The plug-in function can access the attribute value to use for the comparison as well. The attribute value in the parameter block is in a `berval` structure. Thus, the value could be binary data such as a JPEG image. No attempt is made to write the value to the logs.

The following example shows the `slapi_pblock_get()` call used to obtain the attribute value.

EXAMPLE 6-8 Obtaining the Attribute Value

```

#include "slapi-plugin.h"

int
my_compare_fn(Slapi_PBlock * pb)
{
    int            rc = 0;
    struct berval * attr_val;

    /* Obtain the attribute value from the parameter block */
    rc |= slapi_pblock_get(pb, SLAPI_COMPARE_VALUE, &attr_val);

    if (rc != 0) {
        rc = LDAP_OPERATIONS_ERROR;
        slapi_send_ldap_result(pb, rc, NULL, NULL, 0, NULL);
        return 0;
    }
}

```

EXAMPLE 6-8 Obtaining the Attribute Value (Continued)

```
    }

    /* Do something with the value here.                */

    return 0;
}
```

Before using the plug-in function as described here, set up the example suffix and register the plug-in. See [“Extending the Bind Operation” on page 116](#) and “To register the Plug-in”, as described previously.

Try the example plug-in function using the `ldapcompare` tool installed with the Directory Server Resource Kit.

EXAMPLE 6-9 Performing a Comparison

```
$ ldapcompare sn:Jensen uid=bjensen,ou=people,dc=example,dc=com
comparing type: "sn" value: "Jensen" in entry
"uid=bjensen,ou=people,dc=example,dc=com"
compare TRUE
```

The log entry in *instance-path/logs/errors* shows the following results, not including housekeeping information at the beginning of the log entry:

```
Target DN: uid=bjensen,ou=people,dc=example,dc=com    Attribute type: sn
```

Extending the Add Operation

This section shows how to develop functionality called by Directory Server before and after a client add operation.

Prepending a String to an Attribute

Example 39-10 prepends the string `ADD` to the description attribute of the entry to be added to the directory. For complete example code, refer to *instance-path/examples/testpreop.c*.

Before using the plug-in function as described here, set up the example suffix and register the plug-in. See [“Extending the Bind Operation” on page 116](#) and “To register the Plug-in”, as described previously.

To try the preoperation add function, add an entry for Quentin Cubbins who recently joined Example.com. Quentin’s LDIF entry, `quentin.ldif`, reads as shown in the following example.

EXAMPLE 6-10 LDIF Entry

```
dn: uid=qcubbins,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: qcubbins
givenName: Quentin
sn: Cubbins
cn: Quentin Cubbins
mail: qcubbins@example.com
userPassword: qcubbins
secretary: uid=bjensen,ou=People,dc=example,dc=com
description: Entry for Quentin Cubbins
```

The following example performs the prepend.

EXAMPLE 6-11 Prepending ADD to the Description of the Entry (testpreop.c)

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

int
testpreop_add(Slapi_PBlock * pb)
{
    Slapi_Entry * entry;           /* Entry to add          */
    Slapi_Attr * attribute;        /* Entry attributes      */
    Slapi_Value ** values;         /* Modified, duplicate vals*/
    int connId, opId, rc = 0;
    long msgId;

    /* Get the entry. */
    rc |= slapi_pblock_get(pb, SLAPI_ADD_ENTRY, &entry);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID, &msgId);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID, &connId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID, &opId);

    /* Prepend ADD to values of description attribute. */
    rc |= slapi_entry_attr_find(entry, "description", &attribute);
    if (rc == 0) { /* Found a description, so... */
        int nvals, i, count = 0;
        const Slapi_Value * value;

        slapi_attr_get_numvalues(attribute, &nvals);
        values = (Slapi_Value **)
            slapi_ch_malloc((nvals+1)*sizeof(Slapi_Value *));

        for ( /* ...loop for value... */
```

EXAMPLE 6-11 Prepending ADD to the Description of the Entry (testpreop.c) *(Continued)*

```

        i = slapi_attr_first_value_const(attribute, &value);
        i != -1;
        i = slapi_attr_next_value_const(attribute, i, &value)
    ) {
        /* ...prepend "ADD " */
        const char * string;
        char * tmp;
        values[count] = slapi_value_dup(value);
        string = slapi_value_get_string(values[count]);
        tmp = slapi_ch_malloc(5+strlen(string));
        strcpy(tmp, "ADD ");
        strcpy(tmp+4, string);
        slapi_value_set_string(values[count], tmp);
        slapi_ch_free((void **)&tmp);
        ++count;
    }

    values[count] = NULL;

} else {
    /* entry has no desc. */
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "testpreop_add in test-preop plug-in",
        "Entry has no description attribute.\n"
    );
}

rc = slapi_entry_attr_replace_sv(entry, "description", values);
if (rc != 0) {
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "testpreop_add in test-preop plug-in",
        "Description attribute(s) not modified.\n"
    );
}
slapi_valuearray_free(&values);

return 0;
}

```

Add Quentin's entry to the directory. For example, if the entry is in `quentin.ldif`, type:

```
$ ldapmodify -h localhost -p 1389 -a -f quentin.ldif \
-D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
adding new entry uid=qcubbins,ou=People,dc=example,dc=com

$
```

At this point, search the directory for Quentin's entry.

EXAMPLE 6-12 Searching for the Entry

```
$ ldapsearch -h localhost -p 1389 -b dc=example,dc=com uid=qcubbins
version: 1
dn: uid=qcubbins,ou=People,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
uid: qcubbins
givenName: Quentin
sn: Cubbins
cn: Quentin Cubbins
mail: qcubbins@example.com
secretary: uid=bjensen,ou=People,dc=example,dc=com
description: ADD Entry for Quentin Cubbins

$
```

Notice the value of the description attribute.

Delete Quentin's entry so you can use it again later as an example.

```
$ ldapdelete -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery \
uid=qcubbins,ou=people,dc=example,dc=com
```

Turn off the preoperation plug-in to avoid prepending ADD to all the entries that you add.

```
$ dsconf disable-plugin -h localhost -p 1389 "Test Preop"
$ dsadm restart instance-path
```

Logging the Entry to Add

“[Logging the Entry to Add](#)” on page 131 logs the entry to add. For complete example code, refer to *instance-path/examples/testpostop.c*.

Before using the plug-in function as described here, set up the example suffix and register the plug-in. See “[Extending the Bind Operation](#)” on page 116.

The `testpostop_add()` function logs the DN of the added entry and also writes the entry to a log created by the plug-in, `changelog.txt`. The location of `changelog.txt` depends on the platform, as shown in the following example. The `changelog.txt` file managed by the plug-in is not related to other change logs managed by Directory Server.

EXAMPLE 6-13 Tracking Added Entries (`testpostop.c`)

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

int
testpostop_add(Slapi_PBlock * pb)
{
    char          * dn;                /* DN of entry to add          */
    Slapi_Entry * entry;               /* Entry to add                */
    int           is_repl = 0;         /* Is this replication?        */
    int           connId, opId, rc = 0;
    long          msgId;

    rc |= slapi_pblock_get(pb, SLAPI_ADD_TARGET,          &dn);
    rc |= slapi_pblock_get(pb, SLAPI_ADD_ENTRY,           &entry);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID,      &msgId);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID,              &connId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID,         &opId);
    rc |= slapi_pblock_get(pb, SLAPI_IS_REPLICATED_OPERATION, &is_repl);

    if (rc != 0) return (rc);
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "testpostop_add in test-postop plug-in",
        "Added entry (%s)\n", dn
    );

    /* In general, do not interfere in replication operations. */
    /* Log the DN and the entry to the change log file.        */
    if (!is_repl) write_changelog(_ADD, dn, (void *) entry, 0);

    return (rc);
}
```

After activating the plug-in, add Quentin's entry:

```
$ ldapmodify -a -D uid=kvaughan,ou=people,dc=example,dc=com \
-h localhost -p 1389 -w bribery -f quentin.ldif
```

```
adding new entry uid=qcubbins,ou=People,dc=example,dc=com
```

```
$
```

Search *instance-path/logs/errors* for the log message. If you ignore housekeeping information, you get the following message:

```
Added entry (uid=qcubbins,ou=people,dc=example,dc=com)
```

Notice also the entry logged to *changelog.txt*. The entry resembles the LDIF that was added, except that the change log has time stamps attached for internal use as shown in the following example.

EXAMPLE 6-14 Example *changelog.txt* Entry

```
time: 21120506172445
dn: uid=qcubbins,ou=people,dc=example,dc=com
changetype: add
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
uid: qcubbins
givenName: Quentin
sn: Cubbins
cn: Quentin Cubbins
mail: qcubbins@example.com
secretary: uid=bjensen,ou=People,dc=example,dc=com
userPassword: qcubbins
creatorsName: cn=Directory Manager
modifiersName: cn=Directory Manager
createTimestamp: 21120506152444Z
modifyTimestamp: 21120506152444Z
```

Extending the Modify Operation

This section shows how to develop functionality called by Directory Server after a client modify operation. A preoperation plug-in, not demonstrated here, can be found in *install-path/examples/testpreop.c*. Refer to *install-path/examples/testpostop.c* for the source code discussed here.

Before using the plug-in function as described here, set up Directory Server as described in [“Logging the Entry to Add” on page 131](#), making sure to add Quentin’s entry.

The *testpostop_mod()* function logs the DN of the modified entry and also writes the entry to a log managed by the plug-in, *changelog.txt*. The location of *changelog.txt* depends on the platform, as shown in the source code.

The following example shows the code that performs the logging.

EXAMPLE 6-15 Tracking Modified Entries (testpostop.c)

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

int
testpostop_mod(Slapi_PBlock * pb)
{
    char      * dn;                /* DN of entry to modify */
    LDAPMod ** mods;              /* Modifications to apply */
    int        is_repl = 0;        /* Is this replication? */
    int        connId, opId, rc = 0;
    long       msgId;

    rc |= slapi_pblock_get(pb, SLAPI_MODIFY_TARGET,      &dn);
    rc |= slapi_pblock_get(pb, SLAPI_MODIFY_MODS,        &mods);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID,    &msgId);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID,            &connId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID,       &opId);
    rc |= slapi_pblock_get(pb, SLAPI_IS_REPLICATED_OPERATION, &is_repl);

    if (rc != 0) return (rc);
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "testpostop_mod in test-postop plug-in",
        "Modified entry (%s)\n", dn
    );

    /* In general, do not interfere in replication operations. */
    /* Log the DN and the modifications made to the change log file. */
    if (!is_repl) write_changelog(_MOD, dn, (void *) mods, 0);

    return (rc);
}
```

First, check that Quentin's entry is in the directory.

EXAMPLE 6-16 Checking the Directory for an Entry

```
$ ldapsearch -h localhost -p 1389 -b dc=example,dc=com uid=qcubbins
version: 1
dn: uid=qcubbins,ou=People,dc=example,dc=com
```

EXAMPLE 6-16 Checking the Directory for an Entry (Continued)

```

objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
uid: qcubbins
givenName: Quentin
sn: Cubbins
cn: Quentin Cubbins
mail: qcubbins@example.com
secretary: uid=bjensen,ou=People,dc=example,dc=com

```

After the plug-in is activated in Directory Server, modify Quentin's mail address.

EXAMPLE 6-17 Modifying a Mail Address

```

$ ldapmodify -h localhost -p 1389 \
  -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
dn: uid=qcubbins,ou=People,dc=example,dc=com
changetype: modify
replace: mail
mail: quentin@example.com
^D

```

Search *instance-path/logs/errors* for the log message. If you ignore housekeeping information, you get the following message:

```
Modified entry (uid=qcubbins,ou=people,dc=example,dc=com)
```

Notice also the information logged to *change log.txt* as shown in the following example.

EXAMPLE 6-18 Example *change log.txt* After Modification

```

time: 21120506181305
dn: uid=qcubbins,ou=people,dc=example,dc=com
changetype: modify
replace: mail
mail: quentin@example.com
-
replace: modifiersname
modifiersname: cn=Directory Manager
-
replace: modifytimestamp
modifytimestamp: 21120506161305Z
-

```

Extending the Rename Operation

This section demonstrates functionality called by Directory Server after a client modify RDN operation. A preoperation plug-in, not demonstrated here, can be found in *install-path/examples/testpreop.c*. Refer to *install-path/examples/testpostop.c* for the source code discussed here.

Before using the plug-in function as described here, set up Directory Server as described in [“Logging the Entry to Add” on page 131](#), making sure to add Quentin’s entry.

The `testpostop_modrdn()` function logs the DN of the modified entry and also writes the entry to a log managed by the plug-in, `changelog.txt`. The location of `changelog.txt` depends on the platform, as shown in the source code.

The following example shows the code that performs the logging.

EXAMPLE 6-19 Tracking Renamed Entries (testpostop.c)

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

int
testpostop_modrdn( Slapi_PBlock *pb )
{
    char * dn;                /* DN of entry to rename */
    char * newrdn;            /* New RDN */
    int  dflag;               /* Delete the old RDN? */
    int  is_repl = 0;         /* Is this replication? */
    int  connId, opId, rc = 0;
    long msgId;

    rc |= slapi_pblock_get(pb, SLAPI_MODRDN_TARGET,      &dn);
    rc |= slapi_pblock_get(pb, SLAPI_MODRDN_NEWRDN,      &newrdn);
    rc |= slapi_pblock_get(pb, SLAPI_MODRDN_DELOLDRDN,   &dflag);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID,    &msgId);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID,            &connId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID,       &opId);
    rc |= slapi_pblock_get(pb, SLAPI_IS_REPLICATED_OPERATION, &is_repl);

    if (rc != 0) return (rc);
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "testpostop_modrdn in test-postop plug-in",
        "Modrdn entry (%s)\n", dn
    );
}
```


EXAMPLE 6-19 Tracking Renamed Entries (testpostop.c) (Continued)

```

/* In general, do not interfere in replication operations.      */
/* Log the DN of the renamed entry, its new RDN, and the flag
 * indicating whether the old RDN was removed to the change log. */
if (!is_repl) write_changelog(_MODRDN, dn, (void *) newrdn, dflag);

return (rc);
}

```

Check that Quentin’s entry is in the directory as shown in [“Extending the Modify Operation” on page 133](#).

Last weekend, Quentin decided to change his given name to Fred. His user ID is now fcubbins, so his entry must be renamed. With this plug-in activated in Directory Server, modify the entry.

EXAMPLE 6-20 Renaming an Entry

```

$ ldapmodify -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery \
  -h localhost -p 1389
dn: uid=qcubbins,ou=People,dc=example,dc=com
changetype: modify
replace: givenName
givenName: Fred

dn: uid=qcubbins,ou=People,dc=example,dc=com
changetype: modify
replace: mail
mail: fcubbins@example.com

dn: uid=qcubbins,ou=People,dc=example,dc=com
changetype: modify
replace: cn
cn: Fred Cubbins

dn: uid=qcubbins,ou=People,dc=example,dc=com
changetype: modrdn
newrdn: uid=fcubbins
deleteoldrdn: 1
^D

```

Search *instance-path/logs/errors* for the log message. If you ignore housekeeping information, you get the following message:

```
Modrdn entry (uid=qcubbins,ou=people,dc=example,dc=com)
```

Notice also the information that is logged to `changelog.txt`.

EXAMPLE 6-21 Example `changelog.txt` Entry After Rename

```
time: 21120506184432
dn: uid=qcubbins,ou=people,dc=example,dc=com
changetype: modrdn
newrdn: uid=fcubbins
deleteoldrdn: 1
```

Extending the Delete Operation

This section shows how to develop functionality called by Directory Server after a client delete operation. A preoperation plug-in, not demonstrated here, can be found in *install-path/examples/testpreop.c*. Refer to *install-path/examples/testpostop.c* for the source code discussed here.

Before using the plug-in function as described here, set up Directory Server as described in [“Logging the Entry to Add” on page 131](#), making sure to add Quentin’s entry.

The `testpostop_modrdn()` function logs the DN of the modified entry and also writes the entry to a log managed by the plug-in, `changelog.txt`. The location of `changelog.txt` depends on the platform, as shown in the source code.

The following example shows the code that performs the logging.

EXAMPLE 6-22 Tracking Entry Deletion (`testpostop.c`)

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

int
testpostop_del( Slapi_PBlock *pb )
{
    char * dn;                                /* DN of entry to delete */
    int   is_repl = 0;                        /* Is this replication? */
    int   connId, opId, rc = 0;
    long  msgId;

    rc |= slapi_pblock_get(pb, SLAPI_DELETE_TARGET, &dn);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID, &msgId);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID, &connId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID, &opId);
    rc |= slapi_pblock_get(pb, SLAPI_IS_REPLICATED_OPERATION, &is_repl);

    if (rc != 0) return (rc);
    slapi_log_info_ex(
```

EXAMPLE 6-22 Tracking Entry Deletion (testpostop.c) (Continued)

```

        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "testpostop_del in test-postop plug-in",
        "Deleted entry (%s)\n", dn
    );

    /* In general, do not interfere in replication operations.      */
    /* Log the DN of the deleted entry to the change log.          */
    if (!is_repl) write_changelog(_DEL, dn, NULL, 0);

    return (rc);
}

```

First, check that Quentin's entry is in the directory as shown in [“Extending the Modify Operation” on page 133](#).

Quentin's name might be Fred if you have renamed the entry as described in [“Extending the Rename Operation” on page 136](#).

Imagine that Quentin shouted copious verbal abuse at a key customer causing Quentin to be fired from Example.com. With this plug-in activated in Directory Server, delete his entry.

```
$ ldapdelete -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery \
uid=qcubbins,ou=People,dc=example,dc=com
```

Search *instance-path/logs/errors* for the log message. If you ignore housekeeping information, you get the following message:

```
Deleted entry (uid=qcubbins,ou=people,dc=example,dc=com)
```

Notice also the information logged to *changelog.txt* as shown in the following example.

EXAMPLE 6-23 Example *changelog.txt* After Deletion

```
time: 21120506185404
dn: uid=qcubbins,ou=people,dc=example,dc=com
changetype: delete
```

Intercepting Information Sent to the Client

This section demonstrates functionality called by Directory Server before sending a result code, a referral, or an entry to the client application. Refer to *install-path/examples/testpreop.c* for the source code discussed here.

The following example shows the code that logs the operation number and user DN.

EXAMPLE 6-24 Logging and Responding to the Client (*testpreop.c*)

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

int
testpreop_send(Slapi_PBlock * pb)
{
    Slapi_Operation * op;           /* Operation in progress */
    char * connDn;                  /* Get DN from connection */
    int connId, opId, rc = 0;
    long msgId;

    rc |= slapi_pblock_get(pb, SLAPI_OPERATION, &op);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_DN, &connDn);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID, &connId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID, &msgId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID, &opId);
    if (rc == 0) {
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            msgId,
            connId,
            opId,
            "testpreop_send in test-preop plug-in",
            "Operation: %d\tUser: %s\n", slapi_op_get_type(op), connDn
        );
    }
    slapi_ch_free_string(&connDn);

    return (rc);
}
```

Operation identifiers are defined in the plug-in header file *install-path/include/slapi-plugin.h*. Search for `SLAPI_OPERATION_*`.

Before using the plug-in function as described here, set up the example suffix and register the plug-in. See [“Extending the Bind Operation” on page 116](#) and [“To register the Plug-in”](#), as described previously.

With the plug-in activated in Directory Server, perform a search as Kirsten Vaughan:

```
$ ldapsearch -h localhost -p 1389 -b dc=example,dc=com \  
-D uid=kvaughan,ou=people,dc=example,dc=com -w bribery \  
uid=bcubbins
```

Search *instance-path/logs/errors* for the log messages. Minus housekeeping information, the first message reflects the bind result:

```
Operation: 1 User: uid=kvaughan,ou=people,dc=example,dc=com
```

The next message reflects the search:

```
Operation: 4 User: uid=kvaughan,ou=people,dc=example,dc=com
```

Inside plug-in functions, use `slapi_op_get_type()` to determine the type of an operation. Refer to [Part II](#) for info about `slapi_op_get_type()`, or see the plug-in header file *install-path/include/slapi-plugin.h* for a list of operation types.

Handling Authentication Using Plug-Ins

This chapter explains how to write a plug-in that adds to, bypasses, or replaces the authentication mechanisms that Directory Server supports. The chapter demonstrates the use of existing mechanisms for authentication. You must adapt the code examples to meet your particular authentication requirements.



Caution – The examples alone do *not* provide secure authentication methods.

This chapter covers the following topics:

- “How Authentication Works” on page 143
- “How a Plug-In Modifies Authentication” on page 146
- “Developing a Simple Authentication Plug-In” on page 146
- “Developing a SASL Authentication Plug-In” on page 155

How Authentication Works

This section identifies which authentication methods are available. This section also describes how Directory Server handles authentication to identify clients. Consider the Directory Server model described in this section when writing plug-ins to modify the mechanism.

Support for Standard Methods

Directory Server supports the two authentication methods described in [RFC 4511](http://www.ietf.org/rfc/rfc4511.txt) (<http://www.ietf.org/rfc/rfc4511.txt>). One method is simple authentication, which is rendered more secure through the use of Secure Socket Layer (SSL) for transport. The other method is SASL, whose technology is further described in [RFC 2222](http://www.ietf.org/rfc/rfc2222.txt) (<http://www.ietf.org/rfc/rfc2222.txt>), *Simple Authentication and Security Layer (SASL)*.

Through SASL, Directory Server supports Kerberos authentication to the LDAP server and the Directory System Agent (DSA, an X.500 term) as described in [RFC 1777](http://www.ietf.org/rfc/rfc1777.txt) (<http://www.ietf.org/rfc/rfc1777.txt>), *Lightweight Directory Access Protocol*.

Client Identification During the Bind

For LDAP clients, Directory Server keeps track of client identity through the DN the client used to connect. The server also keeps track through the authentication method and external credentials that the client uses to connect. The parameter block holds the relevant client connection information. The DN can be accessed through the `SLAPI_CONN_DN` and `SLAPI_CONN_AUTHTYPE` parameters to `slapi_pblock_set()` and `slapi_pblock_get()`.

For DSML clients that connect over HTTP, Directory Server performs identity mapping for the bind. As a result, plug-ins have the same view of the client bind, regardless of the front—end protocol.

Bind Processing in Directory Server

Before Directory Server calls a preoperation bind plug-in, Directory Server completes authentication for anonymous binds, binds by the Directory Manager, and binds by replication users before calling preoperation bind functions. Thus, the server completes the bind without calling the plug-in.

Note – For SASL authentication mechanisms, preoperation and postoperation bind functions can be called several times during processing of a single authentication request.

In fact, multiple LDAP bind operations can be used to implement the authentication mechanism, as is the case for DIGEST-MD5, for example.

How Directory Server Processes the Bind

To process the bind, Directory Server, does the following:

1. Parses the bind request
2. Determines the authentication method
3. Determines whether the bind DN is handled locally
4. Adds request information to the parameter block
5. Determines whether to handle the bind in the front end or to call preoperation bind plug-in functions
6. Performs the bind or not, using information about the bind DN entry from the server back end

Following is a description of each action:

- **Parsing the bind request.** When a bind request arrives, Directory Server retrieves the DN, the authentication method used, and any credentials sent, such as a password. The processing can involve a mapping if the client accesses the server by using DSML over HTTP. If the request calls for SASL authentication, LDAP_AUTH_SASL, Directory Server retrieves the SASL mechanism. Next, Directory Server normalizes the DN that is retrieved from the request. The server also retrieves any LDAP v3 controls that are included in the request.
- **Determining the authentication method.** In some cases, the method is simple authentication, but the DN or credentials are empty or missing. Directory Server then assumes that the client is binding anonymously, sets SLAPI_CONN_DN to NULL and SLAPI_CONN_AUTHTYPE to SLAPD_AUTH_NONE, and sends an LDAP_SUCCESS result to the client. If the method is SASL authentication, Directory Server first determines whether the mechanism is supported. If not, the server sends an LDAP_AUTH_METHOD_NOT_SUPPORTED result to the client.
- **Determining whether the DN is handled locally.** If the DN is not stored in the local instance, but Directory Server refers clients by default, the server sends the client an LDAP_REFERRAL result. Otherwise, the server sends an LDAP_NO_SUCH_OBJECT result to the client.
- **Adding request information to the parameter block.** Directory Server puts bind request information into the parameter block:
 - SLAPI_BIND_TARGET to the DN retrieved from the request
 - SLAPI_BIND_METHOD to the authentication method requested
 - SLAPI_BIND_CREDENTIALS to the credentials retrieved from the request
 - SLAPI_BIND_MECHANISM to the name of the SASL mechanism, if applicable
- **Determining whether to handle the Bind or call a plug-in.** Directory Server authenticates bind requests from the directory superuser or the replication manager, sending an LDAP_SUCCESS result on success or an LDAP_INVALID_CREDENTIALS result on failure.
After completing all this work, the server calls preoperation bind functions.
 Directory Server does not call preoperation bind functions for bind requests from the directory manager, the replication manager, nor for anonymous binds.
- **Performing the back-end bind.** When preoperation bind functions return 0, Directory Server completes the bind operation. The server checks the bind against the information in the directory database. The server then sets SLAPI_CONN_DN and SLAPI_CONN_AUTHTYPE appropriately. If necessary, Directory Server sends password-related result controls to the client. The server always sends an LDAP_SUCCESS result on success or a non zero result on failure.

How a Plug-In Modifies Authentication

A preoperation bind function can modify Directory Server authentication in one of two ways. The plug-in either completely bypasses the comparison of incoming authentication information to authentication information stored in the directory database or implements a custom SASL mechanism.

Bypassing Authentication

Some plug-ins bypass the comparison of authentication information in the client request to authentication information in the directory. Such plug-ins return nonzero values. A value of 1 prevents the server from completing the bind after the preoperation function returns. Use this approach when you store all authentication information outside the directory, without mapping authentication identities through LDAP or the plug-in API. In addition to the other validation of the plug-in, you must verify that the plug-in works well with server access control mechanisms.

Refer to [“Developing a Simple Authentication Plug-In” on page 146](#) for an example.

Using Custom SASL Mechanisms

If the plug-in implements a custom SASL mechanism, clients that use that mechanism must support it as well.

Refer to [“Developing a SASL Authentication Plug-In” on page 155](#) for a plug-in example.

Developing a Simple Authentication Plug-In

This section shows how a preoperation bind plug-in can use the plug-in API to authenticate a user. The example used in this section obtains the appropriate bind information from the parameter block. The example then handles the authentication if the request is for LDAP_AUTH_SIMPLE, but allows the server to continue the bind if the authentication succeeds.

Notice that some binds are performed by the front end before preoperation bind functions are called.

Locating the Simple Authentication Example

The following example shows a code excerpt from the source file *install-path/examples/testbind.c*.

EXAMPLE 7-1 Preoperation Bind Function (testbind.c)

```

#include "slapi-plugin.h"

int
test_bind(Slapi_PBlock * pb)
{
    char          * dn;           /* Target DN */
    int           method;        /* Authentication method */
    struct berval * credentials;  /* Client SASL credentials */
    Slapi_DN      * sdn = NULL;  /* DN used in internal srch */
    char          * attrs[2] = { /* Look at userPassword only */
                                SLAPI_USERPWD_ATTR,
                                NULL
    };

    Slapi_Entry * entry = NULL; /* Entry returned by srch */
    Slapi_Attr  * attr  = NULL; /* Pwd attr in entry found */
    int          is_repl;       /* Is this replication? */
    int          is_intl;       /* Is this an internal op? */
    int          connId, opId, rc = 0;
    long         msgId;

    /* Obtain the bind information from the parameter block. */
    rc |= slapi_pblock_get(pb, SLAPI_BIND_TARGET, &dn);
    rc |= slapi_pblock_get(pb, SLAPI_BIND_METHOD, &method);
    rc |= slapi_pblock_get(pb, SLAPI_BIND_CREDENTIALS, &credentials);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID, &msgId);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID, &connId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID, &opId);
    rc |= slapi_pblock_get(pb, SLAPI_IS_REPLICATED_OPERATION, &is_repl);
    rc |= slapi_pblock_get(pb, SLAPI_IS_INTERNAL_OPERATION, &is_intl);

    if (rc != 0) {
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            SLAPI_LOG_NO_MSGID,
            SLAPI_LOG_NO_CONNID,
            SLAPI_LOG_NO_OPID,
            "test_bind in test-bind plug-in",
            "Could not get parameters for bind operation (error %d).\n", rc
        );
        slapi_send_ldap_result(
            pb, LDAP_OPERATIONS_ERROR, NULL, NULL, 0, NULL);
        return (LDAP_OPERATIONS_ERROR); /* Server aborts bind here. */
    }

    /* The following code handles simple authentication, where a

```

EXAMPLE 7-1 Preoperation Bind Function (testbind.c) *(Continued)*

```

* user offers a bind DN and a password for authentication.
*
* Handling simple authentication is a matter of finding the
* entry corresponding to the bind DN sent in the request,
* then if the entry is found, checking whether the password
* sent in the request matches a value found on the
* userPassword attribute of the entry. */

/* Avoid interfering with replication or internal operations. */
if (!is_repl && !is_intl) switch (method) {
case LDAP_AUTH_SIMPLE:

    /* Find the entry specified by the bind DN... */
    sdn = slapi_sdn_new_dn_byref(dn);
    rc |= slapi_search_internal_get_entry(
        sdn,
        attrs,
        &entry,
        plugin_id
    );
    slapi_sdn_free(&sdn);

    if (rc != 0 || entry == NULL) {
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            msgId,
            connId,
            opId,
            "test_bind in test-bind plug-in",
            "Could not find entry: %s\n", dn
        );
        rc = LDAP_NO_SUCH_OBJECT;
        slapi_send_ldap_result(pb, rc, NULL, NULL, 0, NULL);
        return (rc);
    } else {
        /* ...check credentials against the userpassword... */
        Slapi_Value * credval; /* Value of credentials */
        Slapi_ValueSet * pwvalues; /* Password attribute values */

        rc |= slapi_entry_attr_find(
            entry,
            SLAPI_USERPWD_ATTR,
            &attr
        );
    }
}

```

EXAMPLE 7-1 Preoperation Bind Function (testbind.c) *(Continued)*

```

if (attr == NULL) {

    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "test_bind in test-bind plug-in",
        "Entry %s has no userpassword.\n",
        dn
    );
    rc = LDAP_INAPPROPRIATE_AUTH;
    slapi_send_ldap_result(pb, rc, NULL, NULL, 0, NULL);
    return (rc);
}

rc |= slapi_attr_get_valueset(
    attr,
    &pwvalues
);
if (rc != 0 || slapi_valueset_count(pwvalues) == 0) {
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "test_bind in test-bind plug-in",
        "Entry %s has no %s attribute values.\n",
        dn, SLAPI_USERPWD_ATTR
    );
    rc = LDAP_INAPPROPRIATE_AUTH;
    slapi_send_ldap_result(pb, rc, NULL, NULL, 0, NULL);
    slapi_valueset_free(pwvalues);
    return (rc);
}

credval = slapi_value_new_berval(credentials);

rc = slapi_pw_find_valueset(pwvalues, credval);

slapi_value_free(&credval);
slapi_valueset_free(pwvalues);

if (rc != 0) {

```

EXAMPLE 7-1 Preoperation Bind Function (testbind.c) *(Continued)*

```
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            msgId,
            connId,
            opId,
            "test_bind in test-bind plug-in",
            "Credentials are not correct.\n"
        );
        rc = LDAP_INVALID_CREDENTIALS;
        slapi_send_ldap_result(pb, rc, NULL, NULL, 0, NULL);
        return (rc);
    }

/* ...if successful, set authentication for the connection. */
if (rc != 0) return (rc);
rc |=slapi_pblock_set(pb, SLAPI_CONN_DN, slapi_ch_strdup(dn));
rc |=slapi_pblock_set(pb, SLAPI_CONN_AUTHMETHOD, SLAPD_AUTH_SIMPLE);
if (rc != 0) {
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "test_bind in test-bind plug-in",
        "Failed to set connection info.\n"
    );
    rc = LDAP_OPERATIONS_ERROR;
    slapi_send_ldap_result(pb, rc, NULL, NULL, 0, NULL);
    return (rc);
} else {
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "test_bind in test-bind plug-in",
        "Authenticated: %s\n", dn
    );

    /* Now that authentication succeeded, the plug-in
     * returns a value greater than 0, even though the
     * authentication has been successful. A return
```

EXAMPLE 7-1 Preoperation Bind Function (testbind.c) (Continued)

```

        * code > 0 tells the server not to continue
        * processing the bind. A return code of 0, such
        * as LDAP_SUCCESS tells the server to continue
        * processing the operation.                                */
        slapi_send_ldap_result(
            pb, LDAP_SUCCESS, NULL, NULL, 0, NULL);
        rc = 1;
    }
    break;

/* This plug-in supports only simple authentication.                */
case LDAP_AUTH_NONE:
    /* Anonymous binds are handled by the front-end before
    * pre-bind plug-in functions are called, so this
    * part of the code should never be reached.                    */
case LDAP_AUTH_SASL:
default:
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "test_bind in test-bind plug-in",
        "Plug-in does not handle auth. method: %d\n", method
    );
    rc = 0;                                                         /* Let server handle bind. */
    break;
}

return (rc);                                                       /* Server stops processing
                                                                    * the bind if rc > 0. */
}

```

The bulk of the code processes the `LDAP_AUTH_SIMPLE` case. In the simple authentication case, the plug-in uses the DN and the password to authenticate the user binding through plug-in API calls.

Seeing the Plug-In Work

The plug-in demonstration works by turning on informational logging for plug-ins. You read the log messages written by the plug-in at different stages in its operation. Before using the plug-in, load a few example users and data because you cannot demonstrate the functionality while binding as a directory superuser. without calling the preoperation bind functions.

▼ To Set Up an Example Suffix

If you have not done so already, set up a directory instance with a suffix, `dc=example,dc=com`, containing data loaded from a sample LDIF file, *install-path*/ds6/ldif/Example.ldif.

1 Create a new Directory Server instance.

For example:

```
$ dsadm create /local/ds
Choose the Directory Manager password:
Confirm the Directory Manager password:
$
```

2 Start the new Directory Server instance.

For example:

```
$ dsadm start /local/ds
Server started: pid=4705
$
```

3 Create a suffix called `dc=example,dc=com`.

For example, with long lines folded for the printed page:

```
$ dsconf create-suffix -h localhost -p 1389 dc=example,dc=com
Enter "cn=directory manager" password:
Certificate "CN=defaultCert, CN=hostname:1636" presented by the
server is not trusted.
Type "Y" to accept, "y" to accept just once,
"n" to refuse, "d" for more details: Y
$
```

4 Load the sample LDIF.

For example, with long lines folded for the printed page:

```
$ dsconf import -h localhost -p 1389 \
/opt/SUNWdsee/ds6/ldif/Example.ldif dc=example,dc=com
Enter "cn=directory manager" password:
New data will override existing data of the suffix
"dc=example,dc=com".
Initialization will have to be performed on replicated suffixes.
Do you want to continue [y/n] ? y

## Index buffering enabled with bucket size 16
## Beginning import job...
## Processing file "/opt/SUNWdsee/ds6/ldif/Example.ldif"
## Finished scanning file "/opt/SUNWdsee/ds6/ldif/Example.ldif" (160 entries)
## Workers finished; cleaning up...
## Workers cleaned up.
## Cleaning up producer thread...
```



```
## Indexing complete.
## Starting numsubordinates attribute generation.
  This may take a while, please wait for further activity reports.
## Numsubordinates attribute generation complete. Flushing caches...
## Closing files...
## Import complete. Processed 160 entries in 5 seconds.
  (32.00 entries/sec)

Task completed (slapd exit code: 0).
$
```

See Also You can use Directory Service Control Center to perform this task. For more information, see the Directory Service Control Center online help.

▼ To Register the Plug-In

If you have not already done so, build the example plug-in library and activate both plug-in informational logging and the example plug-in.

1 Build the plug-in.

Hint Use *install-path/examples/Makefile* or *install-path/examples/Makefile64*.

2 Configure Directory Server to log plug-in informational messages and load the plug-in.

Hint Use the commands specified in the comments at the outset of the plug-in source file.

3 Restart Directory Server.

```
$ dsadm restart instance-path
```

▼ To Bypass the Plug-In

The example suffix contains a number of people. If you look up the entry for one of those people, Barbara Jensen, either anonymously or as Directory Manager, the `test_bind()` plug-in function is never called. The plug-in therefore never logs informational messages to the errors log.

● Run a search that bypasses the plug-in.

```
$ ldapsearch -h localhost -p 1389 -b dc=example,dc=com uid=bjensen sn
version: 1
dn: uid=bjensen, ou=People, dc=example,dc=com
sn: Jensen
$ grep test_bind /local/ds/logs/errors
$
```

Notice that the server bypasses preoperation bind plug-ins when special users request a bind.

▼ To Bind as an Example.com User

1 Check what happens in the errors log when you bind as Barbara Jensen.

```
$ ldapsearch -h localhost -p 1389 -b dc=example,dc=com \
-D uid=bjensen,ou=people,dc=example,dc=com -w hifalutin uid=bjensen sn
version: 1
dn: uid=bjensen, ou=People, dc=example,dc=com
sn: Jensen
$ grep test_bind /local/ds/logs/errors
[04/Jan/2006:11:34:31 +0100] - INFORMATION -
test_bind in test-bind plug-in
- conn=4 op=0 msgId=1 -
Authenticated: uid=bjensen,ou=people,dc=example,dc=com
$
```

2 See what happens when you bind as Barbara Jensen, but get the password wrong.

```
$ ldapsearch -h localhost -p 1389 -b dc=example,dc=com \
-D uid=bjensen,ou=people,dc=example,dc=com -w bogus uid=bjensen sn
ldap_simple_bind: Invalid credentials
$ grep test_bind /local/ds/logs/errors | grep -i credentials
[04/Jan/2006:11:36:07 +0100] - INFORMATION -
test_bind in test-bind plug-in
- conn=5 op=0 msgId=1 - Credentials are not correct.
$
```

Here, the LDAP result is interpreted correctly by the command-line client. The plug-in message to the same effect is written to the errors log.

3 Delete Barbara's password, then try again.

```
$ ldapmodify -h localhost -p 1389 \
-D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
dn: uid=bjensen,ou=people,dc=example,dc=com
changetype: modify
delete: userpassword
modifying entry uid=bjensen,ou=people,dc=example,dc=com
^D
$ ldapsearch -h localhost -p 1389 -b dc=example,dc=com \
-D uid=bjensen,ou=people,dc=example,dc=com -w - uid=bjensen sn
Enter bind password:
ldap_simple_bind: Inappropriate authentication
$ grep test_bind /local/ds/logs/errors | grep -i password
[04/Jan/2006:11:41:25 +0100] - INFORMATION -
test_bind in test-bind plug-in
- conn=8 op=0 msgId=1 -
Entry uid=bjensen,ou=people,dc=example,dc=com has no userpassword.
$
```

Here, the LDAP result is displayed correctly by the command-line client. The plug-in message will provide more information about what went wrong during Barbara's attempt to bind, no userpassword attribute values.

Developing a SASL Authentication Plug-In

This section shows how a pre-bind operation plug-in can implement a custom Simple Authentication and Security Layer (SASL) mechanism. This mechanism enables mutual authentication without affecting existing authentication mechanisms.

The example plug-in responds to SASL bind requests with a mechanism, `my_sasl_mechanism`. Binds with this mechanism always succeed. The example is intended only to illustrate how a SASL authentication plug-in works, not to provide a secure mechanism for use in production.

Locating SASL Examples

This overall SASL section covers the examples `install-path/examples/testsaslbinding.c` and `install-path/examples/clients/saslclient.c`.

Before using the plug-in function as described here, set up an example suffix. The setup is described in [“Seeing the Plug-In Work” on page 151](#). Register the plug-in as described in the comments at the beginning of the sample file.

Registering the SASL Mechanism

Register the SASL mechanism by using the same function that registers the plug-in, `slapi_register_supported_saslmechanism()`. The following example shows the function that registers both the plug-in and the SASL mechanism.

EXAMPLE 7-2 Registering a Custom SASL Plug-In (`testsaslbinding.c`)

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

Slapi_PluginDesc saslpdesc = {
    "test-saslbinding",          /* plug-in identifier */
    "Sun Microsystems, Inc.",    /* vendor name */
    "6.0",                      /* plug-in revision number */
    "Sample SASL pre-bind plug-in" /* plug-in description */
};

#define TEST_MECHANISM "my_sasl_mechanism"
#define TEST_AUTHMETHOD SLAPD_AUTH_SASL TEST_MECHANISM
```

EXAMPLE 7-2 Registering a Custom SASL Plug-In (testsaslbind.c) (Continued)

```

/* Register the plug-in with the server. */
#ifdef _WIN32
__declspec(dllexport)
#endif
int
testsasl_init(Slapi_PBlock * pb)
{
    int rc = 0; /* 0 means success */
    rc |= slapi_pblock_set(
        pb,
        SLAPI_PLUGIN_VERSION,
        SLAPI_PLUGIN_CURRENT_VERSION
    );
    rc |= slapi_pblock_set( /* Plug-in description */
        pb,
        SLAPI_PLUGIN_DESCRIPTION,
        (void *) &saslpdesc
    );
    rc |= slapi_pblock_set( /* Pre-op bind SASL function */
        pb,
        SLAPI_PLUGIN_PRE_BIND_FN,
        (void *) testsasl_bind
    );

    /* Register the SASL mechanism. */
    slapi_register_supported_saslmechanism(TEST_MECHANISM);
    return (rc);
}

```

Refer to [Part II](#) for details on using `slapi_register_supported_saslmechanism()`.

The plug-in function `testsasl_bind()` first obtains the client DN, method, SASL mechanism, and credentials. If the client does not ask to use the `TEST_MECHANISM`, the function returns 0 so the server can process the bind. The following example shows how the processing is done.

EXAMPLE 7-3 Handling the `my_sasl_mechanism Bind` (testsaslbind.c)

```

#include "slapi-plugin.h"

#define TEST_MECHANISM "my_sasl_mechanism"
#define TEST_AUTHMETHOD SLAPD_AUTH_SASL TEST_MECHANISM

int
testsasl_bind(Slapi_PBlock * pb)

```

EXAMPLE 7-3 Handling the `my_sasl_mechanism Bind` (`testsaslbind.c`) (Continued)

```

{
    char          * dn;                /* Target DN                */
    int           method;              /* Authentication method    */
    char          * mechanism;         /* SASL mechanism          */
    struct berval * credentials;       /* SASL client credentials */
    struct berval svrcreds;            /* SASL server credentials */
    int           connId, opId, rc = 0;
    long          msgId;

    rc |= slapi_pblock_get(pb, SLAPI_BIND_TARGET,      &dn);
    rc |= slapi_pblock_get(pb, SLAPI_BIND_METHOD,      &method);
    rc |= slapi_pblock_get(pb, SLAPI_BIND_CREDENTIALS, &credentials);
    rc |= slapi_pblock_get(pb, SLAPI_BIND_SASLMECHANISM, &mechanism);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_MSGID,  &msgId);
    rc |= slapi_pblock_get(pb, SLAPI_CONN_ID,          &connId);
    rc |= slapi_pblock_get(pb, SLAPI_OPERATION_ID,     &opId);
    if (rc == 0) {
        if (mechanism == NULL || strcmp(mechanism, TEST_MECHANISM) != 0)
            return(rc); /* Client binding another way. */
    } else {
        /* slapi_pblock_get() failed! */
        return(rc);
    }

    /* Using the SASL mechanism that always succeeds, set conn. info. */
    rc |= slapi_pblock_set(pb, SLAPI_CONN_DN,          slapi_ch_strdup(dn));
    rc |= slapi_pblock_set(pb, SLAPI_CONN_AUTHMETHOD, TEST_AUTHMETHOD);
    if (rc != 0) { /* Failed to set conn. info! */
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            msgId,
            connId,
            opId,
            "testsasl_bind in test-saslbind plug-in",
            "slapi_pblock_set() for connection information failed.\n"
        );
        slapi_send_ldap_result(
            pb, LDAP_OPERATIONS_ERROR, NULL, NULL, 0, NULL);
        return(LDAP_OPERATIONS_ERROR); /* Server tries other mechs. */
    }

    /* Set server credentials. */
    svrcreds.bv_val = "my credentials";
    svrcreds.bv_len = sizeof("my credentials") - 1;

    rc |= slapi_pblock_set(pb, SLAPI_BIND_RET_SASLCREDS, &svrcreds);

```

EXAMPLE 7-3 Handling the `my_sasl_mechanism Bind` (`testsaslbind.c`) (Continued)

```

if (rc != 0) {
    /* Failed to set credentials! */
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        msgId,
        connId,
        opId,
        "testsasl_bind in test-saslbind plug-in",
        "slapi_pblock_set() for server credentials failed.\n"
    );
    rc |= slapi_pblock_set(pb, SLAPI_CONN_DN, NULL);
    rc |= slapi_pblock_set(pb, SLAPI_CONN_AUTHMETHOD, SLAPD_AUTH_NONE);
    return(LDAP_OPERATIONS_ERROR); /* Server tries other mechs. */
}

/* Send credentials to client. */
slapi_log_info_ex(
    SLAPI_LOG_INFO_AREA_PLUGIN,
    SLAPI_LOG_INFO_LEVEL_DEFAULT,
    msgId,
    connId,
    opId,
    "testsasl_bind in test-saslbind plug-in",
    "Authenticated: %s\n", dn
);
slapi_send_ldap_result(pb, LDAP_SUCCESS, NULL, NULL, 0, NULL);
return 1; /* Server stops processing the
           * bind if the plug-in returns
           * a value greater than 0. */
}

```

The preceding example sets the DN for the client connection and the authentication method, as Directory Server does for a bind. To set `SLAPI_CONN_DN` and `SLAPI_CONN_AUTHMETHOD` is the key step of the bind. Without these values in the parameter block, Directory Server handles the client request as if the bind were anonymous. Thus, the client is left with access rights that correspond to an anonymous bind. Next, according to the SASL model, Directory Server sends credentials to the client as shown.

Developing the SASL Client

To test the plug-in, you need a client that uses `my_sasl_mechanism` to bind. The example discussed here uses this mechanism and the client code, `saslclient.c`, which is delivered with Directory Server.

The client authenticates by using the example SASL mechanism for a synchronous bind to Directory Server. The following example shows how authentication proceeds on the client side for Ted Morris.

EXAMPLE 7-4 Client Using `my_sasl_mechanism` (clients/saslclient.c)

```
#include "ldap.h"

/* Use the fake SASL mechanism. */
#define MECH "my_sasl_mechanism"

/* Global variables for client connection information.
 * You may set these here or on the command line. */
static char * host = "localhost"; /* Server hostname */
static int port = 389; /* Server port */
/* Load <install-path>/ldif/Example.ldif
 * before trying the plug-in with this default user. */
static char * user = "uid=tmorris,ou=people,dc=example,dc=com";
/* New value for userPassword */
static char * npwd = "23skidoo";

/* Check for host, port, user and new password as arguments. */
int get_user_args(int argc, char ** argv);

int
main(int argc, char ** argv)
{
    LDAP * ld; /* Handle to LDAP connection */
    LDAPMod * modPW, * mods[2]; /* For modifying the password */
    char * vals[2]; /* Value of modified password */
    struct berval cred; /* Client bind credentials */
    struct berval * srvCred; /* Server bind credentials */
    int ldapVersion;

    /* Use default hostname, server port, user, and new password
     * unless they are provided as arguments on the command line. */
    if (get_user_args(argc, argv) != 0) return 1; /* Usage error */

    /* Get a handle to an LDAP connection. */
    printf("Getting the handle to the LDAP connection...\n");
    if ((ld = ldap_init(host, port)) == NULL) {
        perror("ldap_init");
        return 1;
    }

    /* By default, the LDAP version is set to 2. */
    printf("Setting the version to LDAP v3...\n");
    ldapVersion = LDAP_VERSION3;
```

EXAMPLE 7-4 Client Using my_sasl_mechanism(clients/saslclient.c) (Continued)

```

ldap_set_option(ld, LDAP_OPT_PROTOCOL_VERSION, &ldapVersion);

/* Authenticate using the example SASL mechanism. */
printf("Bind DN is %s...\n", user);
printf("Binding to the server using %s...\n", MECH);
cred.bv_val = "magic";
cred.bv_len = sizeof("magic") - 1;
if (ldap_sasl_bind_s(ld, user, MECH, &cred, NULL, NULL, &srvCred)) {
    ldap_perror(ld, "ldap_sasl_bind_s");
    return 1;
}

/* Display the credentials returned by the server. */
printf("Server credentials: %s\n", srvCred->bv_val);

/* Modify the user's password. */
printf("Modifying the password...\n");
modPW.mod_op = LDAP_MOD_REPLACE;
modPW.mod_type = "userpassword";
vals[0] = npwd;
vals[1] = NULL;
modPW.mod_values = vals;

mods[0] = &modPW; mods[1] = NULL;

if (ldap_modify_ext_s(ld, user, mods, NULL, NULL)) {
    ldap_perror(ld, "ldap_modify_ext_s");
    return 1;
}

/* Finish up. */
ldap_unbind(ld);
printf("Modification was successful.\n");
return 0;
}

```

Trying the SASL Client

The client changes Ted's password by binding to Directory Server, then requesting a modification to the userPassword attribute value.

After activating the plug-in in the server, compile the client code, `saslclient.c`. Next, run the client to perform the bind and the password modification.


```

$ ./saslclient
Using the following connection info:
    host:    localhost
    port:    389
    bind DN: uid=tmorris,ou=people,dc=example,dc=com
    new pwd: 23skidoo
Getting the handle to the LDAP connection...
Setting the version to LDAP v3...
Bind DN is uid=tmorris,ou=people,dc=example,dc=com...
Binding to the server using my_sasl_mechanism...
Server credentials: my credentials
Modifying the password...
Modification was successful.
$

```

On the Directory Server side, the message showing that the plug-in has authenticated the client is in the errors log.

```

$ grep tmorris /local/ds/logs/errors | grep -i sasl
[04/Jan/2006:12:05:30 +0100] - INFORMATION
- testsasl_bind in test-saslbind plug-in
- conn=12 op=0 msgId=1 -
Authenticated: uid=tmorris,ou=people,dc=example,dc=com

```


Performing Internal Operations With Plug-Ins

This chapter explains how to perform search, add, modify, modify DN, and delete operations for which no corresponding client requests exist.

Refer to [Chapter 16](#) for information about the plug-in API functions used in this chapter.

This chapter covers the following topics:

- “Using Internal Operations” on page 163
- “Finding the Internal Operations Example” on page 164
- “Before Using the Internal Operations Example” on page 164
- “Internal Add” on page 166
- “Internal Modify” on page 168
- “Internal Rename and Move (Modify DN)” on page 170
- “Internal Search” on page 171
- “Internal Delete” on page 174

Using Internal Operations

This chapter shows how to use internal search, add, modify, modify RDN, and delete operations.

When to Use Internal Operations

Internal operations are *internal* in the sense that they are initiated not by external requests from clients, but internally by plug-ins. Use internal operation calls when your plug-in needs Directory Server to perform an operation for which no client request exists.

Issues With Internal Operations

You set up the parameter blocks and handle all memory management directly when developing with internal operations. Debug sessions with optimized binaries such as, the libraries that are delivered with the product, can be tedious. Review the code carefully. If you want to, work with a partner who can point out errors that you miss. Memory management mistakes around internal operation calls lead more quickly to incomprehensible segmentation faults than other calls in the plug-in API.

Furthermore, internal operations result in updates to some internal caches but not others. For example, changes to access control instructions cause updates to the access control cache. Internal operation changes to attributes used in CoS and roles do not cause CoS and roles caches to be updated.

Finding the Internal Operations Example

The plug-in code used in this chapter can be found in *install-path/examples/internal.c*.

Before Using the Internal Operations Example

Before using the example plug-in *internal.c*, create a new suffix, *dc=example*, *dc=com*, and populate the suffix with example data.

▼ To Set Up an Example Suffix

If you have not done so already, set up a directory instance with a suffix, *dc=example*, *dc=com*, containing data loaded from a sample LDIF file, *install-path/ds6/ldif/Example.ldif*.

1 Create a new Directory Server instance.

For example:

```
$ dsadm create /local/ds
Choose the Directory Manager password:
Confirm the Directory Manager password:
$
```

2 Start the new Directory Server instance.

For example:

```
$ dsadm start /local/ds
Server started: pid=4705
$
```

3 Create a suffix called dc=example,dc=com.

For example, with long lines folded for the printed page:

```
$ dsconf create-suffix -h localhost -p 1389 dc=example,dc=com
Enter "cn=directory manager" password:
Certificate "CN=defaultCert, CN=hostname:1636" presented by the
server is not trusted.
Type "Y" to accept, "y" to accept just once,
"n" to refuse, "d" for more details: Y
$
```

4 Load the sample LDIF.

For example, with long lines folded for the printed page:

```
$ dsconf import -h localhost -p 1389 \
/opt/SUNWdsee/ds6/ldif/Example.ldif dc=example,dc=com
Enter "cn=directory manager" password:
New data will override existing data of the suffix
"dc=example,dc=com".
Initialization will have to be performed on replicated suffixes.
Do you want to continue [y/n] ? y

## Index buffering enabled with bucket size 16
## Beginning import job...
## Processing file "/opt/SUNWdsee/ds6/ldif/Example.ldif"
## Finished scanning file "/opt/SUNWdsee/ds6/ldif/Example.ldif" (160 entries)
## Workers finished; cleaning up...
## Workers cleaned up.
## Cleaning up producer thread...
## Indexing complete.
## Starting numsubordinates attribute generation.
This may take a while, please wait for further activity reports.
## Numsubordinates attribute generation complete. Flushing caches...
## Closing files...
## Import complete. Processed 160 entries in 5 seconds.
(32.00 entries/sec)

Task completed (slapd exit code: 0).
$
```

See Also You can use Directory Service Control Center to perform this task. For more information, see the Directory Service Control Center online help.

Internal Add

For an internal add, you allocate space for a parameter block. You set up the parameter block for the add with `slapi_add_entry_internal_set_pb()`. Thus, the entry is in the parameter block when you invoke `slapi_add_internal_pb()`. Then you free the parameter block. The internal add consumes the entry that is passed in to the server through the parameter block.

EXAMPLE 8-1 Internal Add Operation (internal.c)

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

/* Internal operations require an ID for the plug-in. */
static Slapi_ComponentId * plugin_id = NULL;

int
test_internal()
{
    Slapi_DN      * sdn;           /* DN holder for internal ops */
    Slapi_Entry   * entry;        /* Entry holder for internal ops */
    Slapi_PBlock   * pb;          /* PBlock for internal ops */
    char          * str = NULL;   /* String holder for internal ops */
    int            len;           /* Length of LDIF from entry */
    int            rc;            /* Return code; 0 means success. */

    /* Check that the example suffix exists. */
    sdn = slapi_sdn_new_dn_byval("dc=example,dc=com");
    if (slapi_be_exist(sdn)) {
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            SLAPI_LOG_NO_MSGID,
            SLAPI_LOG_NO_CONNID,
            SLAPI_LOG_NO_OPID,
            "test_internal in test-internal plug-in",
            "Suffix (%s) does not exist, exiting.\n",
            slapi_sdn_get_dn(sdn)
        );
        slapi_sdn_free(&sdn);
        return (0);
    }
    slapi_sdn_free(&sdn);

    /*
     * Add an entry for Quentin Cubbins to the example suffix
     * using slapi_add_entry_internal().
     */
    entry = slapi_entry_alloc();
    slapi_entry_set_dn(
        entry,
        /* slapi_entry_set_dn()
         * requires a copy of the DN.
         */
    );
}
```

EXAMPLE 8-1 Internal Add Operation (internal.c) (Continued)

```

    slapi_ch_strdup("uid=qcubbins,ou=People,dc=example,dc=com")
);

/* slapi_entry_add_string() */
/* does not require a copy. */

slapi_entry_add_string(entry, "objectclass", "top");
slapi_entry_add_string(entry, "objectclass", "person");
slapi_entry_add_string(entry, "objectclass", "organizationalPerson");
slapi_entry_add_string(entry, "objectclass", "inetOrgPerson");
slapi_entry_add_string(entry, "uid", "qcubbins");
slapi_entry_add_string(entry, "givenName", "Quentin");
slapi_entry_add_string(entry, "sn", "Cubbins");
slapi_entry_add_string(entry, "cn", "Quentin Cubbins");
slapi_entry_add_string(entry, "mail", "qcubbins@example.com");
slapi_entry_add_string(entry, "userPassword", "qcubbins");
slapi_entry_add_string(entry, "secretary",
    "uid=bjensen,ou=People,dc=example,dc=com");

pb = slapi_pblock_new(); /* Make a new PBlock... */
rc = slapi_add_entry_internal_set_pb(
    pb,
    entry,
    NULL, /* No controls */
    plugin_id,
    SLAPI_OP_FLAG_NEVER_CHAIN /* Never chain this operation. */
);
if (rc != 0) {
    slapi_pblock_destroy(pb);
    return (-1);
}

str = slapi_entry2str(entry, &len); /* Entry as LDIF for the log.
    * Note you have to capture this
    * before the internal add, during
    * which the entry is consumed. */

rc = slapi_add_internal_pb(pb); /* Entry consumed here */
/* ... get status ... */

slapi_pblock_get(pb, SLAPI_PLUGIN_INTOP_RESULT, &rc);
if (rc != LDAP_SUCCESS) {
    slapi_pblock_destroy(pb);
    return (-1);
}

slapi_pblock_destroy(pb);

slapi_log_info_ex(
    SLAPI_LOG_INFO_AREA_PLUGIN,

```

EXAMPLE 8-1 Internal Add Operation (internal.c) *(Continued)*

```
        SLAPI_LOG_INFO_LEVEL_DEFAULT,  
        SLAPI_LOG_NO_MSGID,  
        SLAPI_LOG_NO_CONNID,  
        SLAPI_LOG_NO_OPID,  
        "test_internal in test-internal plug-in",  
        "\nAdded entry:\n%sEntry length: %d\n", str, len  
    );  
  
    return (0);  
}
```

Notice that the internal operation requires a `plugin_id`. As shown, the plug-in ID is a global variable. The plug-in ID is set during plug-in initialization, using this function:

```
slapi_pblock_get(pb, SLAPI_PLUGIN_IDENTITY, &plugin_id);
```

The parameter block, `pb`, is passed to the plug-in initialization function. Refer to the `internal_init()` function in `internal.c` for a sample implementation.

Internal Modify

For internal modify, you first set up an array of `LDAPMod` modifications. The array contains information about the attribute types to modify. The modifications also contain the attribute values. Then, as for internal add, you allocate space for a parameter block. You set up the parameter block with `slapi_modify_internal_set_pb()`. Then you invoke the modify operation with `slapi_modify_internal_pb()`. Finally, you free the memory used.

EXAMPLE 8-2 Internal Modify Operation (internal.c)

This example demonstrates internal modification of a user mail address.

```
#include "slapi-plugin.h"  
  
static Slapi_ComponentId * plugin_id    = NULL;  
  
int  
test_internal()  
{  
    Slapi_Entry    * entry;           /* Entry holder for internal ops */  
    Slapi_PBlock    * pb;             /* PBlock for internal ops      */  
    LDAPMod         mod_attr;         /* Attribute to modify          */  
    LDAPMod         * mods[2];        /* Array of modifications       */  
    char            * mail_vals[] =   /* New mail address             */  
}
```


EXAMPLE 8-2 Internal Modify Operation (internal.c) *(Continued)*

```

        {"quentin@example.com", NULL};
int          rc;                                /* Return code; 0 means success. */

/* Modify Quentin's entry after his email address changes from
 * qcubbins@example.com to quentin@example.com. */
mod_attr.mod_type = "mail";
mod_attr.mod_op    = LDAP_MOD_REPLACE;
mod_attr.mod_values = mail_vals; /* mail: quentin@example.com */
mods[0]            = &mod_attr;
mods[1]            = NULL;

pb = slapi_pblock_new(); /* Set up a PBlock... */
rc = slapi_modify_internal_set_pb(
    pb,
    "uid=qcubbins,ou=people,dc=example,dc=com",
    mods,
    NULL, /* No controls */
    NULL, /* DN rather than unique ID */
    plugin_id,
    SLAPI_OP_FLAG_NEVER_CHAIN /* Never chain this operation. */
);
if (rc != 0) {
    slapi_pblock_destroy(pb);
    return (-1);
}

rc = slapi_modify_internal_pb(pb); /* Unlike internal add,
                                   /* nothing consumed here
                                   /* ... get status ...
slapi_pblock_get(pb, SLAPI_PLUGIN_INTOP_RESULT, &rc);
if (rc != LDAP_SUCCESS) {
    slapi_pblock_destroy(pb);
    return (-1);
}

slapi_pblock_destroy(pb); /* ... clean up the PBlock.

slapi_log_info_ex(
    SLAPI_LOG_INFO_AREA_PLUGIN,
    SLAPI_LOG_INFO_LEVEL_DEFAULT,
    SLAPI_LOG_NO_MSGID,
    SLAPI_LOG_NO_CONNID,
    SLAPI_LOG_NO_OPID,
    "test_internal in test-internal plug-in",
    "\nModified attribute: %s\nNew value: %s\n",
    mod_attr.mod_type, mail_vals[0]

```

EXAMPLE 8-2 Internal Modify Operation (`internal.c`) *(Continued)*

```
    );  
  
    return (0);  
}
```

Notice that the data in `mod_attr` and `mail_vals` is still available for use after the modification. Unlike `internal add`, `internal modify` does not consume the data that you set in the parameter block.

Internal Rename and Move (Modify DN)

This section describes how to develop a plug-in to rename an entry, or to move an entry.

An internal modify DN operation is performed in these stages:

1. Allocate space for a parameter block.
2. Set up the operation by using `slapi_rename_internal_set_pb()`.
3. Invoke the operation by using `slapi_modrdn_internal_pb()`.

The first stage of the operation is `rename`. The second stage of the operation is `modrdn`.

4. Free the parameter block.

EXAMPLE 8-3 Internal Rename or Move Operation (`internal.c`)

This example demonstrates an internal modify DN operation. Internal modify DN does not consume the data that you set in the parameter block.

```
#include "slapi-plugin.h"  
  
static Slapi_ComponentId * plugin_id    = NULL;  
  
int  
test_internal()  
{  
    Slapi_PBlock * pb;           /* PBlock for internal ops      */  
    int          rc;           /* Return code; 0 means success. */  
  
    pb = slapi_pblock_new();      /* Set up a PBlock again...    */  
    rc = slapi_rename_internal_set_pb(  
        pb,  
        "uid=qcubbins,ou=people,dc=example,dc=com", /*Specify target entry*/  
        "uid=fcubbins",                               /*Specify new RDN      */  
        "ou=people,dc=example,dc=com",                 /*Specify new superior*/  
    );  
}
```

EXAMPLE 8-3 Internal Rename or Move Operation (internal.c) *(Continued)*

```

        /* The new superior is the same as the old superior.          */
        1,                      /* Delete old RDN              */
        NULL,                   /* No controls                */
        NULL,                   /* DN rather than unique ID   */
        plugin_id,
        SLAPI_OP_FLAG_NEVER_CHAIN /* Never chain this operation. */
    );

    if (rc != LDAP_SUCCESS) {
        slapi_pblock_destroy(pb);
        return (-1);
    }

    rc = slapi_modrdn_internal_pb(pb); /* Like internal modify,      */
                                     /* nothing consumed here.    */
                                     /* ... get status ...        */
    slapi_pblock_get(pb, SLAPI_PLUGIN_INTOP_RESULT, &rc);
    if (rc != LDAP_SUCCESS) {
        slapi_pblock_destroy(pb);
        return (-1);
    }

    slapi_pblock_destroy(pb); /* ... cleaning up.          */

    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        SLAPI_LOG_NO_MSGID,
        SLAPI_LOG_NO_CONNID,
        SLAPI_LOG_NO_OPID,
        "test_internal in test-internal plug-in",
        "\nNew entry RDN: %s\n", "uid=fcubbins"
    );

    return (0);
}

```

Internal Search

For an internal search, use callbacks to retrieve what the server finds. The callbacks allow you to retrieve the info that would be sent back to a client application were the operation initiated by an external request: LDAP result codes, entries found, and referrals.

You set up the callback data that you want to retrieve. You also write the function that is called back by the server.

EXAMPLE 8-4 Search Callback (internal.c)

This code shows how the example plug-in `internal.c` uses a callback to retrieve an LDIF representation of the first entry that is found. The entry is found during an internal search with `slapi_entry2str()` as the callback function.

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

struct cb_data {
    char * e_str;           /* Data returned from search */
    int   e_len;           /* Entry as LDIF */
                           /* Length of LDIF */
};

int
test_internal_entry_callback(Slapi_Entry * entry, void * callback_data)
{
    struct cb_data * data;
    int             len;

    /* This callback could do something more interesting with the
     * data such as build an array of entries returned by the search.
     * Here, simply log the result. */
    data = (struct cb_data *)callback_data;
    data->e_str = slapi_entry2str(entry, &len);
    data->e_len = len;
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        SLAPI_LOG_NO_MSGID,
        SLAPI_LOG_NO_CONNID,
        SLAPI_LOG_NO_OPID,
        "test_internal_entry_callback in test-internal plug-in",
        "\nFound entry: %sLength: %d\n", data->e_str, data->e_len
    );
    return (-1);           /* Stop at the first entry. */
}                          /* To continue, return 0. */
```

This callback stops the search at the first entry. Your plug-in might have to deal with more than one entry being returned by the search. Therefore, consider how you want to allocate space for your data depending on what your plug-in does.

With the callback data and function implemented, you are ready to process the internal search. First, allocate space for the parameter block and your callback data, and set up the parameter block with `slapi_search_internal_pb_set()`. Next, invoke the search with `slapi_search_internal_pb()`, and also set up the callback with `slapi_search_internal_callback_pb()`. When you are finished, free the space that you have allocated.

EXAMPLE 8-5 Internal Search Operation (internal.c)

```

#include "slapi-plugin.h"

static Slapi_ComponentId * plugin_id    = NULL;

int
test_internal()
{
    Slapi_PBlock * pb;                /* PBlock for internal ops */
    char * srch_attrs[] =             /* Attr. to get during search */
        {LDAP_ALL_USER_ATTRS, NULL};
    struct cb_data callback_data;      /* Data returned from search */
    int rc;                           /* Return code; 0 means success. */

    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        SLAPI_LOG_NO_MSGID,
        SLAPI_LOG_NO_CONNID,
        SLAPI_LOG_NO_OPID,
        "test_internal in test-internal plug-in",
        "\nSearching with base DN %s, filter %s...\n",
        "dc=example,dc=com", "(uid=fcubbins)"
    );

    pb = slapi_pblock_new();           /* Set up a PBlock... */
    rc = slapi_search_internal_set_pb(
        pb,
        "dc=example,dc=com",           /* Base DN for search */
        LDAP_SCOPE_SUBTREE,            /* Scope */
        "(uid=fcubbins)",              /* Filter */
        srch_attrs,                   /* Set to get all user attrs. */
        0,                             /* Return attrs. and values */
        NULL,                          /* No controls */
        NULL,                          /* DN rather than unique ID */
        plugin_id,
        SLAPI_OP_FLAG_NEVER_CHAIN      /* Never chain this operation. */
    );

    if (rc != LDAP_SUCCESS) {
        slapi_pblock_destroy(pb);
        return (-1);
    }

    /* Internal search puts results into the PBlock, but uses callbacks
     * to get at the data as it is turned up by the search. In this case,
     * what you want to get is the entry found by the search. */
}

```

EXAMPLE 8-5 Internal Search Operation (internal.c) *(Continued)*

```
rc = slapi_search_internal_pb(pb);
rc |= slapi_search_internal_callback_pb(
    pb,
    &callback_data,
    NULL, /* No result callback */
    test_internal_entry_callback, /* No referral callback */
    NULL);

/* ... get status ... */
slapi_pblock_get(pb, SLAPI_PLUGIN_INTOP_RESULT, &rc);
if (rc != LDAP_SUCCESS) {
    slapi_pblock_destroy(pb);
    return -1;
}

/* Free the search results when
 * finished with them. */
slapi_free_search_results_internal(pb);
slapi_pblock_destroy(pb); /* ... done cleaning up. */

return (0);
}
```

Here, you allocate and free `callback_data` locally. You can manage memory differently if you pass the data to another plug-in function.

Internal Delete

For internal delete, you allocate space for the parameter block, then set up the parameter block with `slapi_delete_internal_set_pb()`. You invoke the delete with `slapi_delete_internal_pb()`. Finally, you free the parameter block.

Internal delete does not consume the data that you set in the parameter block.

Writing Entry Store and Entry Fetch Plug-Ins

This chapter covers how to write plug-ins that modify how Directory Server operates when writing entries to and reading entries from the directory database.

Note – You can use attribute encryption instead of writing an entry store and entry fetch plug-in. See “Encrypting Attribute Values” in *Sun Java System Directory Server Enterprise Edition 6.1 Administration Guide*.

This chapter covers the following topics:

- [“Calling Entry Store and Entry Fetch Plug-Ins” on page 175](#)
- [“Writing a Plug-In to Encrypt Entries” on page 177](#)

Calling Entry Store and Entry Fetch Plug-Ins

This section describes when entry store and entry fetch plug-ins are called. This section also describes how entries are expected to be handled by the plug-in.

The server calls entry store plug-in functions before writing data to a directory database. It calls entry fetch plug-in functions after reading data from the directory database. Entry store and entry fetch plug-ins may therefore typically be used to encrypt and decrypt directory entries, or to perform auditing work.

Using LDIF String Parameters

Unlike other types of plug-ins, entry store and entry fetch plug-in functions do not take a parameter block as an argument. Instead, the functions take two parameters, the LDIF string representation of an entry and the length of the string. Directory Server uses the modified LDIF string after the plug-in returns successfully. An example prototype for an entry store plug-in function is as follows:

```
int my_entrystore_fn(char ** entry, unsigned int * length);
```

Plug-in functions can manipulate the string as necessary. Entry store and entry fetch plug-ins return zero, 0, on success. When the function returns, Directory Server expects `entry` and `length` to contain the modified versions of the parameters.

Locating the Entry Store and Entry Fetch Examples

The plug-in function examples in this chapter can be found in *install-path/examples/testentry.c*.

The following example shows the entry store scrambling function used in this chapter. This function is called by Directory Server before writing an entry to the database.

EXAMPLE 9-1 Entry Fetch Scrambler (testentry.c)

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

#ifdef _WIN32
typedef unsigned int uint;
__declspec(dllexport)
#endif
int
testentry_scramble(char ** entry, uint * len)
{
    uint i;

    /* Scramble using bitwise exclusive-or on each character. */
    for (i = 0; i < *len - 1; i++) { (*entry)[i] ^= 0xaa; }

    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        SLAPI_LOG_NO_MSGID,
        SLAPI_LOG_NO_CONNID,
        SLAPI_LOG_NO_OPID,
        "testentry_scramble in test-entry plug-in",
        "Entry data scrambled.\n"
    );

    return 0;
}
```

The following example shows the entry fetch unscrambling function used in this chapter. The function is called by the server after reading an entry from the database.

EXAMPLE 9-2 Entry Fetch UnScrambler (testentry.c)

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

#ifdef _WIN32
typedef unsigned int uint;
__declspec(dllexport)
#endif
int
testentry_unscramble(char ** entry, uint * len)
{
    uint i;

    /* Return now if the entry is not scrambled. */
    if (!strcmp(*entry, "dn:", 3)) { return 0; }

    /* Unscramble using bitwise exclusive-or on each character. */
    for (i = 0; i < *len - 1; i++) { (*entry)[i] ^= 0xaa; }

    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        SLAPI_LOG_NO_MSGID,
        SLAPI_LOG_NO_CONNID,
        SLAPI_LOG_NO_OPID,
        "testentry_unscramble in test-entry plug-in",
        "Entry data unscrambled.\n"
    );

    return 0;
}
```

Notice the symmetry between the two functions. The scrambling mask, 0xaa or 10101010 in binary, makes the transformation simple to understand but not secure. A secure encryption mechanism can be significantly more complicated.

Writing a Plug-In to Encrypt Entries

This section shows how to register entry stored entry fetch plug-ins. It demonstrates how to write a plug-in that scrambles directory entries that are written to the directory database. It also demonstrates how to unscramble directory entries that are read from the directory database.



Caution – The examples in this chapter do *not* constitute a secure entry storage scheme.

The following example demonstrates how entry store and entry fetch plug-ins are registered with Directory Server.

Registering Entry Store and Entry Fetch Plug-Ins

The following example demonstrates how entry store and entry fetch plug-ins are registered with Directory Server.

EXAMPLE 9-3 Registering Entry Store and Entry Fetch Plug-Ins (testentry.c)

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

Slapi_PluginDesc entrypdesc = {
    "test-entry",                /* plug-in identifier */
    "Sun Microsystems, Inc.",    /* vendor name */
    "6.0",                      /* plug-in revision number */
    "Sample entry store/fetch plug-in" /* plug-in description */
};

int
testentry_init(Slapi_PBlock *pb)
{
    int rc = 0;                /* 0 means success */
    rc |= slapi_pblock_set(     /* Plug-in API version */
        pb,
        SLAPI_PLUGIN_VERSION,
        SLAPI_PLUGIN_CURRENT_VERSION
    );
    rc |= slapi_pblock_set(     /* Plug-in description */
        pb,
        SLAPI_PLUGIN_DESCRIPTION,
        (void *) &entrypdesc
    );
    rc |= slapi_pblock_set(     /* Entry store function */
        pb,
        SLAPI_PLUGIN_ENTRY_STORE_FUNC,
        (void *) testentry_scramble
    );
    rc |= slapi_pblock_set(     /* Entry fetch function */
        pb,
        SLAPI_PLUGIN_ENTRY_FETCH_FUNC,
        (void *) testentry_unscramble
    );
    return rc;
}
```

Trying the Entry Store and Entry Fetch Examples

You demonstrate the plug-in by showing the difference between scrambled and unscrambled data in the database. You therefore do not enable the plug-in immediately. Instead, you add an entry to a new directory suffix before scrambling. You then observe the results in the database on disk. Next, you remove the entry and enable the scrambling plug-in. Then you add the same entry again. Finally, you observe the results after the entry has been scrambled.

▼ To Set Up an Example Suffix

If you have not done so already, set up a directory instance with a suffix, `dc=example,dc=com`, containing data loaded from a sample LDIF file, *install-path/ds6/ldif/Example.ldif*.

1 Create a new Directory Server instance.

For example:

```
$ dsadm create /local/ds
Choose the Directory Manager password:
Confirm the Directory Manager password:
$
```

2 Start the new Directory Server instance.

For example:

```
$ dsadm start /local/ds
Server started: pid=4705
$
```

3 Create a suffix called `dc=example,dc=com`.

For example, with long lines folded for the printed page:

```
$ dsconf create-suffix -h localhost -p 1389 dc=example,dc=com
Enter "cn=directory manager" password:
Certificate "CN=defaultCert, CN=hostname:1636" presented by the
server is not trusted.
Type "Y" to accept, "y" to accept just once,
"n" to refuse, "d" for more details: Y
$
```

4 Load the sample LDIF.

For example, with long lines folded for the printed page:

```
$ dsconf import -h localhost -p 1389 \
/opt/SUNWdsee/ds6/ldif/Example.ldif dc=example,dc=com
Enter "cn=directory manager" password:
New data will override existing data of the suffix
"dc=example,dc=com".
Initialization will have to be performed on replicated suffixes.
```

```
Do you want to continue [y/n] ? y

## Index buffering enabled with bucket size 16
## Beginning import job...
## Processing file "/opt/SUNWdsee/ds6/ldif/Example.ldif"
## Finished scanning file "/opt/SUNWdsee/ds6/ldif/Example.ldif" (160 entries)
## Workers finished; cleaning up...
## Workers cleaned up.
## Cleaning up producer thread...
## Indexing complete.
## Starting numsubordinates attribute generation.
  This may take a while, please wait for further activity reports.
## Numsubordinates attribute generation complete. Flushing caches...
## Closing files...
## Import complete. Processed 160 entries in 5 seconds.
  (32.00 entries/sec)

Task completed (slapd exit code: 0).
$
```

See Also You can use Directory Service Control Center to perform this task. For more information, see the Directory Service Control Center online help.

Looking for Strings in the Database Before Scrambling

Here, you add an entry for Quentin Cubbins to the example suffix before registering the entry store and fetch plug-in with Directory Server. You see that Quentin's mail address is visible in the database that holds mail address attribute values. Quentin's entry, `quentin.ldif`, appears as shown in the following example.

EXAMPLE 9-4 LDIF Representation of an Entry

```
dn: uid=qcubbins,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: qcubbins
givenName: Quentin
sn: Cubbins
cn: Quentin Cubbins
mail: qcubbins@example.com
userPassword: qcubbins
secretary: uid=bjensen,ou=People,dc=example,dc=com
```

Add Quentin's entry to the directory. For example, if the entry is in `quentin.ldif`, add the following:

```
$ ldapmodify -a -h localhost -p 1389 -f quentin.ldif \
-D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
```

Now look for strings in the directory database file for the mail attribute values.

EXAMPLE 9-5 Attribute Values in a Database File Before Scrambling

```
$ cd instance-path/db/example/
$ strings example_mail.db3 | grep example.com
=qcubbins@example.com
=agodiva@example.com
=hfuddnud@example.com
=pblinn@example.com
=scooper@example.com
=bcubbins@example.com
=yyorgens@example.com
```

Notice that Quentin's mail address is clearly visible if a user gains access to the database files. If the value was a credit card number, security would have been an issue.

Looking for Strings in the Database After Scrambling

Here, you add an entry for Quentin Cubbins to the example suffix after registering the entry store and fetch plug-in with Directory Server. You see that Quentin's mail address is no longer visible in the database that holds mail address attribute values.

Before loading the plug-in, delete Quentin's entry:

```
$ ldapdelete -D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
uid=qcubbins,ou=People,dc=example,dc=com
```

Next, configure Directory Server to load the plug-in as shown in the comments at the beginning of `testentry.c`, and then restart the server.

With the entry store-fetch plug-in active, add Quentin's entry back into the directory:

```
$ ldapmodify -a -h localhost -p 1389 -f quentin.ldif \
-D uid=kvaughan,ou=people,dc=example,dc=com -w bribery
```

Now search again for strings in the directory database file for the mail attribute values.

EXAMPLE 9-6 Attribute Values in a Database File After Scrambling

```
$ cd instance-path/db/example/
$ strings example_mail.db3 | grep example.com
=agodiva@example.com
=hfuddnud@example.com
```

EXAMPLE 9-6 Attribute Values in a Database File After Scrambling (Continued)

```
=pblinn@example.com  
=scooper@example.com  
=bcubbins@example.com  
=yyorgens@example.com
```

Notice that Quentin's mail address value is now not visible in the directory database. Directory users who have appropriate access rights, anonymous in this simple example case, can still view the attribute during a search. The attribute and its value are *emphasized* in the following example.

EXAMPLE 9-7 Unscrambled Search Results

```
$ ldapsearch -h localhost -p 1389 -b dc=example,dc=com uid=qcubbins  
dn: uid=qcubbins,ou=People,dc=example,dc=com  
objectClass: top  
objectClass: person  
objectClass: organizationalPerson  
objectClass: inetOrgPerson  
uid: qcubbins  
givenName: Quentin  
sn: Cubbins  
cn: Quentin Cubbins  
mail: qcubbins@example.com  
secretary: uid=bcubbins,ou=People,dc=example,dc=com
```

In this way, you see that entry store and entry fetch plug-ins affect only the way entries are stored, not the directory front end.

Writing Extended Operation Plug-Ins

This chapter describes how to write plug-ins to take advantage of LDAP v3 extended operations. Extended operations are defined in [RFC 4511](http://www.ietf.org/rfc/rfc4511.txt) (<http://www.ietf.org/rfc/rfc4511.txt>). The RFC defines “additional operations to be defined for services not available elsewhere in this protocol, for instance digitally signed operations and results.”

This chapter covers the following topics:

- “Calling Extended Operation Plug-Ins” on page 183
- “Implementing an Extended Operation Plug-In” on page 184

Calling Extended Operation Plug-Ins

This section describes how extended operations are handled by Directory Server. This section also describes what requirements are placed on extended operation plug-ins.

Directory Server identifies extended operations by object identifiers (OIDs). Clients request an operation by sending an extended operation request, specifying the following:

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

Upon receiving an extended operation request, Directory Server calls the plug-in registered to handle the OID that is sent in the request. The plug-in function that handles the request obtains the OID and operation-specific data. The plug-in processes the info, then sends a response to the client that contains an OID as well as additional data that pertains to the extended operation.

When implementing extended operation support, you need to determine which OID to use. The Internet Assigned Numbers Authority helps with registration of official OIDs. If you select your own OIDs, you must avoid OIDs that conflict with other OIDs. In particular, do not use OIDs that conflict with OIDs defined by the LDAP schema used by Directory Server.

Directory Server determines which extended operation plug-ins handle which extended operation OIDs at startup. Directory Server then calls the initialization function for each plug-in. Extended operation plug-ins register the extended operation OIDs the plug-ins handle as part of their initialization function, as described in [“Initializing the Extended Operation Plug-In” on page 185](#).

The configuration entry for a plug-in can include parameters that Directory Server passes to the plug-in initialization function at startup. These parameters are described in [“Retrieving Arguments Passed to Plug-Ins” on page 94](#). The parameters allow for the possibility that OIDs are determined after the plug-in functionality is written. Such OIDs can be passed as configuration entry arguments.

Implementing an Extended Operation Plug-In

This section demonstrates how to implement a basic extended operation plug-in and client application to trigger the extended operation.

The client sends an extended operation request to the server by using the OID that identifies the extended operation, in this case 1.2.3.4. The client also sends a value that holds a string. The example plug-in responds to an extended operation request by sending the client an OID. The plug-in also sends a modified version of the string that the client sent with the request.

Locating the Extended Operation Examples

The rest of this chapter refers to the plug-in code in *install-path/examples/testextendedop.c*, and the client code in *install-path/examples/clients/reqextop.c*.

Example Extended Operation Plug-In

This section explains how the extended operation plug-in works.

Registering the Extended Operation Plug-In

Before using the plug-in function as described here, build the plug-in. Also, configure Directory Server to load the plug-in.

Notice that OID 1.2.3.4 is passed as an argument through the configuration entry. The configuration entry could specify more than one `nsslapd-pluginarg` attribute if the plug-in supported multiple extended operations, each identified by a distinct OID, for example.

▼ To Register the Plug-In

If you have not already done so, build the example plug-in library and activate both plug-in informational logging and the example plug-in.

1 Build the plug-in.

Hint Use `install-path/examples/Makefile` or `install-path/examples/Makefile64`.

2 Configure Directory Server to log plug-in informational messages and load the plug-in.

Hint Use the commands specified in the comments at the outset of the plug-in source file.

3 Restart Directory Server.

```
$ dsadm restart instance-path
```

Initializing the Extended Operation Plug-In

As for other plug-in types, extended operation plug-ins include an initialization function that registers other functions in the plug-in with Directory Server. For extended operation plug-ins, this initialization function also registers the OIDs handled by the plug-in. The function registers OIDs by setting `SLAPI_PLUGIN_EXT_OP_OIDLIST` in the parameter block Directory Server, which the function passes to the initialization function.

EXAMPLE 10-1 Registering Plug-In Functions and OIDs (testextendedop.c)

This example demonstrates how the OID list is built and registered.

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

Slapi_PluginDesc expdesc = {
    "test-extendedop",          /* plug-in identifier */
    "Sun Microsystems, Inc.",    /* vendor name */
    "6.0",                      /* plug-in revision number */
}
```

EXAMPLE 10-1 Registering Plug-In Functions and OIDs (testextendedop.c) *(Continued)*

```

    "Sample extended operation plug-in"/* plug-in description    */
};

#ifdef _WIN32
__declspec(dllexport)
#endif
int
testexop_init(Slapi_PBlock * pb)
{
    char ** argv;                /* Args from configuration */
    int     argc;                /* entry for plug-in.      */
    char ** oid_list;            /* OIDs supported          */
    int     rc = 0;              /* 0 means success         */
    int     i;

    /* Get the arguments from the configuration entry.          */
    rc |= slapi_pblock_get(pb, SLAPI_PLUGIN_ARGV, &argv);
    rc |= slapi_pblock_get(pb, SLAPI_PLUGIN_ARGC, &argc);
    if (rc != 0) {
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            SLAPI_LOG_NO_MSGID,
            SLAPI_LOG_NO_CONNID,
            SLAPI_LOG_NO_OPID,
            "testexop_init in test-extendedop plug-in",
            "Could not get plug-in arguments.\n"
        );
        return (rc);
    }

    /* Extended operation plug-ins may handle a range of OIDs. */
    oid_list = (char **)slapi_ch_malloc((argc + 1) * sizeof(char *));
    for (i = 0; i < argc; ++i) {
        oid_list[i] = slapi_ch_strdup(argv[i]);
        slapi_log_info_ex(
            SLAPI_LOG_INFO_AREA_PLUGIN,
            SLAPI_LOG_INFO_LEVEL_DEFAULT,
            SLAPI_LOG_NO_MSGID,
            SLAPI_LOG_NO_CONNID,
            SLAPI_LOG_NO_OPID,
            "testexop_init in test-extendedop plug-in",
            "Registering plug-in for extended operation %s.\n",
            oid_list[i]
        );
    }
}

```

EXAMPLE 10-1 Registering Plug-In Functions and OIDs (testextendedop.c) *(Continued)*

```

oid_list[argc] = NULL;

rc |= slapi_pblock_set(          /* Plug-in API version    */
    pb,
    SLAPI_PLUGIN_VERSION,
    SLAPI_PLUGIN_CURRENT_VERSION
);
rc |= slapi_pblock_set(          /* Plug-in description */
    pb,
    SLAPI_PLUGIN_DESCRIPTION,
    (void *) &expdesc
);
rc |= slapi_pblock_set(          /* Extended op. handler */
    pb,
    SLAPI_PLUGIN_EXT_OP_FN,
    (void *) test_extendedop
);
rc |= slapi_pblock_set(          /* List of OIDs handled */
    pb,
    SLAPI_PLUGIN_EXT_OP_OIDLIST,
    oid_list
);
return (rc);
}

```

Notice that you extract OIDs from the arguments passed by Directory Server. Directory Server passes the arguments from the configuration entry to the parameter block, by using `slapi_ch_strdup()` on each `argv[]` element. The OID list is then built by allocating space for the array by using `slapi_ch_malloc()` and placing the OIDs in each `oid_list[]` element. You then register the plug-in OID list by using `SLAPI_PLUGIN_EXT_OP_OIDLIST`. You register the extended operation handler function, `test_extendedop()`, using `SLAPI_PLUGIN_EXT_OP_FN` as shown.

Refer to [Part II](#) for details about parameter block arguments and plug-in API functions.

Handling the Extended Operation

The plug-in function `test_extendedop()` gets the OID and value for the operation from the client request. The function then sends the client a response, as shown in [“Developing the Extended Operation Client” on page 188](#).

Notice how the function obtains the OID and value from the request by using `SLAPI_EXT_OP_REQ_OID` and `SLAPI_EXT_OP_REQ_VALUE`. The function then uses `slapi_ch_malloc()` to construct a string to return to the client through the pointer to a `berval` structure, `result_bval`. A different extended operation plug-in might do something entirely different at this point.

Also notice that the function sends a different OID back to the client than the OID in the client request. The OID that is sent back can be used to indicate a particular result to the client, for example. The function uses `slapi_send_ldap_result()` to indicate success as well as to send the OID and value to the client. The function then frees the memory that was allocated. Finally, the function returns `SLAPI_PLUGIN_EXTENDED_SENT_RESULT` to indicate to Directory Server that processing of the plug-in function is complete.

If the function had not sent a result code to the client, it would return an LDAP result code to Directory Server. Directory Server would then send the result code to the client.

If the function cannot handle the extended operation with the specified OID, the function returns `SLAPI_PLUGIN_EXTENDED_NOT_HANDLED`. Then Directory Server sends an `LDAP_PROTOCOL_ERROR` result code to the client.

Developing the Extended Operation Client

To test the plug-in, you need a client that requests an extended operation with OID 1.2.3.4. The example discussed here, `reqextop.c`, is delivered with the product.

The client sets up a short string to send to Directory Server in the extended operation request. The client then gets an LDAP connection that supports LDAP version 3, and binds to Directory Server. The client then sends an extended operation request and displays the result on `STDOUT`.

EXAMPLE 10-2 Client Requesting an Extended Operation (`clients/reqextop.c`)

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

/* Global variables for client connection information.
 * You may set these here or on the command line. */
static char * host      = "localhost"; /* Server hostname */
static int   port       = 389;         /* Server port */
static char * bind_DN   = "cn=Directory Manager"; /* DN to bind as */
static char * bind_pwd  = "23skidoo"; /* Password for bind DN */

/* Check for connection info as command line arguments. */
int get_user_args(int argc, char ** argv);

int
main(int argc, char ** argv)
{
    /* OID of the extended operation that you are requesting */
    const char * oidrequest = "1.2.3.4"; /* Ext op OID */
    char * oidresult; /* OID in reply from server */
    struct berval valrequest; /* Request sent */
    struct berval * valresult; /* Reply received */
```

EXAMPLE 10-2 Client Requesting an Extended Operation (clients/reqextop.c) *(Continued)*

```

LDAP      * ld;                /* Handle to connection */
int        version;            /* LDAP version */

/* Use default connection arguments unless all four are
 * provided as arguments on the command line. */
if (get_user_args(argc, argv) != 0) return 1; /* Usage error */

/* Set up the value that you want to pass to the server */
printf("Setting up value to pass to server...\n");
valrequest.bv_val = "My Value";
valrequest.bv_len = strlen("My Value");

/* Get a handle to an LDAP connection */
printf("Getting the handle to the LDAP connection...\n");
if ((ld = ldap_init(host, port)) == NULL) {
    perror("ldap_init");
    ldap_unbind(ld);
    return 1;
}

/* Set the LDAP protocol version supported by the client
   to 3. (By default, this is set to 2. Extended operations
   are part of version 3 of the LDAP protocol.) */
printf("Resetting version %d to 3.0...\n", version);
version = LDAP_VERSION3;
ldap_set_option(ld, LDAP_OPT_PROTOCOL_VERSION, &version);

/* Authenticate to the directory as the Directory Manager */
printf("Binding to the directory...\n");
if (ldap_simple_bind_s(ld, bind_DN, bind_pwd) != LDAP_SUCCESS) {
    ldap_perror(ld, "ldap_simple_bind_s");
    ldap_unbind(ld);
    return 1;
}

/* Initiate the extended operation */
printf("Initiating the extended operation...\n");
if (ldap_extended_operation_s(
    ld,
    oidrequest,
    &valrequest,
    NULL,
    NULL,
    &oidresult,
    &valresult
) != LDAP_SUCCESS) {

```

EXAMPLE 10-2 Client Requesting an Extended Operation (clients/reqextop.c) *(Continued)*

```

        ldap_perror(ld, "ldap_extended_operation_s failed: ");
        ldap_unbind(ld);
        return 1;
    }

    /* Get OID and value from result returned by server. */
    printf("Operation successful.\n");
    printf("\tReturned OID: %s\n", oidresult);
    printf("\tReturned value: %s\n", valresult->bv_val);

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

```

In this example, the client identifies the request by OID. Notice also that the client resets the protocol version to LDAP_VERSION3 to ensure extended operation support in the protocol. The value that the client sends with the request, `val request`, points to a `berval` structure. Also notice that the calls used for the bind and extended operation are synchronous. Asynchronous versions are also available.

Trying the Extended Operation Example

1. After activating the plug-in in the server, compile the client code `reqextop.c`.
2. Add an entry under `cn=features` in the configuration to let users access the extended operation.

```

$ cat myextop.ldif
dn: oid=1.2.3.4,cn=features,cn=config
objectClass: top
objectClass: directoryServerFeature
oid: 1.2.3.4
cn: Fake extended operation
aci: (targetattr != "aci")(version 3.0;acl "Fake extended operation"; allow
  ( read, search, compare, proxy ) userdn = "ldap:///all";)

$ ldapmodify -h localhost -p 389 -a -D cn=directory\ manager -w - -f myextop.ldif
Enter bind password:
adding new entry uid=qcubbins,ou=People,dc=example,dc=com

$ dsadm restart /local/ds
Waiting for server to stop...
Server stopped

```

```
Server started: pid=5658
$
```

3. Run the client to send the extended operation request and display the result.

```
$ ./reqextop -w password
Using the following connection info:
    host:    localhost
    port:    389
    bind DN: cn=Directory Manager
    pwd:     password
Setting up value to pass to server...
Getting the handle to the LDAP connection...
Resetting version 2 to 3.0...
Binding to the directory...
Initiating the extended operation...
Operation successful.
    Returned OID: 5.6.7.8
    Returned value: Value from client: My Value
```

4. On the Directory Server side, turn on logging.

Messages similar to the following in the errors log show that the plug-in handled the client request:

```
[22/May/2112:08:54:15 +0200] - INFORMATION -
test_extendedop in test-extendedop plug-in - conn=0 op=1 msgId=2 -
Request with OID: 1.2.3.4 Value from client: My Value
[22/May/2112:08:54:15 +0200] - INFORMATION -
test_extendedop in test-extendedop plug-in - conn=0 op=1 msgId=2 -
OID sent to client: 5.6.7.8
Value sent to client:
Value from client: My Value
```

You have thus demonstrated that the example extended operation plug-in handles requests for the extended operation with OID 1.2.3.4.

Writing Matching Rule Plug-Ins

This chapter covers plug-ins that enable Directory Server to handle custom matching rules. The server requires such matching rule plug-ins to support LDAP v3 extensible matching, such as “sounds like” searches.

Code excerpts in this chapter demonstrate a conceptually simple case exact matching scheme. Here, exact matching is a byte comparison between `DirectoryString` attribute values. Despite the straightforward concept, the plug-in code runs to many lines. The plug-in must provide several functions as well as wrapper functions to enable indexing, searching, and sorting.

Note – You must understand how Directory Server uses matching rule plug-ins before trying to implement your own plug-in. Read this chapter, then read the sample plug-in code. Avoid creating a new matching rule plug-in from scratch.

This chapter covers the following topics:

- “How Matching Rule Plug-Ins Work” on page 193
- “Example Matching Rule Plug-In” on page 196
- “Handling Extensible Match Filters” on page 199
- “Indexing Entries According to a Matching Rule” on page 211
- “Enabling Sorting According to a Matching Rule” on page 219
- “Handling an Unknown Matching Rule” on page 219

How Matching Rule Plug-Ins Work

This section summarizes what matching rule plug-ins are and how Directory Server handles them.

What a Matching Rule Is

A *matching rule* defines a specific way to compare attribute values that have a given syntax. In other words, a matching rule defines how potentially matching attributes are compared.

Every matching rule is identified by a unique object identifier (OID) string. A client application that requests a search can specify the matching rule OID in the search filter. The OID indicates to Directory Server how to check for a match of two attribute values.

In practice, a client application might want only entries with attribute values that match the value provided exactly. The sample plug-in demonstrates how you might implement a solution for that case. Another client might want to sort entries according to the rules for a given locale. Directory Server actually uses a matching rule plug-in to handle locale—specific matching.

Chapter 11, “Directory Server Internationalization Support,” in *Sun Java System Directory Server Enterprise Edition 6.1 Reference* includes a list of matching rules. Directory Server supports the rules for internationalized searches. You can also view the list by searching the default schema.

```
$ ldapsearch -h localhost -p 1389 -b cn=schema cn=schema matchingRules
```

Requesting a Matching Rule

To request a custom matching rule on a specific attribute, a client application includes the matching rule OID in the search filter. LDAP v3 calls these search filters *extensible match filters*. An extensible match filter looks like the following:

```
(cn:2.5.13.5:=Quentin)
```

This filter tells the server to search for `Quentin` in the common name (CN) of the entries by using matching rule `2.5.13.5`. The matching rule happens to be a case exact match.

The case exact matching rule plug-in OID `2.5.13.5` enables Directory Server to perform the search correctly. Directory Server calls code in this matching rule plug-in to check for matches during a search. The server also uses the code to generate indexes that accelerate case exact searches. The indexes also help to sort entries found during such searches.

What a Matching Rule Plug-In Does

A matching rule plug-in can provide code to do the following:

- Check for matches during an extensible match search
- Sort results for an extensible match search by using a sort control
- Maintain an index to speed an extensible match search (optional)

To enable these capabilities, the plug-in implements matching and indexing routines that Directory Server calls to handle requests involving the particular matching rule.

The plug-in also implements factory functions that specify which routine to call when handling a particular matching rule. As a result, a plug-in can support multiple matching rules. Yet, plug-ins implementing only one matching rule also require factory function code to wrap indexing and matching routines. A plug-in therefore requires many lines of code and several functions to handle even a minimal matching rule.

The following table shows all the functions that a matching rule plug-in can implement.

TABLE 11-1 Functions Defined in Matching Rule Plug-Ins

Type	Parameter Block Identifier	Required?
Filter factory	SLAPI_PLUGIN_MR_FILTER_CREATE_FN	Required
Filter index, used to check an index for matches	SLAPI_PLUGIN_MR_FILTER_INDEX_FN	Not required
Filter match	SLAPI_PLUGIN_MR_FILTER_MATCH_FN	Required
Filter match reset	SLAPI_PLUGIN_MR_FILTER_RESET_FN	If filter must be reset for reuse
Filter object destructor	SLAPI_PLUGIN_DESTROY_FN	If needed to free memory
Indexer	SLAPI_PLUGIN_MR_INDEX_FN	Not required
Indexer factory	SLAPI_PLUGIN_MR_INDEXER_CREATE_FN	Not required
Indexer object destructor	SLAPI_PLUGIN_DESTROY_FN	If needed to free memory
Plug-in initialization function	Not applicable, but instead specified in configuration settings	Required
Server shutdown (cleanup) function	SLAPI_CLOSE_FN	Not required
Server startup function	SLAPI_START_FN	Not required

Refer to [Part II](#) for details about parameter block identifiers that you can use with matching rule plug-ins.

Example Matching Rule Plug-In

The example code cited in this chapter, from *install-path/examples/matchingrule.c*, mimics functionality installed by default with Directory Server. For this reason, you do not need to build and load the plug-in unless you modify the plug-in to implement your own matching rule. The example uses a different OID from the plug-in that is provided with Directory Server. Therefore, the sample plug-in does not interfere with the existing plug-in if you do choose to load the sample.

Configuring Matching Rule Plug-Ins

Matching rule plug-ins have type `matchingrule`. Directory Server plug-ins can depend on matching rule plug-ins by type. Directory Server cannot load plug-ins unless plug-ins of the type the plug-ins depend on load without error.

Refer to “[Plugging Libraries Into Directory Server](#)” on page 90 for details on loading plug-in configuration entries into Directory Server.

Registering Matching Rule Plug-Ins

Matching rule plug-ins include factory functions. Factory functions provide function pointers to the indexing or filter match routines dynamically. You therefore register only the factory functions as part of the plug-in initialization function.

A plug-in that handles both indexing and matching registers both factory functions and a description, as shown in the following example.

EXAMPLE 11-1 Registering Matching Rule Factory Functions (*matchingrule.c*)

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

static Slapi_PluginDesc plg_desc = {
    "caseExactMatchingRule",      /* Plug-in identifier      */
    "Sun Microsystems, Inc.",     /* Vendor name            */
    "6.0",                       /* Plug-in revision number */
    "Case Exact Matching Rule plug-in" /* Plug-in description    */
};

#ifdef _WIN32
__declspec(dllexport)
#endif
int
plg_init(Slapi_PBlock * pb)
{
```

EXAMPLE 11-1 Registering Matching Rule Factory Functions (matchingrule.c) (Continued)

```

int                                rc;

/* Matching rule factory functions are registered using
 * the parameter block structure. Other functions are then
 * registered dynamically by the factory functions.
 *
 * This means that a single matching rule plug-in may handle
 * a number of different matching rules.                                */
rc = slapi_pblock_set(
    pb,
    SLAPI_PLUGIN_MR_INDEXER_CREATE_FN,
    (void *)plg_indexer_create
);
rc |= slapi_pblock_set(
    pb,
    SLAPI_PLUGIN_MR_FILTER_CREATE_FN,
    (void *)plg_filter_create
);
rc |= slapi_pblock_set(
    pb,
    SLAPI_PLUGIN_DESCRIPTION,
    (void *)&plg_desc
);

/* Register the matching rules themselves. */

return rc;
}

```

Here, `plg_init()` is the plug-in initialization function, `plg_indexer_create()` is the indexer factory, `plg_filter_create()` is the filter factory, and `plg_desc` is the plug-in description structure.

If your plug-in uses private data, set `SLAPI_PLUGIN_PRIVATE` in the parameter block as a pointer to the private data structure.

EXAMPLE 11-2 Registering a Matching Rule (matchingrule.c)

Register matching rules by using `slapi_matchingrule_register()` as shown here rather than using `slapi_pblock_set()` in the initialization function.

```

#include "slapi-plugin.h"

#define PLG_DESC    "Case Exact Matching on Directory String" \
    " [defined in X.520]"

```

EXAMPLE 11-2 Registering a Matching Rule (matchingrule.c) *(Continued)*

```

#define PLG_NAME      "caseExactMatch"
/* This OID is for examples only and is not intended for reuse. The
 * official OID for this matching rule is 2.5.13.5. */
#define PLG_OID       "1.3.6.1.4.1.42.2.27.999.4.1"

#ifdef _WIN32
__declspec(dllexport)
#endif
int
plg_init(Slapi_PBlock * pb)
{
    Slapi_MatchingRuleEntry * mrentry = slapi_matchingrule_new();
    int rc;

    /* Register matching rule factory functions.*/

    /* Matching rules themselves are registered using a
     * Slapi_MatchingRuleEntry structure, not the parameter
     * block structure used when registering plug-in functions.
     *
     * This plug-in registers only one matching rule. Yours
     * may register many matching rules. */
    rc |= slapi_matchingrule_set(
        mrentry,
        SLAPI_MATCHINGRULE_DESC,
        (void *)slapi_ch_strdup(PLG_DESC)
    );
    rc |= slapi_matchingrule_set(
        mrentry,
        SLAPI_MATCHINGRULE_NAME,
        (void *)slapi_ch_strdup(PLG_NAME)
    );
    rc |= slapi_matchingrule_set(
        mrentry,
        SLAPI_MATCHINGRULE_OID,
        (void *)slapi_ch_strdup(PLG_OID)
    );
    rc |= slapi_matchingrule_set(
        mrentry,
        SLAPI_MATCHINGRULE_SYNTAX, /* Here you use DirectoryString.*/
        (void *)slapi_ch_strdup("1.3.6.1.4.1.1466.115.121.1.15")
    );
    rc |= slapi_matchingrule_register(mrentry);
    slapi_matchingrule_free(&mrentry, 1);

    return rc;
}

```

EXAMPLE 11-2 Registering a Matching Rule (`matchingrule.c`) (Continued)

- `PLG_DESC` is a string that describes the matching rule.
- `PLG_NAME` is a string identifier for the matching rule.
- `PLG_OID` is the object identifier for the matching rule.

The sample plug-in `#defines` every item. If the plug-in implements several matching rules, the initialization function must register each item.

Notice that plug-ins must register factory functions separately, using `slapi_pblock_set()`.

Handling Extensible Match Filters

This section explains and demonstrates how to handle extensible match filters corresponding to a matching rule. All matching rule plug-ins must enable filter matching.

How Directory Server Handles Extensible Match Searches

The following process describes how Directory Server handles an extensible match search.

1. When Directory Server receives a search request containing an extensible match filter, the server calls the *filter factory function* in the plug-in handling the corresponding matching rule. The server passes the filter factory a parameter block containing the extensible match filter information.
2. The filter factory function builds a filter object. The function then sets pointers to the object as well as the appropriate *filter matching* and *filter index functions*.
3. The server attempts to look up a set of results in an index rather than searching the entire directory.

In order to read an index, the server calls the filter index function, which helps the server locate the appropriate indexer function. Directory Server only considers all entries in the directory, which can slow the search considerably if one of the following is the case:

- No existing index applies to the search
- The plug-in specifies no indexer function
- The plug-in generates no keys for the values specified

Directory Server calls the indexer function to generate the keys. Refer to “[Indexer Function](#)” on page 213 for details about that function.

4. Directory Server builds a list of candidate entries.
5. Directory Server verifies that the entry is in the scope of the search before returning the entry to the client as a search result.

6. Directory Server frees memory that was allocated for the operation.

The following figure shows how Directory Server processes the search request.

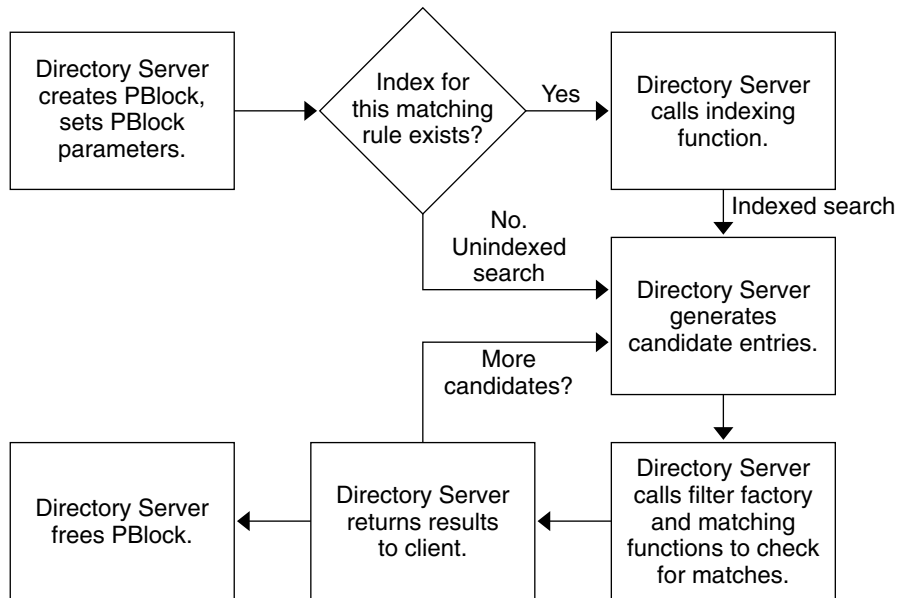


FIGURE 11-1 Directory Server Performing an Extensible Match Search

The main filter functions are as follows:

- [Filter matching function](#)
- [Filter index function](#)
- [Filter factory function](#)
- [Filter object destructor function](#)

Filter Matching Function

The filter matching function takes pointers to the filter object, the entry, and the first attribute to check for a match.

The following figure shows how Directory Server uses the filter matching function.

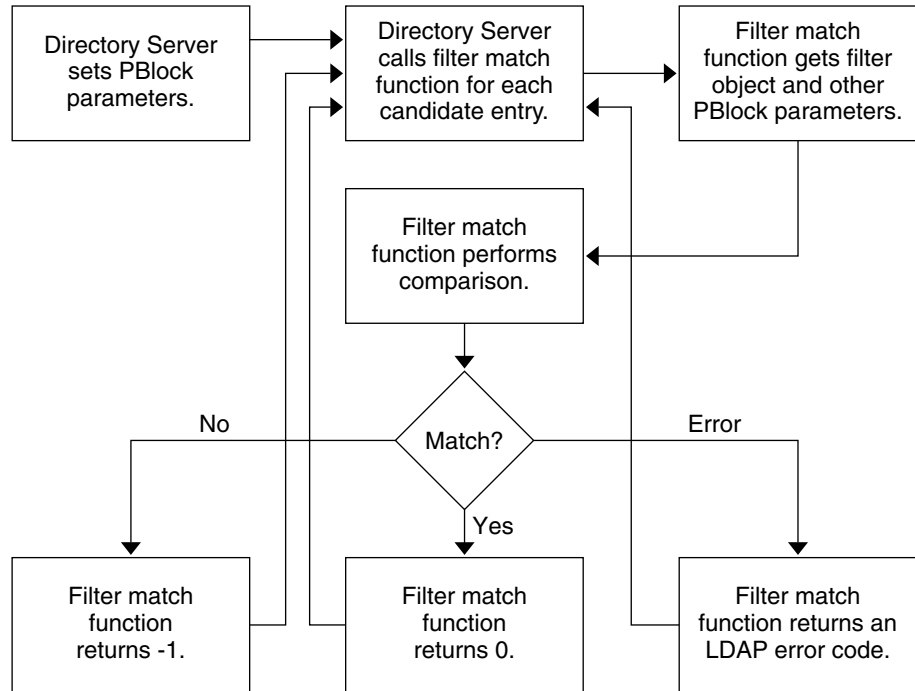


FIGURE 11-2 Filter Match Function Context

Notice that the filter matching function returns 0 (match) -1 (no match) or an LDAP error code for each entry processed.

EXAMPLE 11-3 Filter Match Function (matchingrule.c)

This example shows the filter match function for the case exact matching rule where a match occurs when the two berval structures match.

```

#include "slapi-plugin.h"

typedef struct plg_filter_t
{
    char      * f_type;
    int       f_op;

    struct berval ** f_values;
} plg_filter_t;

/* For manipulating filter obj. */
/* Attribute type to match */
/* Type of comparison
 * (<, <=, ==, >=, >, substr)
 * for the filter.
 */
/* Array of values to match */

/* Check for a match against the filter.
 * Returns: 0 filter matched
 *          -1 filter did not match

```

EXAMPLE 11-3 Filter Match Function (matchingrule.c) (Continued)

```

    *          > 0 an LDAP Error code                                */
static int
plg_filter_match(
    void      * obj,          /* Matching rule object      */
    Slapi_Entry * entry,      /* Entry to match            */
    Slapi_Attr * attr         /* Attributes to match       */
)
{
    plg_filter_t * fobj = (plg_filter_t *)obj;
    struct berval * mrVal = NULL; /* Values handled as bervals... */
    Slapi_Value * sval;          /* ...and as Slapi_Value structs*/
    int          rc = -1;        /* No match                   */

    if (fobj && fobj->f_type && fobj->f_values && fobj->f_values[0]) {

        mrVal = fobj->f_values[0];

        /* Iterate through the attributes, matching subtypes.          */
        for (; attr != NULL; slapi_entry_next_attr(entry, attr, &attr)) {

            char * type = NULL; /* Attribute type to check      */

            if ((slapi_attr_get_type(attr, &type) == 0) &&
                (type != NULL) &&
                (slapi_attr_type_cmp(fobj->f_type, type, 2) == 0)
            ) { /* slapi_attr_type_cmp(type1, type2, ==>2<==)
                * matches subtypes, too. Refer to the reference
                * documentation for details.                                */

                /* Type and subtype match, so iterate through the
                 * values of the attribute.                                */
                int hint = slapi_attr_first_value(attr, &sval);
                while (hint != -1) {
                    const struct berval * val =
                        slapi_value_get_berval(sval);

                    /* The case exact matching rule
                     * compares the two bervals.
                     *
                     * Your matching rule may do
                     * lots of different checks here.                */
                    rc = slapi_berval_cmp(val, mrVal);
                    if (rc == 0) { /* Successful match                */
                        /* If you have allocated memory for a custom
                         * matching rule, do not forget to release it. */
                        return 0;
                    }
                }
            }
        }
    }
}

```

EXAMPLE 11-3 Filter Match Function (matchingrule.c) (Continued)

```

        }

        hint = slapi_attr_next_value(attr, hint, &sval);
    }
}
}
}
return rc;
}

```

Notice that checking for a case exact match involves only `slapi_berval_cmp()`, which performs a byte by byte comparison. Your plug-in might do something more complex for the comparison.

Subtype Matches

Notice in the code for iterating through the attributes that `slapi_attr_type_cmp()` takes 2, the value assigned to `SLAPI_TYPE_CMP_SUBTYPE`, as its third argument. The argument forces a comparison of attribute subtypes, such as the locale of an attribute, in addition to attribute types. Refer to [Part II](#) for details on plug-in API functions and their arguments.

Thread Safety and Filter Matching Functions

Directory Server never calls a filter matching function for the same filter object concurrently. Directory Server can, however, call the function concurrently for different filter objects. If you use global variables, ensure that your filter matching function handles such variables safely.

Filter Index Function

The filter index function takes a parameter block from Directory Server. The function also sets pointers in the parameter block, enabling the server to read the index.

EXAMPLE 11-4 Filter Index Function (matchingrule.c)

This example shows the filter index function for the case exact matching rule where the keys are the same as the values.

```

#include "slapi-plugin.h"

#define PLG_OID        "1.3.6.1.4.1.42.2.27.999.4.1"
#define SUBSYS         "CaseExactMatching Plugin"

typedef struct plg_filter_t          /* For manipulating filter obj. */
{  char          *  f_type;          /* Attribute type to match      */

```

EXAMPLE 11-4 Filter Index Function (matchingrule.c) (Continued)

```
int          f_op;          /* Type of comparison
                             * (<, <=, ==, >=, >, substr)
                             * for the filter.          */
struct berval ** f_values;  /* Array of values to match */
} plg_filter_t;

static int
plg_filter_index(Slapi_PBlock * pb)
{
    int          rc          = LDAP_UNAVAILABLE_CRITICAL_EXTENSION;
    void         * obj       = NULL; /* Server lets the plug-in */
    plg_filter_t * fobj;      /* handle the object type.      */
    int          query_op    = SLAPI_OP_EQUAL; /* Only exact matches */

    if (!slapi_pblock_get(pb, SLAPI_PLUGIN_OBJECT, &obj)) {

        fobj = (plg_filter_t *)obj;

        /* Case exact match requires no modifications to
         * the at this point. Your plug-in may however
         * modify the object, then set it again in the
         * parameter block.          */
        rc = slapi_pblock_set( /* This is for completeness. */
            pb,                /* Your plug-in may modify */
            SLAPI_PLUGIN_OBJECT, /* the object if necessary. */
            fobj
        );
        rc |= slapi_pblock_set( /* Set attr type to match. */
            pb,
            SLAPI_PLUGIN_MR_TYPE,
            fobj->f_type
        );
        rc |= slapi_pblock_set( /* Set fcn to obtain keys. */
            pb,
            SLAPI_PLUGIN_MR_INDEX_FN,
            (void*)plg_index_entry
        );
        rc |= slapi_pblock_set( /* Set values to obtain keys. */
            pb,
            SLAPI_PLUGIN_MR_VALUES,
            fobj->f_values
        );
        rc |= slapi_pblock_set( /* This is for completeness. */
            pb,                /* Your plug-in may set a */
            SLAPI_PLUGIN_MR_OID, /* different MR OID than the */
            PLG_OID             /* one in the parameter block */
        );
    }
}
```

EXAMPLE 11-4 Filter Index Function (matchingrule.c) (Continued)

```

    );
    rc |= slapi_pblock_set(          /* <, <=, ==, >=, >, substr      */
        pb,                        /* In this case, ==          */
        SLAPI_PLUGIN_MR_QUERY_OPERATOR,
        &query_op
    );
}

return rc;
}

```

This code uses the variables and macros that follow:

- `fobj->ftype` indicates the attribute type that is specified in the filter.
- `plg_index_entry()` indicates the indexer function for generating keys from values.
- `fobj->values` points to the `berval` array of attribute values.
- `PLG_OID` is the object identifier of the matching rule.
- `query_op` indicates the query operator from the filter.

Input Parameters for Filter Index Functions

A filter index function can get a pointer to the filter object from `SLAPI_PLUGIN_OBJECT` in the parameter block.

Output Parameters for Filter Index Functions

A filter index function should set values for at least the following in the parameter block before returning control to Directory Server:

- `SLAPI_PLUGIN_MR_INDEX_FN`, the pointer to the indexer for generating the keys
- `SLAPI_PLUGIN_MR_OID`
- `SLAPI_PLUGIN_MR_QUERY_OPERATOR`
- `SLAPI_PLUGIN_MR_TYPE`
- `SLAPI_PLUGIN_MR_VALUES`
- `SLAPI_PLUGIN_OBJECT`, if modified by your function

Refer to [Chapter 18](#) for details.

Thread Safety and Filter Index Functions

Directory Server never calls a filter index function for the same filter object concurrently. Directory Server can, however, call the function concurrently for different filter objects. If you use global variables, ensure that your function handles such variables safely.

Filter Factory Function

The filter factory function takes a parameter block from Directory Server. The function then sets parameters such that Directory Server can build a list of candidate entries, candidates the server checks for matches.

The following figure shows how the filter factory function operates.

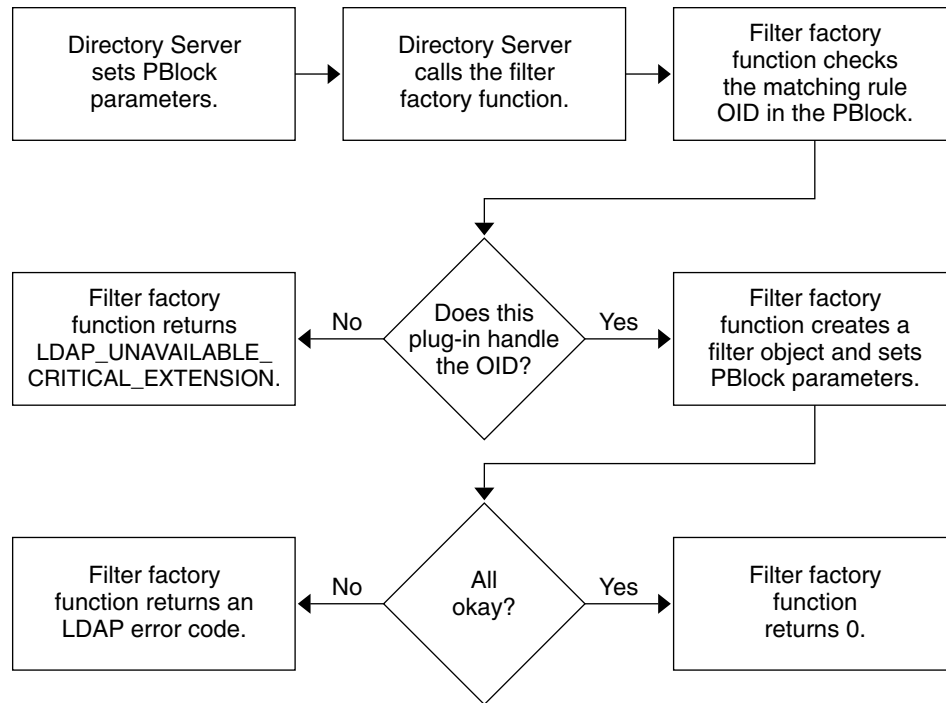


FIGURE 11-3 Filter Factory Function Context

The following example shows the filter factory function for the case exact matching rule.

EXAMPLE 11-5 Filter Factory Function (matchingrule.c)

```

#include "slapi-plugin.h"

#define PLG_OID      "1.3.6.1.4.1.42.2.27.999.4.1"
#define SUBSYS      "CaseExactMatching Plugin"

/* Functions to obtain connection information for logging. */
static long mypblock_get_msgid( Slapi_PBlock * pb);
static int  mypblock_get_connid(Slapi_PBlock * pb);
static int  mypblock_get_opid( Slapi_PBlock * pb);
  
```

EXAMPLE 11-5 Filter Factory Function (matchingrule.c) (Continued)

```

typedef struct plg_filter_t          /* For manipulating filter obj. */
{
    char      * f_type;              /* Attribute type to match */
    int       f_op;                  /* Type of comparison
                                     * (<, <=, ==, >=, >, substr)
                                     * for the filter. */
    struct berval ** f_values;        /* Array of values to match */
} plg_filter_t;

static int
plg_filter_create(Slapi_PBlock * pb)
{
    int rc = LDAP_UNAVAILABLE_CRITICAL_EXTENSION;
    char * mr_oid = NULL; /* MR OID from the server */
    char * mr_type = NULL; /* Attr type to match */
    struct berval * mr_value = NULL; /* Attr value to match */
    plg_filter_t * fobj = NULL; /* Object to create */

    if (
        slapi_pblock_get(pb, SLAPI_PLUGIN_MR_OID, &mr_oid) ||
        (mr_oid == NULL)
    ) {
        slapi_log_error_ex(
            -1, /* errorId */
            mypblock_get_msgid(pb),
            mypblock_get_connid(pb),
            mypblock_get_opid(pb),
            SUBSYS,
            SLAPI_INTERNAL_ERROR,
            "plg_filter_create failed: NULL OID values are invalid.\n"
        );
    }
    else if (strcmp(mr_oid, PLG_OID) == 0) {
        /* The MR OID from the server is handled by this plug-in. */
        if (
            (slapi_pblock_get(pb, SLAPI_PLUGIN_MR_TYPE, &mr_type) == 0) &&
            (mr_type != NULL) &&
            (slapi_pblock_get(pb, SLAPI_PLUGIN_MR_VALUE, &mr_value) == 0) &&
            (mr_value != NULL)
        ) { /* ...provide a pointer to a filter match function. */
            int op = SLAPI_OP_EQUAL;
            fobj = (plg_filter_t *)slapi_ch_calloc(
                1,
                sizeof (plg_filter_t)
            );
            fobj->f_type = slapi_ch_strdup(mr_type);

```

EXAMPLE 11-5 Filter Factory Function (matchingrule.c) *(Continued)*

```
fobj->f_op      = op;
fobj->f_values   = (struct berval **)slapi_ch_malloc(
                    2 * sizeof(struct berval *)
                );
fobj->f_values[0] = slapi_ch_bvdup(mr_value);
fobj->f_values[1] = NULL;

rc = slapi_pblock_set(          /* Set object destructor.   */
    pb,
    SLAPI_PLUGIN_DESTROY_FN,
    (void *)plg_filter_destroy
);
rc |= slapi_pblock_set(         /* Set object itself.   */
    pb,
    SLAPI_PLUGIN_OBJECT,
    (void *)fobj
);
rc |= slapi_pblock_set(         /* Set filter match fcn. */
    pb,
    SLAPI_PLUGIN_MR_FILTER_MATCH_FN,
    (void *)plg_filter_match
);
rc |= slapi_pblock_set(         /* Set sorting function. */
    pb,
    SLAPI_PLUGIN_MR_FILTER_INDEX_FN,
    (void *)plg_filter_index
);

if (rc == 0) {
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
        mypblock_get_msgid(pb),
        mypblock_get_connid(pb),
        mypblock_get_opid(pb),
        SUBSYS,
        "plg_filter_create (oid %s; type %s) OK\n",
        mr_oid, mr_type
    );
} else {
    slapi_log_error_ex(
        -1, /* errorId */
        mypblock_get_msgid(pb),
        mypblock_get_connid(pb),
        mypblock_get_opid(pb),
        SUBSYS,
```


EXAMPLE 11-5 Filter Factory Function (matchingrule.c) (Continued)

```

        SLAPI_INTERNAL_ERROR,
        "plg_filter_create (oid %s; type %s) - pblock error \n",
        mr_oid, mr_type
    );
}
} else { /* Missing parameters in the pblock */
    slapi_log_error_ex(
        -1,
        mypblock_get_msgid(pb),
        mypblock_get_connid(pb),
        mypblock_get_opid(pb),
        SUBSYS,
        "Parameter errors ",
        "plg_filter_create: invalid input pblock.\n"
    );
}
}

return rc;
}

```

Notice that this function returns `LDAP_UNAVAILABLE_CRITICAL_EXTENSION` if the function does not handle the matching rule OID in the parameter block. Refer to [“Handling an Unknown Matching Rule” on page 219](#).

Input Parameters for Filter Factory Functions

A filter factory function can get values for the following from the parameter block:

- `SLAPI_PLUGIN_MR_OID`
- `SLAPI_PLUGIN_MR_TYPE`
- `SLAPI_PLUGIN_MR_VALUE`
- `SLAPI_PLUGIN_PRIVATE`, if your plug-in uses private data that is specified in the plug-in initialization function

Output Parameters for Filter Factory Functions

An indexer factory function should set values for the following in the parameter block before returning control to Directory Server:

- `SLAPI_PLUGIN_DESTROY_FN`, a pointer to the destructor for the filter object
- `SLAPI_PLUGIN_MR_FILTER_INDEX_FN`, a pointer to the filter index function
- `SLAPI_PLUGIN_MR_FILTER_MATCH_FN`, a pointer to the filter match function
- `SLAPI_PLUGIN_MR_FILTER_RESET_FN`, if required, a pointer to the filter reset function

- `SLAPI_PLUGIN_MR_FILTER_REUSABLE`, if required to specify that the filter can be reused
- `SLAPI_PLUGIN_OBJECT`, a pointer to the filter object

Refer to [Chapter 18](#) for details.

Thread Safety and Filter Factory Functions

This function must be thread safe. Directory Server can call this function concurrently.

Filter Object Destructor Function

A filter object destructor function frees memory that was allocated for a filter object set up by the filter factory function. Directory Server calls the destructor after the operation completes, passing the parameter block indicating the filter object as the value of `SLAPI_PLUGIN_OBJECT`.

Directory Server never calls a destructor for the same object concurrently.

EXAMPLE 11-6 Filter Object and Destructor (`matchingrule.c`)

This code shows an example object and destructor.

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

typedef struct plg_filter_t          /* For manipulating filter obj. */
{
    char * f_type;                  /* Attribute type to match */
    int f_op;                       /* Type of comparison
                                   * (<, <=, ==, >=, >, substr)
                                   * for the filter. */
    struct berval ** f_values;      /* Array of values to match */
} plg_filter_t;

/* Free memory allocated for the filter object. */
static int
plg_filter_destroy(Slapi_PBlock * pb)
{
    void * obj = NULL;              /* Server lets the plug-in */
    plg_filter_t * fobj = NULL;     /* handle the object type. */

    if (!slapi_pblock_get(pb, SLAPI_PLUGIN_OBJECT, &obj))
    {
        fobj = (plg_filter_t *)obj;
        if (fobj){
            slapi_ch_free((void **)&fobj->f_type);
            if (fobj->f_values){
                ber_bvecfree(fobj->f_values);
            }
        }
    }
}
```

EXAMPLE 11-6 Filter Object and Destructor (matchingrule.c) (Continued)

```

    }
    slapi_ch_free((void **)&fobj);
}
}
return 0;
}

```

Indexing Entries According to a Matching Rule

This section covers how to provide plug-in support for a directory index that is based on a matching rule. Matching rule plug-ins are not required to enable indexing for the matching rules the plug-ins support.

Note – Directory Server can use your plug-in for extensible match searches even if you provide no indexing capability.

Without support for indexing, however, Directory Server must generate search results from the entire directory, which severely impacts search performance.

Directory indexes speed up client searches. Directory Server can build a list of search result entries by looking through the index rather than the entire directory. When directories contain many entries, indexes offer a considerable search performance boost. Directory administrators therefore configure the directory to maintain indexes for attributes that are searched regularly.

How Directory Server Handles the Index

To maintain an index for a custom matching rule supported by your plug-in, the server requires an *indexer function*. The indexer function is capable of translating attribute values to index keys. Directory Server calls the indexer function to create the index, and subsequently when adding, modifying, or deleting attribute values, because such changes can affect the index.

To read an index for a custom matching rule, Directory Server requires a *filter index function*. The filter factory function provides a pointer to this function. Refer to [“Handling Extensible Match Filters” on page 199](#) and [“Filter Index Function” on page 203](#) for details.

Directory Server relies on the *indexer factory function* in your plug-in to set up an indexing object and provide a pointer to the appropriate indexer function.

The following figure shows how Directory Server performs indexing based on a matching rule.

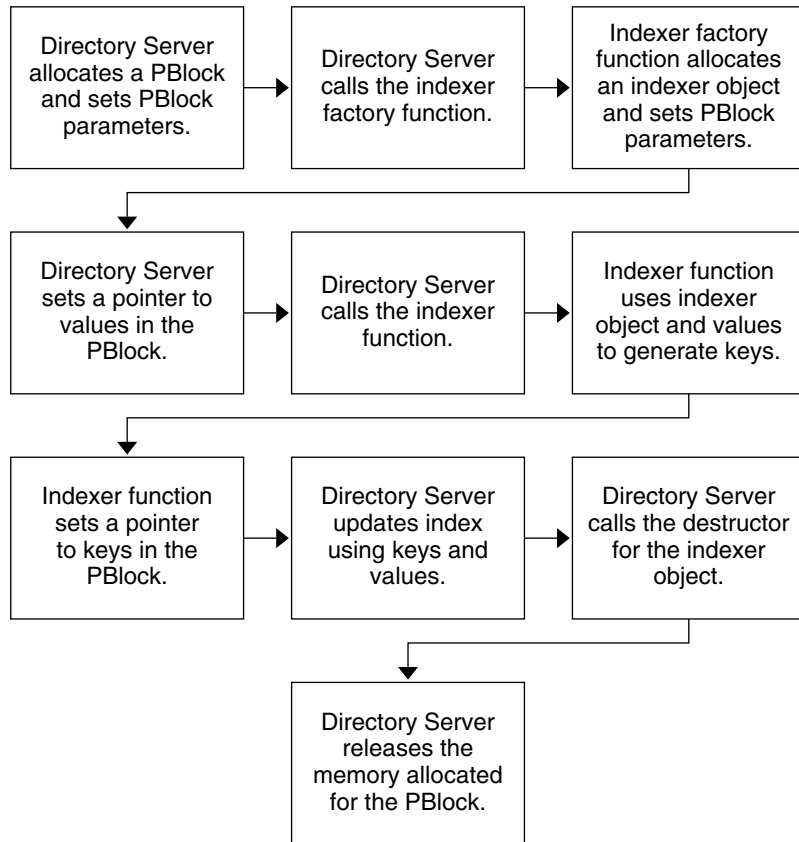


FIGURE 11-4 Directory Server Maintaining an Index Using a Matching Rule

The following summarizes the process shown in [Figure 11-4](#).

1. Directory Server creates a parameter block, setting parameters to indicate the matching rule OID and attribute type to match. Directory Server then calls the indexer factory function.
2. The indexer factory function examines the parameter block and allocates an indexer object if necessary. The function also sets pointers to the indexer and filter index functions. If necessary, the function sets a pointer to a destructor to free the indexer object before returning control to Directory Server.
3. Directory Server sets the parameter block to indicate attribute values to translate to keys and calls the indexer function.
4. The indexer function translates values to keys. After the indexer returns, Directory Server uses the keys to update the index.
5. Directory Server frees the parameter block. If necessary, the server frees the indexer object by using the destructor provided.

Be aware that Directory Server can call indexer factory functions concurrently. Indexer factory functions must therefore be thread safe.

The main filter functions are as follows:

- [Indexer function](#)
- [Indexer factory function](#)
- [Indexer object destructor function](#)

Indexer Function

An indexer function takes a parameter block from Directory Server that contains a pointer to a `berval` array of attribute values. The function then generates a `berval` array of corresponding keys from the values. Finally, the function sets a pointer in the parameter block to the keys.

The following figure shows how Directory Server uses the indexer function.

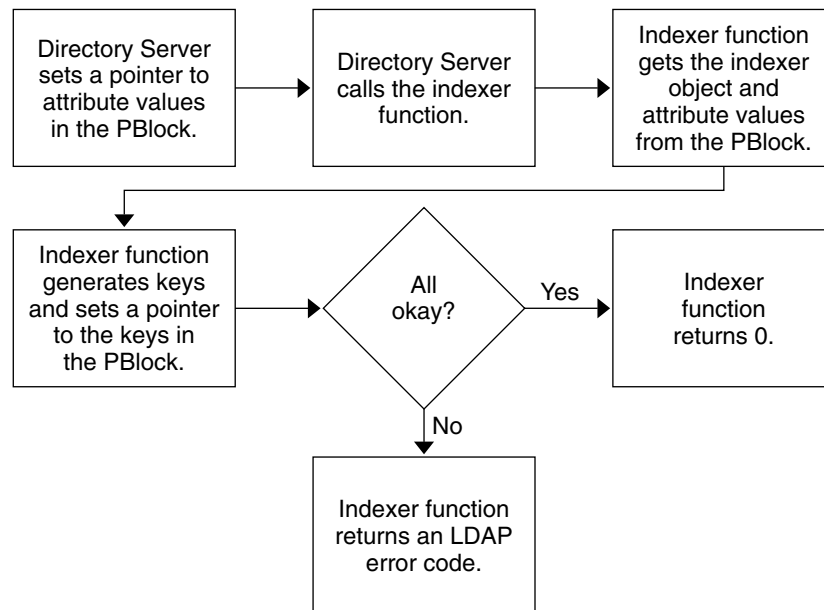


FIGURE 11-5 Indexer Function Context

The following example shows the indexer function for the case exact matching rule where the keys are the same as the values.

EXAMPLE 11-7 Indexer Function (matchingrule.c)

```

#include "slapi-plugin.h"

typedef struct plg_filter_t          /* For manipulating filter obj. */
{
    char * f_type;                  /* Attribute type to match */
    int f_op;                       /* Type of comparison
                                   * (<, <=, ==, >=, >, substr)
                                   * for the filter. */
    struct berval ** f_values;      /* Array of values to match */
} plg_filter_t;

static int
plg_index_entry(Slapi_PBlock * pb)
{
    int rc = LDAP_OPERATIONS_ERROR;
    void * obj = NULL; /* Server lets the plug-in */
    plg_filter_t * fobj; /* handle the object type. */
    struct berval ** values; /* Values from server */

    if (slapi_pblock_get(pb, SLAPI_PLUGIN_OBJECT, &obj) == 0) {
        fobj = (plg_filter_t *)obj;
        if (
            (slapi_pblock_get(pb, SLAPI_PLUGIN_MR_VALUES, &values) == 0) &&
            (values != NULL)
        ) {

            /* The case exact match builds the index keys
             * from the values by copying the values because
             * the keys and values are the same.
             *
             * Your matching rule may do something quite
             * different before setting the keys associated
             * with the values in the parameter block. */
            rc = slapi_pblock_set( /* Set keys based on values. */
                pb,
                SLAPI_PLUGIN_MR_KEYS,
                slapi_ch_bvecdup(values)
            );
        }
    }

    return rc;
}

```

Here, `obj` points to the indexer object, and `values` points to the `berval` array of attribute values. Notice that the function returns `LDAP_OPERATIONS_ERROR` on failure. Technically, `LDAP_OPERATIONS_ERROR` indicates bad sequencing of LDAP operations. For historical reasons,

EXAMPLE 11-7 Indexer Function (`matchingrule.c`) (Continued)

the error is used in this context to indicate an internal error.

Input Parameters for Indexers

An indexer function can get values for the following from the parameter block:

- `SLAPI_PLUGIN_MR_VALUES`
- `SLAPI_PLUGIN_MR_OBJECT`

Output Parameter for Indexers

An indexer function should generate the `berval` array of keys. The function should set `SLAPI_PLUGIN_MR_KEYS` in the parameter block before returning control to Directory Server. Refer to [Part II](#) for details.

Thread Safety and Indexers

Directory Server never calls an indexer for the same indexer object concurrently. Directory Server can, however, call the function concurrently for different indexer objects. If you use global variables, ensure that your function handles such variables safely.

Indexer Factory Function

The indexer factory function takes a parameter block from Directory Server. The function sets parameters such that Directory Server can update an index or sort results based on a matching rule.

The following figure shows how Directory Server uses the indexer factory function.

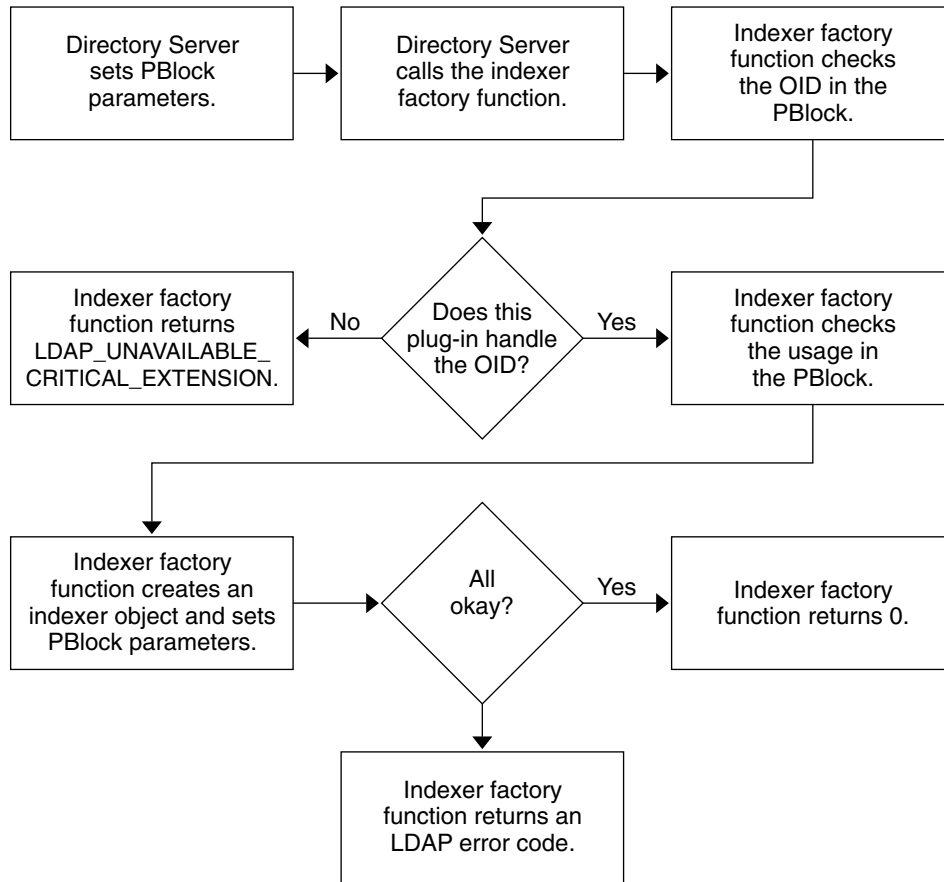


FIGURE 11-6 Indexer Factory Function Context

The following example shows the indexer factory function for the case exact matching rule.

EXAMPLE 11-8 Indexer Factory Function (matchingrule.c)

```

#include "slapi-plugin.h"

#define PLG_OID      "1.3.6.1.4.1.42.2.27.999.4.1"
#define SUBSYS      "CaseExactMatching Plugin"

/* Functions to obtain connection information for logging. */
static long mypblock_get_msgid( Slapi_PBlock * pb);
static int  mypblock_get_connid(Slapi_PBlock * pb);
static int  mypblock_get_opid( Slapi_PBlock * pb);

static int

```


EXAMPLE 11-8 Indexer Factory Function (matchingrule.c) (Continued)

```

plg_indexer_create(Slapi_PBlock * pb)
{
    int rc = LDAP_UNAVAILABLE_CRITICAL_EXTENSION; /* Init failed*/
    char * mr_oid = NULL; /* MR OID from the server */

    if (slapi_pblock_get(pb, SLAPI_PLUGIN_MR_OID, mr_oid) ||
        (mr_oid == NULL)) {
        slapi_log_error_ex(
            -1, /* errorId */
            mypblock_get_msgid(pb),
            mypblock_get_connid(pb),
            mypblock_get_opid(pb),
            SUBSYS,
            SLAPI_INTERNAL_ERROR,
            "plg_indexer_create failed: NULL OID values are invalid.\n"
        );
    } else if (strcmp(mr_oid, PLG_OID) == 0) {
        if ( /* The MR OID from the server is handled by this plug-in. */
            (slapi_pblock_set( /* This is for completeness. */
                pb, /* Your plug-in may set a */
                SLAPI_PLUGIN_MR_OID, /* different OID than the one */
                PLG_OID /* provided by the server. */
            ) == 0) &&
            (slapi_pblock_set( /* Provide an appropriate */
                pb, /* indexer function pointer. */
                SLAPI_PLUGIN_MR_INDEX_FN,
                (void *)plg_index_entry
            ) == 0) &&
            (slapi_pblock_set( /* Set the object destructor. */
                pb,
                SLAPI_PLUGIN_DESTROY_FN,
                (void *)plg_filter_destroy
            ) == 0)) {
            rc = LDAP_SUCCESS;
        } else {
            slapi_log_error_ex(
                -1, /* errorId */
                mypblock_get_msgid(pb),
                mypblock_get_connid(pb),
                mypblock_get_opid(pb),
                SUBSYS,
                SLAPI_INTERNAL_ERROR,
                "plg_indexer_create failed %d \n", rc
            );
        }
    }
}

```

EXAMPLE 11-8 Indexer Factory Function (`matchingrule.c`) *(Continued)*

```
        return rc;
    }
```

Here, `PLG_OID` is the object identifier for the matching rule. `plg_index_entry()` is the indexer. `plg_filter_destroy()` is the destructor. Notice that the function returns `LDAP_UNAVAILABLE_CRITICAL_EXTENSION` on failure.

Input Parameters for Indexer Factory Functions

An indexer factory function can read the values for the following from the parameter block:

- `SLAPI_PLUGIN_MR_OID`
- `SLAPI_PLUGIN_MR_TYPE`
- `SLAPI_PLUGIN_MR_USAGE`, indicates whether results must be sorted
- `SLAPI_PLUGIN_PRIVATE`, if your plug-in uses private data that you specify in the plug-in initialization function

Output Parameters for Indexer Factory Functions

An indexer factory function should set values for at least the following in the parameter block before returning control to Directory Server:

- `SLAPI_PLUGIN_DESTROY_FN`, a pointer to the indexer object destructor
- `SLAPI_PLUGIN_MR_INDEX_FN`, a pointer to the indexer
- `SLAPI_PLUGIN_OBJECT`, a pointer to the indexer object

Refer to [Chapter 18](#) for details about the parameter block.

Thread Safety and Indexer Factory Functions

This function must be thread safe. Directory Server can call this function concurrently.

Indexer Object Destructor Function

An indexer object destructor function frees memory that was allocated for an indexer object set up by the indexer factory function. Directory Server calls the destructor after an index operation completes.

Indexer object destructors take a parameter block as their only argument, as do filter object destructors. The parameter block holds a pointer to the object in `SLAPI_PLUGIN_OBJECT`. Refer to the [“Filter Object Destructor Function” on page 210](#) for details.

Directory Server never calls a destructor for the same object concurrently.

Enabling Sorting According to a Matching Rule

Clients can request that Directory Server sort results from an extensible match search. This section explains how to enable sorting based on a matching rule.

How Directory Server Performs Sorting According to a Matching Rule

Directory Server performs sorting as a variation of indexing, using the keys generated by an indexer function to sort results. The process is as follows:

1. Directory Server creates a parameter block as for indexing, setting `SLAPI_PLUGIN_MR_USAGE` to `SLAPI_PLUGIN_MR_USAGE_SORT`, before passing the parameter block to the indexer factory function.
2. The indexer factory function should set parameters in the parameter block as for indexing. If the sort function is different from the normal indexer function, ensure that the function checks the value of `SLAPI_PLUGIN_MR_USAGE`, and then sets `SLAPI_MR_INDEXER_FN` accordingly.
3. Directory Server sets `SLAPI_PLUGIN_MR_VALUES` in the parameter block as a pointer to the values to be sorted. Directory Server then passes the parameter block the indexer function.
4. Directory Server sorts results based on the keys the indexer function set in `SLAPI_PLUGIN_MR_KEYS`.
5. Directory Server frees memory that was allocated for the operation.

Refer to [“Indexing Entries According to a Matching Rule” on page 211](#) for details on how matching rule plug-ins can allow Directory Server to perform indexing based on a matching rule.

Handling an Unknown Matching Rule

This section explains how Directory Server finds a plug-in for a matching rule OID not recognized by that Directory Server.

Internal List of Correspondences

Directory Server identifies matching rules by OID. The server keeps an internal list of which matching rule plug-ins handle which OIDs.

Directory Server uses the internal list to determine which plug-in to call when the server receives an extensible match filter search request that includes a matching rule OID. Directory

Server initially builds the list as matching rule plug-ins register OIDs that the plug-ins handle using `slapi_matchingrule_register()` in the plug-in initialization function.

OIDs Not in the Internal List

When the server encounters a matching rule OID that is not in the list, Directory Server queries each matching rule plug-in through registered factory functions. For every matching rule plug-in, Directory Server passes a parameter block that contains the OID to the matching rule factory function. The server then checks whether the factory function has in return provided a pointer in the parameter block. This pointer identifies the appropriate indexing or matching function for the OID.

If the factory function sets a pointer to the appropriate function, Directory Server assumes that the plug-in supports the matching rule.

The following shows how Directory Server checks whether a plug-in handles a specific filter match operation. The process for indexing closely resembles the process for matching.

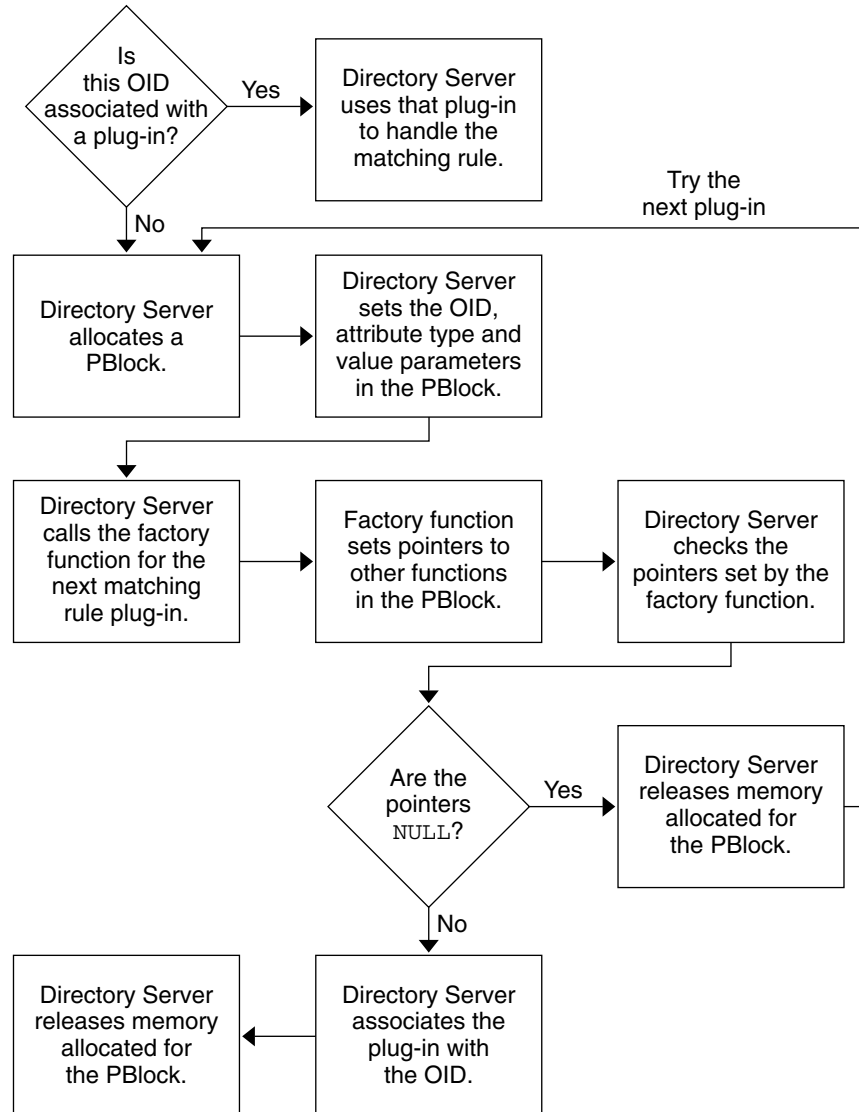


FIGURE 11-7 Finding a Matching Rule Plug-In for an Unknown OID

The following summarizes the process shown in [Figure 11-7](#).

1. Directory Server finds no match in the internal list of plug-ins to handle the OID. The server therefore creates a parameter block, and sets appropriate parameters for the factory, including `SLAPI_PLUGIN_MR_OID`.
2. Directory Server calls the factory function. The factory function either sets a pointer in the parameter block to the appropriate function to handle the matching rule, or the function leaves the pointer `NULL`.
3. Directory Server checks the parameter block after the factory function returns. If the factory function has set a function pointer in the parameter block, Directory Server updates its internal list. The function thus indicates that the plug-in supports the matching rule operation that is associated with the OID. Otherwise, the pointer remains `NULL`, and Directory Server tries the process again on the next matching rule plug-in that is registered.

If after calling all matching rule plug-ins, Directory Server has found no plug-in to handle the matching rule OID, the server returns `LDAP_UNAVAILABLE_CRITICAL_EXTENSION` to the client.
4. Directory Server frees the memory that was allocated for the operation.

Be aware that Directory Server calls plug-in factory functions for the purpose of extending the internal list of correspondences.

Writing Password Storage Scheme Plug-Ins

This chapter explains how to write plug-ins that allow you to modify how Directory Server stores password attribute values.

This chapter covers the following topics:

- [“Calling Password Storage Scheme Plug-Ins” on page 223](#)
- [“Writing a Password Storage Scheme Plug-In” on page 225](#)

Calling Password Storage Scheme Plug-Ins

This section describes the circumstances in which Directory Server calls password storage scheme plug-ins. This section also describes how password values are expected to be handled by the plug-ins.

Types of Password Storage Scheme Plug-Ins

Two types of password storage scheme plug-ins work with Directory Server, `pwdstoragescheme` and `reverpwdstoragescheme`. The `pwdstoragescheme` type is one-way. After the server encodes and stores a password, the password is not decoded. The `pwdstoragescheme` type therefore includes plug-in functions only for encoding passwords to be stored and for comparing incoming passwords with encoded, stored passwords. The `reverpwdstoragescheme` type is reversible, in that the plug-in allows Directory Server to encode and decode values. The reversible type therefore includes encode, compare, and decode plug-in functions.

Note – This chapter covers the one-way type `pwdstorage` scheme plug-ins.

The reversible type is for internal use only.

Preinstalled Schemes

Existing schemes delivered with Directory Server are provided as password storage scheme plug-ins. Search `cn=config` for entries whose DN contains `cn=Password Storage Schemes`. The default password storage scheme uses the Salted Secure Hashing Algorithm (SSHA).

You can change the password storage scheme used to encode user passwords. See Chapter 7, “Directory Server Password Policy,” in *Sun Java System Directory Server Enterprise Edition 6.1 Administration Guide* for instructions.

Effect on Password Attribute Values

Password storage scheme plug-in functions act on `userPassword` attribute values. Directory Server registers password storage scheme plug-ins at startup. After startup, any registered, enabled password storage scheme plug-in can then be used to encode password values. The plug-ins can also be used to compare incoming passwords to the encoded values. Which plug-in Directory Server invokes depends on the password storage scheme that is used for the entry in question.

Invocation for Add and Modify Requests

Add and modify requests can imply that Directory Server encode an input password, and then store it in the directory. First, Directory Server determines the storage scheme for the password value. Next, it invokes the plug-in encode function for the appropriate scheme. The encode function returns the encoded password to Directory Server.

Invocation for Bind Requests

Bind requests imply that Directory Server compares an input password value to a stored password value. As for add and modify requests, Directory Server determines the storage scheme for the password value. Next, the server invokes the plug-in compare function for the appropriate scheme. The compare scheme returns an `int` that communicates to Directory Server whether the two passwords match as described in [“Comparing a Password” on page 227](#).

Part of a Password Policy

Password storage scheme plug-ins typically do no more than encode passwords and compare input passwords with stored, encoded passwords. In other words, plug-ins represent only a part of a comprehensive password policy. Refer to the *Sun Java System Directory Server Enterprise Edition 6.1 Deployment Planning Guide* for suggestions on designing secure directory services.

Writing a Password Storage Scheme Plug-In

This section demonstrates how to write a plug-in that encodes passwords. The plug-in also allows Directory Server to compare stored passwords with passwords provided by a client application.



Caution – The examples in this chapter do *not* constitute a secure password storage scheme.

The source for the example plug-in referenced in this chapter is `install-path/examples/testpwdstore.c`. For encoding and comparing, the plug-in performs an exclusive or with 42 on each character of the password.

Encoding a Password

When Directory Server calls a password storage scheme plug-in encode function, it passes that function an input password `char *` and expects an encoded password `char *` in return. The prototype for the example encode function, `xorenc()`, is as follows:

```
static char * xorenc(char * pwd);
```

Allocate space for the encoded password with `slapi_ch_malloc()` rather than regular `malloc()`. Directory Server can then terminate with an “out of memory” message if allocation fails memory with `slapi_ch_free()`.

By convention, you prefix the encoded password with the name of the password storage scheme, enclosed in braces, { and }. In other words, the example plug-in is called XOR.

The name is declared in the example:

```
static char * name = "XOR"; /* Storage scheme name */
```

You return encoded strings prefixed with {XOR}. You also register the name with Directory Server.

EXAMPLE 12-1 Encoding a userPassword Value (testpwdstore.c)

```

#include "slapi-plugin.h"

static char * name          ="XOR";    /* Storage scheme name */

#define PREFIX_START '{'
#define PREFIX_END   '}'

static char *
xorenc(char * pwd)
{
    char * tmp      = NULL;            /* Used for encoding */
    char * head     = NULL;            /* Encoded password */
    char * cipher   = NULL;            /* Prefix, then pwd */
    int i, len;

    /* Allocate space to build the encoded password */
    len = strlen(pwd);
    tmp = slapi_ch_malloc(len + 1);
    if (tmp == NULL) return NULL;

    memset(tmp, '\0', len + 1);
    head = tmp;

    /* Encode. This example is not secure by any means. */
    for (i = 0; i < len; i++, pwd++, tmp++) *tmp = *pwd ^ 42;

    /* Add the prefix to the cipher */
    if (tmp != NULL) {
        cipher = slapi_ch_malloc(3 + strlen(name) + strlen(head));
        if (cipher != NULL) {
            sprintf(cipher, "%c%s%c%s", PREFIX_START, name, PREFIX_END, head);
        }
    }
    slapi_ch_free((void **) &head);

    return (cipher);                    /* Server frees cipher */
}

```

Notice that you free only memory allocated for temporary use. Directory Server frees memory for the char * returned, not the plug-in. For details on `slapi_ch_malloc()` and `slapi_ch_free()`, see [Chapter 16](#).

Comparing a Password

When Directory Server calls a password storage scheme plug-in compare function, it passes that function an input password `char *` and a stored, encoded password `char *` from the directory. The compare function returns zero, 0, if the input password matches the password from the directory. The function returns 1 otherwise. The prototype for the example compare function, `xorcmp()`, is therefore as follows:

```
static int xorcmp(char * userpwd, char * dbpwd);
```

Here, `userpwd` is the input password. `dbpwd` is the password from the directory. The compare function must encode the input password to compare the result to the password from the directory.

EXAMPLE 12-2 Comparing a `userPassword` Value (`testpwdstore.c`)

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

static int
xorcmp(char * userpwd, char * dbpwd)
{
    /* Check the correspondence of the two char by char */
    int i, len = strlen(userpwd);
    for (i = 0; i < len; i++) {
        if ((userpwd[i] ^ 42) != dbpwd[i])
            return 1;          /* Different passwords */
    }
    return 0;                  /* Identical passwords */
}
```

Notice that Directory Server strips the prefix from the password before passing the value to the compare function. In other words, you need not account for {XOR} in this case.

Not all encoding algorithms have such a trivial compare function.

Registering the Password Storage Scheme Plug-In

You must register four password storage scheme specific items with Directory Server:

- The storage scheme name that is used for the prefix
- The encode function
- The compare function
- The decode function

Notice that you provide no decoding function. In this case, Directory Server does not decode user passwords after they are stored.

EXAMPLE 12-3 Registering a Password Storage Scheme Plug-In (testpwdstore.c)

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

static char * name          = "XOR";    /* Storage scheme name */

static Slapi_PluginDesc desc = {
    "xor-password-storage-scheme",      /* Plug-in identifier */
    "Sun Microsystems, Inc.",           /* Vendor name */
    "6.0",                              /* Revision number */
    "Exclusive-or example (XOR)"        /* Plug-in description */
};

#ifdef _WIN32
__declspec(dllexport)
#endif
int
xor_init(Slapi_PBlock * pb)
{
    int rc = 0;                        /* 0 means success */
    rc |= slapi_pblock_set(             /* Plug-in API version */
        pb,
        SLAPI_PLUGIN_VERSION,
        (void *) SLAPI_PLUGIN_CURRENT_VERSION
    );
    rc |= slapi_pblock_set(             /* Plug-in description */
        pb,
        SLAPI_PLUGIN_DESCRIPTION,
        (void *) &desc
    );
    rc |= slapi_pblock_set(             /* Storage scheme name */
        pb,
        SLAPI_PLUGIN_PWD_STORAGE_SCHEME_NAME,
        (void *) name
    );
    rc |= slapi_pblock_set(             /* Encode password */
        pb,
        SLAPI_PLUGIN_PWD_STORAGE_SCHEME_ENC_FN,
        (void *) xorenc
    );
    rc |= slapi_pblock_set(             /* Compare password */
        pb,
        SLAPI_PLUGIN_PWD_STORAGE_SCHEME_CMP_FN,
        (void *) xorcmp
    );
    rc |= slapi_pblock_set(             /* Never decode pwd */
        pb,
        SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DEC_FN,
```

EXAMPLE 12-3 Registering a Password Storage Scheme Plug-In (`testpwdstore.c`) (Continued)

```

        NULL
    );
    return rc;
}

```

Setting Up the Password Storage Scheme Plug-In

Set up a directory instance and build the plug-in if you have not done so already.

▼ To Register the Plug-In

If you have not already done so, build the example plug-in library and activate both plug-in informational logging and the example plug-in.

1 Build the plug-in.

Hint Use *install-path/examples/Makefile* or *install-path/examples/Makefile64*.

2 Configure Directory Server to log plug-in informational messages and load the plug-in.

Hint Use the commands specified in the comments at the outset of the plug-in source file.

3 Restart Directory Server.

```
$ dsadm restart instance-path
```

▼ To Set Up an Example Suffix

If you have not done so already, set up a directory instance with a suffix, `dc=example,dc=com`, containing data loaded from a sample LDIF file, *install-path/ds6/ldif/Example.ldif*.

1 Create a new Directory Server instance.

For example:

```
$ dsadm create /local/ds
Choose the Directory Manager password:
Confirm the Directory Manager password:
$
```

2 Start the new Directory Server instance.

For example:

```
$ dsadm start /local/ds
Server started: pid=4705
$
```

3 Create a suffix called dc=example,dc=com.

For example, with long lines folded for the printed page:

```
$ dsconf create-suffix -h localhost -p 1389 dc=example,dc=com
Enter "cn=directory manager" password:
Certificate "CN=defaultCert, CN=hostname:1636" presented by the
server is not trusted.
Type "Y" to accept, "y" to accept just once,
  "n" to refuse, "d" for more details: Y
$
```

4 Load the sample LDIF.

For example, with long lines folded for the printed page:

```
$ dsconf import -h localhost -p 1389 \
/opt/SUNWdsee/ds6/ldif/Example.ldif dc=example,dc=com
Enter "cn=directory manager" password:
New data will override existing data of the suffix
  "dc=example,dc=com".
Initialization will have to be performed on replicated suffixes.
Do you want to continue [y/n] ? y

## Index buffering enabled with bucket size 16
## Beginning import job...
## Processing file "/opt/SUNWdsee/ds6/ldif/Example.ldif"
## Finished scanning file "/opt/SUNWdsee/ds6/ldif/Example.ldif" (160 entries)
## Workers finished; cleaning up...
## Workers cleaned up.
## Cleaning up producer thread...
## Indexing complete.
## Starting numsubordinates attribute generation.
  This may take a while, please wait for further activity reports.
## Numsubordinates attribute generation complete. Flushing caches...
## Closing files...
## Import complete. Processed 160 entries in 5 seconds.
  (32.00 entries/sec)

Task completed (slapd exit code: 0).
$
```

See Also You can use Directory Service Control Center to perform this task. For more information, see the Directory Service Control Center online help.

Trying the Password Storage Scheme Example

This section demonstrates the example plug-in for this chapter.

Perform a Quick Test

Plug the XOR password storage scheme into Directory Server if you have not done so already.

Before you do anything else, quickly check that Directory Server calls the plug-in encode function as expected. To perform this quick test, use the `pwdhash` tool. The `pwdhash` tool has Directory Server encode a password, then display the result.

EXAMPLE 12-4 Testing the Password Storage Scheme

```
$ pwdhash -D /local/ds -s XOR password
{XOR}ZKYY]EXN
```

Do not be concerned with the exact value of the resulting encoded password. The output should, however, start with `{XOR}`.

As Directory Server calls the encode function dynamically, you can fix the plug-in library. Then try `pwdhash` without doing anything to Directory Server. If this quick test does not work, fix the example.

▼ To Encode a Password With the XOR Scheme

Here, you use the XOR scheme to encode a new password for Barbara Jensen.

1 Change the password storage scheme for the suffix to XOR.

```
$ dsconf set-server-prop -h localhost -p 1389 pwd-storage-scheme:XOR
```

2 Change Barbara's password to password.

3 View Barbara's newly encoded password.

```
$ ldapsearch -h localhost -p 1389 -b dc=example,dc=com uid=bjensen
version: 1
dn: uid=bjensen, ou=People, dc=example,dc=com
cn: Barbara Jensen
cn: Babs Jensen
sn: Jensen
givenName: Barbara
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
ou: Product Development
ou: People
l: Cupertino
uid: bjensen
mail: bjensen@example.com
```

```
telephoneNumber: +1 408 555 1862
facsimileTelephoneNumber: +1 408 555 1992
roomNumber: 0209
userPassword: {XOR}ZKYY]EXN
```

Notice that Barbara's password is XOR-encoded.

Compare an XOR-Encoded Password

Barbara has the right to search other entries under `dc=example,dc=com`. Here, you search for Kirsten Vaughan's entry as `bjensen`.

EXAMPLE 12-5 Binding With the New Password

```
$ ldapsearch -h localhost -p 1389 -b dc=example,dc=com
-D uid=bjensen,ou=People,dc=example,dc=com -w password uid=kvaughan
version: 1
dn: uid=kvaughan, ou=People, dc=example,dc=com
cn: Kirsten Vaughan
sn: Vaughan
givenName: Kirsten
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
ou: Human Resources
ou: People
l: Sunnyvale
uid: kvaughan
mail: kvaughan@example.com
telephoneNumber: +1 408 555 5625
facsimileTelephoneNumber: +1 408 555 3372
roomNumber: 2871
```

You know that Directory Server uses a plug-in to check Barbara's password during the bind. Thus, Directory Server must have used the XOR plug-in because you saw that Barbara's password was XOR-encoded. If the whole process appears to work, you can conclude that the compare function works, too.

Writing Password Quality Check Plug-Ins

Directory Server is delivered with a configurable password quality check plug-in. The plug-in allows you to guard against users who change their password to a value that is susceptible to typical English-language dictionary attacks. You might decide however to extend password quality checking further. This chapter shows how to write custom plug-ins that check password quality when the password is modified.

This chapter covers the following topics:

- [“How Directory Server Uses Password Quality Check Plug-Ins” on page 233](#)
- [“Writing a Custom Password Quality Check Plug-In” on page 235](#)

How Directory Server Uses Password Quality Check Plug-Ins

This section explains how Directory Server password policy configurations affect password quality checks. This section also explains what Directory Server expects your password quality check plug-in to do.

Password Policy to Check Password Quality

One aspect of password policy involves refusing passwords that do not meet your quality criteria. When a user submits a password value, that value is to be stored on her entry in the `userPassword(5dsat)` attribute. You might want the server to ensure the password value is not easy to guess or to discover. You might also want the server to log a warning when a weak password is accepted, or even to refuse weak passwords. Determining what you want is part of setting up a password policy.

The password policy entry attribute that governs to what extent the server checks password quality is `pwdCheckQuality(5dsat)`. When this attribute is set to cause the server to check password quality, the server can call plug-ins to do so. This chapter shows how to write a plug-in that checks password quality.

For a more extensive explanation of password policy configuration, see Chapter 7, “Directory Server Password Policy,” in *Sun Java System Directory Server Enterprise Edition 6.1 Administration Guide*.

Whether to Implement a Password Check Plug-In

If you can achieve your password policy requirements by using the strong password check plug-in provided with Directory Server, do not write your own plug-in.

The strong password check plug-in provided with Directory Server can be configured for your deployment, checking the password quality aspects that you consider essential. You can configure the following:

- The number of lowercase, uppercase, numeric, and special characters required in a password value
A special configuration setting also allows you to enforce a mix of characters.
- The absence of prohibited strings, read from a dictionary file
You can use the default English-language file that is provided. Alternatively, you can add or substitute your own file.

These configuration settings are in addition to other checks. Such checks govern password length, whether the password matches common attributes on the entry, and so forth.

In many cases, you can avoid writing your own password check plug-in.

What a Password Check Plug-In Must Do

If you decide to implement your own password check plug-in, it must be of type `passwordcheck`. Your plug-in must implement at least the password checking function, `SLAPI_PLUGIN_PASSWDCHECK_FN`. This function verifies that all passwords supplied in the parameter block array, `SLAPI_PASSWDCHECK_VALS`, satisfy the quality check. The `SLAPI_PASSWDCHECK_VALS` may contain more than one password value.

The function must return the following values, depending on the password values to check.

- 1 Return this value when you must send a failure result to the client application for a reason other than quality check failure. You might return this value for example when the server is busy loading a large dictionary file. The server then would not be ready to check password quality.

When you return this value, the server aborts the add or modify operation in progress.
- 0 Return this value when all passwords in `SLAPI_PASSWDCHECK_VALS` satisfy the password quality check.

`>0` Return a positive integer when at least one of the passwords in `SLAPI_PASSWDCHECK_VALS` fails the password quality check.

When you return a positive integer, pass an error message back through `SLAPI_RESULT_TEXT` in the parameter block with `slapi_pblock_set()`. Allocate space for the message using `slapi_ch_malloc()`. The server takes responsibility to free the memory allocated for the message.

The message string must start with `invalid password syntax:` , exactly as it is given here.

The server sends the client application a result of `LDAP_CONSTRAINT_VIOLATION`, 19. If, however, `passwordRootdnMayBypassModsChecks(5dsat)` is set to allow password administrators, including the directory root DN user, to modify passwords irrespective of password policy checks, the server nevertheless accepts modifications by that user.

In the simple case used in this chapter, the work is minimal. For real—world cases, the work might however be significant, particularly if you choose to implement your own dictionary check. If you implement your own dictionary check, the plug-in might cache the dictionary content from a large file. In this case, load the dictionary as part of a plug-in start function rather than part of plug-in initialization. You thus avoid blocking startup of the server while the cache is constructed.

Writing a Custom Password Quality Check Plug-In

This section demonstrates how to write a plug-in that performs a simple password quality check.

Only code excerpts are shown in this chapter. The complete code example can be found where you installed Directory Server, *install-path/ds6/examples/pwdcheck.c*.

Checking Password Values

When Directory Server receives a request to add or modify a `userPassword` value, the server calls the registered `passwordcheck` plug-in. The server passes one or more values as a set of `Slapi_Value` structures in the parameter block. You can retrieve these values with `slapi_pblock_get()`.

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

static int
check_pwd(Slapi_PBlock * pb)
```

```
{
    Slapi_Value ** pwdvals = NULL;
    slapi_pblock_get(pb, SLAPI_PASSWDCHECK_VALS, &pwdvals);
}
```

Your code must then return zero, 0, when password values are acceptable. Your code must return nonzero when password values are unacceptable. In the simple case where bad password values are only those equal to `secret12`, the code is a quick `strcmp`.

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

/* Reject password values equal to secret12. */
static int
check_pwd(Slapi_PBlock * pb)
{
    Slapi_Value ** pwdvals = NULL;

    ...

    /* Do not check values if none exist. */
    if (pwdvals == NULL) return 0;

    for (i=0 ; pwdvals[i] != NULL; i++) {
        const char * password = slapi_value_get_string(pwdvals[i]);
        if (strcmp("secret12", password) == 0) {
            slapi_pblock_set(pb, SLAPI_RESULT_TEXT,
                slapi_ch_strdup("invalid password syntax: Bad password!"));
            return 1;
        }
    }
    return 0;
}
```

Here, the code has Directory Server reject a password only when its value is `secret12`.

Initializing the Password Check Plug-In

Password check plug-ins register a password checking function, `SLAPI_PLUGIN_PASSWDCHECK_FN`, during initialization with the server.

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

int
pwdcheck_init(Slapi_PBlock * pb)
{
    int rc = 0;
```

```

rc |= slapi_pblock_set(
    pb,
    SLAPI_PLUGIN_VERSION,
    SLAPI_PLUGIN_CURRENT_VERSION
);

rc |= slapi_pblock_set(
    pb,
    SLAPI_PLUGIN_DESCRIPTION,
    (void *) &pwd_desc; /* See the code for pwd_desc. */
);

rc |= slapi_pblock_set(
    pb,
    SLAPI_PLUGIN_PASSWDCHECK_FN,
    (void *) check_pwd
);
return rc;
}

```

You can implement your own dictionary check that caches the dictionary content from a large file. In this case, load the dictionary as part of a plug-in start function rather than part of plug-in initialization. Thus, you avoid blocking startup of the server while the cache is constructed.

Testing the Password Check Plug-In

You test the plug-in using sample data delivered with Directory Server. You use command-line tools to setup and register the plug-in.

▼ To Set Up an Example Suffix

If you have not done so already, set up a directory instance with a suffix, `dc=example,dc=com`, containing data loaded from a sample LDIF file, *install-path/ds6/ldif/Example.ldif*.

1 Create a new Directory Server instance.

For example:

```

$ dsadm create /local/ds
Choose the Directory Manager password:
Confirm the Directory Manager password:
$

```

2 Start the new Directory Server instance.

For example:

```
$ dsadm start /local/ds
Server started: pid=4705
$
```

3 Create a suffix called dc=example,dc=com.

For example, with long lines folded for the printed page:

```
$ dsconf create-suffix -h localhost -p 1389 dc=example,dc=com
Enter "cn=directory manager" password:
Certificate "CN=defaultCert, CN=hostname:1636" presented by the
server is not trusted.
Type "Y" to accept, "y" to accept just once,
"n" to refuse, "d" for more details: Y
$
```

4 Load the sample LDIF.

For example, with long lines folded for the printed page:

```
$ dsconf import -h localhost -p 1389 \
/opt/SUNWdsee/ds6/ldif/Example.ldif dc=example,dc=com
Enter "cn=directory manager" password:
New data will override existing data of the suffix
"dc=example,dc=com".
Initialization will have to be performed on replicated suffixes.
Do you want to continue [y/n] ? y

## Index buffering enabled with bucket size 16
## Beginning import job...
## Processing file "/opt/SUNWdsee/ds6/ldif/Example.ldif"
## Finished scanning file "/opt/SUNWdsee/ds6/ldif/Example.ldif" (160 entries)
## Workers finished; cleaning up...
## Workers cleaned up.
## Cleaning up producer thread...
## Indexing complete.
## Starting numsubordinates attribute generation.
This may take a while, please wait for further activity reports.
## Numsubordinates attribute generation complete. Flushing caches...
## Closing files...
## Import complete. Processed 160 entries in 5 seconds.
(32.00 entries/sec)

Task completed (slapd exit code: 0).
$
```

See Also You can use Directory Service Control Center to perform this task. For more information, see the Directory Service Control Center online help.

▼ To Register the Plug-In

If you have not already done so, build the example plug-in library and activate both plug-in informational logging and the example plug-in.

1 Build the plug-in.

Hint Use *install-path/examples/Makefile* or *install-path/examples/Makefile64*.

2 Configure Directory Server to log plug-in informational messages and load the plug-in.

Hint Use the commands specified in the comments at the outset of the plug-in source file.

3 Restart Directory Server.

```
$ dsadm restart instance-path
```

▼ To Use the Password Check Plug-In

Before You Begin Populate the suffix `dc=example`, `dc=com` with sample data. Also, register the plug-in with Directory Server.

1 Enforce password quality checking so Directory Server calls your password check plug-in.

```
$ dsconf set-server-prop -h localhost -p 1389 \
  pwd-check-enabled:on pwd-strong-check-enabled:off
```

2 Enable logging of informational messages.

```
$ dsconf set-log-prop -h localhost -p 1389 error level:err-plugins
```

3 Prepare an entry that tests your password quality check.

```
$ cat quentin.ldif
dn: uid=qcubbins,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: qcubbins
givenName: Quentin
sn: Cubbins
cn: Quentin Cubbins
mail: quentin.cubbins@example.com
userPassword: secret12
```

4 Add the entry to the directory.

```
$ ldapmodify -a -D uid=kvaughan,ou=people,dc=example,dc=com \  
-w bribery -h localhost -p 1389 -f quentin.ldif
```

```
adding new entry uid=qcubbins,ou=People,dc=example,dc=com  
ldap_add_s: Constraint violation
```

5 Check the errors log for further information.

```
$ grep secret12 /local/ds/logs/errors  
[16/Feb/2006:18:13:06 +0100] - INFORMATION -  
Sample password check plug-in - conn=0 op=1 msgId=2 -  
Invalid password: secret12
```

The example log message as shown has been wrapped for readability in the printed version of this document.

Writing Computed Attribute Plug-Ins

This chapter describes a plug-in that computes an attribute value when that value is requested by a client application.

This chapter covers the following topics:

- [“Computed Attributes and Performance” on page 241](#)
- [“Writing a Computed Attribute Plug-In” on page 242](#)

Computed Attributes and Performance

Unlike most attributes in Directory Server, computed attributes do not have values stored in the Directory Server database. Computed attributes become available when the values are requested. Therefore, the attribute values must be computed at the time of the request, every time the values are requested.

In general, therefore, computed attributes should be quick to process. Computed attributes that depend on large internal searches, or even access to data not contained in the directory, can be expensive to generate. Therefore, such attributes are not ideal candidates for computed attributes.

Instead computed attributes that generate values quickly from the data at hand are more typical. For example, you might use a computed attribute to count or otherwise process the values of other attributes in an entry. This chapter shows a simple, artificial example that uses the API to get the current time when reading an entry.

Writing a Computed Attribute Plug-In

This section demonstrates how to write a plug-in that generates the value of an attribute, `currentTime`.

Only code excerpts are shown in this chapter. The complete code example can be found where you installed Directory Server, *install-path*/ds6/examples/computed.c.

Initializing the Computed Attribute Plug-In

Plug-ins that compute attribute values register the functions of type `slapi_compute_callback_t` to compute attributes with `slapi_compute_add_evaluator()` during initialization with the server.

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

int
compute_init(Slapi_PBlock * pb)
{
    int rc = 0;                                /* 0 means success */
    rc |= slapi_pblock_set(                     /* Plug-in API version */
        pb,
        SLAPI_PLUGIN_VERSION,
        SLAPI_PLUGIN_CURRENT_VERSION
    );
    rc |= slapi_pblock_set(                     /* Plug-in description */
        pb,
        SLAPI_PLUGIN_DESCRIPTION,
        (void *) &comp_desc;                  /* See the code for comp_desc. */
    );
    rc |= slapi_compute_add_evaluator(compute_attrs);
    return rc;
}
```

Here, `compute_attrs()` is the function that computes the attribute value.

Computing an Attribute Value

When Directory Server receives a request for the computed attribute value, in this case `currentTime`, the server calls the functions that are registered to compute attributes. Directory Server provides your function the context, attribute type, and entry with which to compute the attribute.

Your function must therefore check the attribute type to see whether your function should handle the computation. If your function does not handle the attribute type in question, return -1. If your function can handle the attribute, your function must determine the value or values of the attribute. Your function must then add the computed content to the result that the server returns. The following code excerpt shows the `compute_attrs()` function generating a current time string for the attribute value.

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

/* Compute an attribute called currentTime */
static int
compute_attrs(
    computed_attr_context * context,
    char                  * type,
    Slapi_Entry           * entry,
    slapi_compute_output_t outfn
)
{
    /* If the attribute type is not recognized return -1. */
    int rc = -1;

    if (slapi_attr_type_cmp("currentTime", type, 2) == 0) {
        Slapi_Attr * time_attr;
        time_t      now;
        char         current_time[30];
        Slapi_Value * time_value;
        int          rc = 0;

        /* Construct the computed attribute. */
        time_attr = slapi_attr_new();
        slapi_attr_init(time_attr, type);
        now = time(NULL);
        ctime_r(&now, current_time, 30);
        current_time[(int)strlen(current_time) - 1] = '\0';
        time_value = slapi_value_new_string(current_time);
        slapi_attr_add_value(time_attr, time_value);
        slapi_value_free(&time_value);

        /* Add the attribute to the result returned. */
        rc = (*outfn)(context, time_attr, entry);
        attr_done(time_attr);
        return rc;

        /* Handle other computed attributes...
    } else if (slapi_attr_type_cmp("myCompAttr", type, 2) == 0) {
    */
}
```

```
        return rc;
    }
```

Notice the lines of code that add the attribute to the result that is returned. The output function in this case is of type `slapi_compute_output_t`.

Notice also, when constructing the computed attribute value, that the `compute_attrs()` function removes the newline character from the string in the `ctime_r()` function. This removal prevents the attribute value from appearing base64-encoded when the value is displayed by the `ldapsearch` command.

Testing the Computed Attribute Plug-In

Before you can test the plug-in with the `ldapsearch` command, you must register the plug-in with Directory Server.

▼ To Register the Plug-In

If you have not already done so, build the example plug-in library and activate both plug-in informational logging and the example plug-in.

1 Build the plug-in.

Hint Use *install-path/examples/Makefile* or *install-path/examples/Makefile64*.

2 Configure Directory Server to log plug-in informational messages and load the plug-in.

Hint Use the commands specified in the comments at the outset of the plug-in source file.

3 Restart Directory Server.

```
$ dsadm restart instance-path
```

▼ To use the Computed Attribute Plug-In

Before You Begin Register the computed attribute plug-in with Directory Server.

● Run a search on the root DSE.

```
$ ldapsearch -h localhost -p 1389 -b "" -s base "(objectclass=*)" currentTime
version: 1
dn:
currentTime: Fri Feb 17 09:30:28 2006
```



PART II

Directory Server Plug-In API Reference

The Directory Server plug-in API allows you to develop *plug-ins*, libraries registered with a Directory Server instance that customize and extend server capabilities. This part provides a complete reference to the API.

This part includes the following chapters:

- [Chapter 15](#) covers key data structures, not including the server parameter block
- [Chapter 16](#) covers plug-in API functions, both by functional area and also in alphabetic order
- [Chapter 17](#) continues coverage of the plug-in API functions, in alphabetic order
- [Chapter 18](#) covers the server parameter block holding data pertaining to a particular operation

For help developing Directory Server plug-ins, read [Part I](#).

Data Type and Structure Reference

This chapter covers data types and structures for use in plug-in functions. [Table 15–1](#) summarizes the list of available data structures. [Table 15–2](#) summarizes the list of callbacks.

Quick Reference Tables

TABLE 15–1 Quick Reference to Data Structures

Data Structure or Callback	Short Description
“berval” on page 249	Binary data encoded using Basic Encoding Rules
“computed_attr_context” on page 249	Information for use when handling computed attributes
“LDAPControl” on page 250	LDAP v3 control associated with an LDAP operation
“LDAPMod” on page 250	Set of modifications to a directory entry attribute
“Slapi_Attr” on page 259	Directory entry attribute
“Slapi_Backend” on page 259	Server backend
“Slapi_ComponentId” on page 260	Server assigned identifier, used for internal operations
“Slapi_CondVar” on page 262	Condition variable for thread synchronization
“Slapi_Connection” on page 263	Client connection
“Slapi_DN” on page 263	Distinguished Name
“Slapi_Entry” on page 263	Directory entry
“Slapi_Filter” on page 265	Search filter
“Slapi_MatchingRuleEntry” on page 265	Matching rule handled by the plug-in

TABLE 15-1 Quick Reference to Data Structures *(Continued)*

Data Structure or Callback	Short Description
"Slapi_Mod" on page 265	A modification to an individual attribute
"Slapi_Mods" on page 266	Set of modifications to a directory entry
"Slapi_Mutex" on page 266	Mutex for thread synchronization
"Slapi_Operation" on page 267	Pending LDAP operation
"Slapi_PBlock" on page 267	Parameter block containing LDAP operation data
"Slapi_PluginDesc" on page 268	Plug-in description that you provide
"Slapi_RDN" on page 269	Relative Distinguished Name
"Slapi_Value" on page 270	Individual attribute value
"Slapi_ValueSet" on page 270	Set of values of an attribute
"vattr_type_thang" on page 270	Attribute that may be virtual

TABLE 15-2 Quick Reference to Callbacks

Data Structure or Callback	Short Description
"mrFilterMatchFn" on page 252	Handles an extensible match filter
"plugin_referral_entry_callback" on page 253	Handles referrals found by internal search
"plugin_result_callback" on page 254	Handles results sent after internal search
"plugin_search_entry_callback" on page 255	Handles entries found by internal search
"roles_get_scope_fn_type" on page 256	Determines scope of a role
"send_ldap_referral_fn_ptr_t" on page 256	Modifies referrals before sending them to a client
"send_ldap_result_fn_ptr_t" on page 257	Modifies a result before sending it to a client
"send_ldap_search_entry_fn_ptr_t" on page 258	Modifies an entry before sending it to a client
"slapi_compute_callback_t" on page 261	Handles a computed attribute
"slapi_compute_output_t" on page 262	Calculates a computed attribute
"slapi_extension_constructor_fnptr" on page 264	Handles object extension creation
"slapi_extension_destructor_fnptr" on page 264	Handles object extension destruction
"slapi_plugin_init_fnptr" on page 269	Handles recursive plug-in registration

Data Types and Structures Alphabetically

berval

Represents binary data encoded using simplified Basic Encoding Rules (BER).

Use a `berval` structure when working with binary attribute values such as a JPEG or audio file.

Syntax

```
/* Defined in ldap.h, which is included by slapi-plugin.h */
#include "slapi-plugin.h"
struct berval {
    unsigned long    bv_len;
    char             * bv_val;
};
```

Fields

This structure has the following fields.

TABLE 15-3 `berval` Fields

Field	Description
<code>bv_len</code>	Size of the encoded data.
<code>bv_val</code>	Encoded data itself.

computed_attr_context

An opaque structure representing information such as attribute type, current BER-encoded request and parameter block context for attributes that are computed.

Syntax

```
#include "slapi-plugin.h"
typedef struct _computed_attr_context computed_attr_context;
```

Description

Before the server sends an entry to a client application, it determines whether any of the attributes are computed. If so, it generates those attributes and includes them in the entry.

As part of this process, the server creates a `computed_attr_context` structure to pass relevant information to functions generating the attribute values.

LDAPControl

Represents a client or server control associated with an LDAP operation as specified by LDAP v3.

Client and server controls may be used to extend the functionality of an LDAP operation.

Syntax

```
/* Defined in ldap.h, which is included by slapi-plugin.h */
#include "slapi-plugin.h"
typedef struct ldapcontrol {
    char          * ldctl_oid;
    struct berval  ldctl_value;
    char          ldctl_iscritical;
} LDAPControl, * PLDAPControl;
```

Fields

This structure has the following fields.

TABLE 15-4 LDAPControl Fields

Field	Description
ldctl_oid	Object identifier (OID) for the control.
ldctl_value	“berval” on page 249 structure containing the value used by the control for the operation.
ldctl_iscritical	LDAP_OPT_ON indicates the control is critical to the operation. If the control is critical to the operation but not supported, the server returns LDAP_UNAVAILABLE_CRITICAL_EXTENSION. LDAP_OPT_OFF indicates the control is not critical.

LDAPMod

Represents a modification to an attribute in a directory entry.

Syntax

```
/* Defined in ldap.h, which is included by slapi-plugin.h */
#include "slapi-plugin.h"
typedef struct ldapmod {
    int          mod_op;
    char          * mod_type;
    union {
```

```
        char          ** modv_strvals;
        struct berval ** modv_bvals;
    } mod_vals;
} LDAPMod;
#define mod_values  mod_vals.modv_strvals
#define mod_bvalues mod_vals.modv_bvals
```

Fields

This structure has the following fields.

TABLE 15-5 LDAPMod Fields

Field	Description
mod_op	Operation to perform and data type of attribute values. The mod_op field takes the following values specifying the operation to perform. <ul style="list-style-type: none">■ LDAP_MOD_ADD to add the attribute values to the entry.■ LDAP_MOD_DELETE to remove the attribute values from the entry.■ LDAP_MOD_REPLACE to replace existing attribute values. To specify binary values, bitwise OR the macro LDAP_MOD_BVALUES with the operation type identifier. mod->mod_op = LDAP_MOD_ADD LDAP_MOD_BVALUES;
mod_type	Attribute type to modify.
mod_values	Pointer to NULL terminated array of string values for the attribute.
mod_bvalues	Pointer to NULL terminated array of “berval” on page 249 structures for the attribute.

Examples

Example 15-1 sets up an LDAPMod to change an mail address.

EXAMPLE 15-1 Preparing Modifications to an Entry

```
#include "slapi-plugin.h"
/* Declare the appropriate structures. */
LDAPMod      mod_attr;          /* Attribute to modify */
LDAPMod      * mods[2];         /* Array of modifications */
char         * mail_vals[] =    /* New mail address */
                {"quentin@example.com", NULL};

/* Set up the LDAPMod structure used to modify the entry. */
mod_attr.mod_type = "mail";
mod_attr.mod_op   = LDAP_MOD_REPLACE;
```

EXAMPLE 15-1 Preparing Modifications to an Entry (Continued)

```
mod_attr.mod_values = mail_vals;    /* "quentin@example.com" */
mods[0]              = &mod_attr;
mods[1]              = NULL;

/* Modify the entry using slapi_modify_internal_set_pb()... */
```

[Example 15-2](#) optionally adds additional modifications to those present in the parameter block. This code might be part of a pre-operation modify plug-in function, for example.

EXAMPLE 15-2 Adding Further Modifications

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

/*
 * Set up an LDAPMod array, modify_mods, of additional modifications.
 */

if (modify_mods != NULL) {
    LDAPMod ** mods;
    Slapi_Mods smods;
    int        i;

    slapi_pblock_get(pb, SLAPI_MODIFY_MODS, &mods);
    slapi_mods_init_passin(&smods, mods);

    for (i = 0; modify_mods[i] != NULL; i++) {
        /* Do not copy mods. */
        slapi_mods_add_ldapmod(&smods, modify_mods[i]);
    }

    mods = slapi_mods_get_ldapmods_passout(&smods);
    slapi_pblock_set(pb, SLAPI_MODIFY_MODS, mods);
    slapi_mods_done(&smods);

    /* Release container only. Content is still pointed to by mods. */
    slapi_ch_free((void **)&modify_mods);
}
```

mrFilterMatchFn

Specifies a filter matching callback function. The server calls this function when processing a search operation, once for each candidate that matches an extensible match search filter.

Syntax

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

Parameters

The callback takes the following parameters.

TABLE 15-6 mrFilterMatchFn Parameters

Parameter	Description
filter	Filter created by your filter factory function.
entry	“Slapi_Entry” on page 263 representing the candidate entry checked by the server.
vals	“Slapi_Attr” on page 259 representing the first attribute in the entry. Call “slapi_entry_next_attr()” on page 376 to iterate through the rest of the attributes.

Description

The server calls this filter matching function when processing an extensible match filter using a matching rule plug-in. 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 could be implemented to find all entries that sound like a given value.

To handle the extensible match filter for a matching rule, implement both this callback and a filter factor that creates the filter structure, `filter`. The callback retrieves information about the filter from this structure, such as the attribute type and value, then compares this information with attribute types and values in the candidate `entry`.

Returns

The callback must return 0 if the filter matches, -1 if the filter does not match. On error, it may return an LDAP error code as specified in the *Result Message* section of RFC 4511, *Lightweight Directory Access Protocol (v3)*.

plugin_referral_entry_callback

Specifies a referral callback function. The server calls this function when the internal search implemented to trigger this callback finds LDAP v3 referrals.

Syntax

```
#include "slapi-plugin.h"
typedef int (*plugin_referral_entry_callback)(char * referral,
      void *callback_data);
```

Parameters

The callback takes the following parameters.

TABLE 15-7 plugin_referral_entry_callback Parameters

Parameter	Description
referral	URL of a reference found by the search.
callback_data	Pointer passed to the internal search operation. Use this to pass your own data between the callback function and the body of the plug-in.

Description

Pass this as the prec parameter of “[slapi_search_internal_callback_pb\(\)](#)” on page 518. Each time the internal search finds a referral entry, it calls this function.

Returns

The server does not use the callback return value.

plugin_result_callback

Specifies a search result callback function. The server calls this function when the internal search implemented to trigger this callback returns an LDAP result.

Syntax

```
#include "slapi-plugin.h"
typedef void (*plugin_result_callback)(int rc, void *callback_data);
```

Parameters

The callback takes the following parameters.

TABLE 15-8 plugin_result_callback Parameters

Parameter	Description
rc	LDAP result code returned by the search.
callback_data	Pointer passed to the internal search operation. Use this to pass your own data between the callback function and the body of the plug-in.

Description

Pass this as the prc parameter of “[slapi_search_internal_callback_pb\(\)](#)” on page 518. The server calls this function once for each search operation, unless the search is abandoned, in which case the function is not called.

plugin_search_entry_callback

Specifies an entry callback function. The server calls this function when the internal search implemented to trigger this callback finds an LDAP entry.

Syntax

```
#include "slapi-plugin.h"
typedef int (*plugin_search_entry_callback)(Slapi_Entry *e,
void *callback_data);
```

Parameters

The callback takes the following parameters.

TABLE 15-9 plugin_search_entry_callback Parameters

Parameter	Description
e	Pointer to the entry found by the search.
callback_data	Pointer passed to the internal search operation. Use this to pass your own data between the callback function and the body of the plug-in.

Description

Pass this as the psec parameter of “[slapi_search_internal_callback_pb\(\)](#)” on page 518. Each time the internal search finds a referral entry, it calls this function.

Returns

The callback must return 0 to continue the search, -1 to interrupt the search.

roles_get_scope_fn_type

Specifies a callback function to determine the role of a scope. The plug-in triggers this callback using “[slapi_role_get_scope\(\)](#)” on page 495.

Syntax

```
#include "slapi-plugin.h"
typedef int (*roles_get_scope_fn_type)(Slapi_Entry *role_entry,
    Slapi_DN ***scope, int *nb_scope);
```

Parameters

The callback takes the following parameters.

TABLE 15–10 roles_get_scope_fn_type Parameters

Parameter	Description
role_entry	Entry defining the role
scope	Set this to the Distinguished Name of the entry at the base of the scope
nb_scope	Set this to a value such as LDAP_SCOPE_BASE, LDAP_SCOPE_ONELEVEL , or LDAP_SCOPE_SUBTREE

Description

This callback determines the role of a scope identified by role_entry. Register the callback with the server using “[slapi_register_role_get_scope\(\)](#)” on page 490 .

Returns

The callback returns 0 if successful, -1 otherwise.

send_ldap_referral_fn_ptr_t

Specifies a callback triggered before the server sends a result to the client.

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

The callback takes the following parameters.

TABLE 15-11 send_ldap_referral_fn_ptr_t Parameters

Parameter	Description
pb	Current parameter block for the operation.
e	Current entry for the operation.
refs	Pointer to the NULL terminated array of “ berval ” on page 249 structures containing the LDAP v3 referrals (search result references) found in the entry.
urls	Pointer to the array of “ berval ” on page 249 structures used to collect LDAP referrals for LDAP v2 clients.

Description

This callback lets you modify referrals returned to the client. Register the callback with the server using “[slapi_search_internal_callback_pb\(\)](#)” on page 518.

Returns

The callback should return 0 if successful, -1 otherwise.

send_ldap_result_fn_ptr_t

Specifies a callback triggered before the server sends a result to the client.

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 callback takes the following parameters.

TABLE 15-12 send_ldap_result_fn_ptr_t Parameters

Parameter	Description
pb	Current parameter block for the operation.
err	Result code.
matched	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.)
text	Error message that you want sent back to the client. (Pass NULL if you do not want an error message sent back.)
nentries	When sending back the result code for an LDAP search operation, use this argument to specify the number of matching entries found.
urls	When sending back an LDAP_PARTIAL_RESULTS result code to an LDAP v2 client or an LDAP_REFERRAL result code to an LDAP v3 client, use this argument to specify the array of “ berval ” on page 249 structures containing the referral URLs. (Pass NULL in other situations.)

Description

This callback lets you modify the result returned to the client. Register the callback with the server using “[slapi_search_internal_callback_pb\(\)](#)” on [page 518](#).

Returns

The callback should return 0 if successful, -1 otherwise.

send_ldap_search_entry_fn_ptr_t

Specifies a callback triggered before the server sends an entry returned by a search to the client.

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 );
```

Parameters

The callback takes the following parameters.

TABLE 15–13 send_ldap_search_entry_fn_ptr_t Parameters

Parameter	Description
pb	Current parameter block for the operation.
e	Entry returned by the search.
ectrls	Array of controls for the operation.
attrs	Array of attribute types to return in the search results.
attronly	<ul style="list-style-type: none"> ■ 0 to return both values and types ■ 1 to return only attribute types

Description

This callback lets you modify what is returned to the client. Register the callback with the server using [“slapi_search_internal_callback_pb\(\)” on page 518](#).

Returns

The callback should return 0 if successful, -1 otherwise.

Slapi_Attr

Opaque structure representing a directory entry attribute.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_attr Slapi_Attr;
```

See Also

[“Slapi_Entry” on page 263](#)

Slapi_Backend

Opaque structure representing a server backend.

Syntax

```
#include "slapi-plugin.h"
typedef struct backend Slapi_Backend;
```

See Also

[Table 16–18](#)

Slapi_ComponentId

Opaque structure representing a component identifier used by the server to identify the plug-in when processing internal operations.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_componentid Slapi_ComponentId;
```

Example

The following code excerpt sets a plug-in component identifier used during an internal search.

EXAMPLE 15-3 Setting a Plug-In Identifier for Use with Internal Operations

```
#include "slapi-plugin.h"
/* Declare the identifier as global to the plug-in. */
static Slapi_ComponentId * postop_id;

/* Assign a value as part of plug-in initialization. */
int
testpostop_init(Slapi_PBlock * pb)
{
    int rc = 0;
    /* Register description, other functions, and so forth. */
    rc |= slapi_pblock_set(
        pb,
        SLAPI_PLUGIN_START_FN,
        (void *) testpostop_set_log
    );
    /* Assign a value to the identifier. */
    rc |= slapi_pblock_get(pb, SLAPI_PLUGIN_IDENTITY, &postop_id);
    return (rc);
}

/* The server uses the identifier when processing
 * internal operations, such as an internal search. */
int
testpostop_set_log(Slapi_PBlock * pb)
{
    Slapi_DN * confdn = NULL; /* DN for configuration entry */
    Slapi_Entry * config = NULL; /* Configuration entry */
    int rc = 0;

    confdn = slapi_sdn_new_dn_byval("cn=config");
    rc |= slapi_search_internal_get_entry(confdn, NULL, &config, postop_id);
    /* Use the results of the search. */
}
```

EXAMPLE 15-3 Setting a Plug-In Identifier for Use with Internal Operations (Continued)

```
        return(rc);
    }
```

slapi_compute_callback_t

Specifies a function to determine which function should be called to provide a value for a computed attribute.

Syntax

```
typedef int (*slapi_compute_callback_t)(computed_attr_context *c,
    char* type, Slapi_Entry *e, slapi_compute_output_t outputfn);
```

Parameters

The callback takes the following parameters.

TABLE 15-14 slapi_compute_callback_t Parameters

Field	Description
c	Context of the callback.
type	Attribute type for which the output fn computes values.
e	Entry including computed attributes, returned to client after this callback returns.
outputfn	Callback to compute the attribute value.

Description

This callback selects the function that provides a computed value for an attribute of type `type` on an entry `e`, given context `c`. The server calls this function to get a a function for calculating attribute values before returning the entry `e` to the client having requested the operation.

This function is registered with the server using “[slapi_compute_add_evaluator\(\)](#)” on [page 333](#) in the plug-in initialization function.

Returns

This function should return 0 on success. It should return -1 if does not handle the attribute type passed in `type`. Otherwise, it should return an LDAP error code.

slapi_compute_output_t

Specifies a callback function to determine the value of a computed attribute.

Syntax

```
typedef int (*slapi_compute_output_t)(computed_attr_context *c,  
    Slapi_Attr *a , Slapi_Entry *e);
```

Parameters

The callback takes the following parameters.

TABLE 15–15 slapi_compute_output_t Parameters

Parameter	Description
c	Context of the callback.
a	Attribute for which the callback computes values.
e	Entry including computed attributes, returned to client after this callback returns.

Description

This callback provides a computed value for attribute a of entry e, given context c. The `slapi_compute_callback_t` function you register using “[slapi_compute_add_evaluator\(\)](#)” [on page 333](#) calls this function to compute a value for a before returning the entry e to the server.

Returns

This function should return 0 on success. It should return -1 if does not handle the attribute type passed in a. Otherwise, it should return an LDAP error code.

Slapi_ConiVar

Opaque structure representing a condition variable used by the plug-in to handle thread synchronization.

Syntax

```
#include "slapi-plugin.h"  
typedef struct slapi_condvar Slapi_ConiVar;
```

See Also

[“slapi_destroy_condvar\(\)” on page 338](#)

[“slapi_new_condvar\(\)” on page 457](#)

[“slapi_notify_condvar\(\)” on page 459](#)

[“slapi_wait_condvar\(\)” on page 575](#)

Slapi_Connection

Opaque structure representing a connection to the server.

Syntax

```
#include "slapi-plugin.h"
typedef struct conn Slapi_Connection;
```

Slapi_DN

Opaque structure representing a Distinguished Name (DN). The structure retains the original DN and can also hold a normalized version after [“slapi_sdn_get_ndn\(\)” on page 500](#) is called.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_dn Slapi_DN;
```

See Also

[Table 16–7](#) and [Table 16–19](#)

Slapi_Entry

Opaque structure representing a directory entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_entry Slapi_Entry;
```

See Also

[“Slapi_Attr” on page 259](#)

[Table 16–8](#)

slapi_extension_constructor_fnptr

Specifies a callback function for an object extension.

Syntax

```
#include "slapi-plugin.h"
typedef void (*slapi_extension_constructor_fnptr)
    (void *object, void *parent);
```

Parameters

The callback takes the following parameters.

TABLE 15-16 slapi_extension_constructor_fnptr Parameters

Parameter	Description
object	Extended object
parent	Parent object for the extension

Description

This callback registers an object extension that can be retrieved using [“slapi_get_object_extension\(\)” on page 398](#).

Ensure that this callback only creates the object extension if it does not already exist.

The callback is registered with the server using [“slapi_register_object_extension\(\)” on page 488](#) as part of the actual plug-in initialization function.

Returns

This callback returns a pointer to the extension. Otherwise it returns NULL.

slapi_extension_destructor_fnptr

Specifies a callback function to free memory allocated for an object extension.

Syntax

```
#include "slapi-plugin.h"
typedef void (*slapi_extension_destructor_fnptr)(void *extension,
    void *object, void *parent);
```

Parameters

The callback takes the following parameters.

TABLE 15-17 `slapi_extension_destructor_fnptr` Parameters

Parameter	Description
<code>extension</code>	Object extension
<code>object</code>	Extended object
<code>parent</code>	Parent for the extension

Description

This callback releases memory allocated for an object extension. The function is registered with the server using “[slapi_register_object_extension\(\)](#)” on [page 488](#) in the plug-in initialization function.

Slapi_Filter

Opaque structure representing a search filter.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_filter Slapi_Filter;
```

See Also

[Table 16-10](#)

Slapi_MatchingRuleEntry

Opaque structure representing an LDAP v3 matching rule handled by the plug-in.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_matchingRuleEntry Slapi_MatchingRuleEntry;
```

See Also

[Table 16-12](#)

Slapi_Mod

Opaque structure representing a modification of a directory entry attribute.

Parameter blocks use “[LDAPMod](#)” on page 250 structures rather than `Slapi_Mod` structures. The latter type is provided as a convenience for plug-ins dealing extensively with modifications.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_mod Slapi_Mod;
```

See Also

“[LDAPMod](#)” on page 250

[Table 16–13](#)

Slapi_Mods

Opaque structure representing a set of modifications to a directory entry.

Parameter blocks use arrays of “[LDAPMod](#)” on page 250 structures rather than `Slapi_Mods` structures. The latter type is provided as a convenience for plug-ins dealing extensively with modifications.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_mods Slapi_Mods;
```

See Also

“[LDAPMod](#)” on page 250

[Table 16–13](#)

Slapi_Mutex

Opaque structure representing a mutex lock used by the plug-in.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_mutex Slapi_Mutex;
```

See Also

[“slapi_destroy_mutex\(\)” on page 338](#)

[“slapi_lock_mutex\(\)” on page 407](#)

[“slapi_new_mutex\(\)” on page 458](#)

[“slapi_UTF-8STRTOLOWER\(\)” on page 534](#)

Slapi_Operation

Opaque structure representing a pending operation from an LDAP client.

The structure records, among other data, the type of operation requested.

Syntax

```
#include "slapi-plugin.h"
typedef struct op Slapi_Operation;
```

See Also

[“slapi_op_abandoned\(\)” on page 460](#)

[“slapi_op_get_type\(\)” on page 460](#)

Slapi_PBlock

Opaque structure representing, called a parameter block, containing name-value pairs updated in the context of a server operation.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_pblock Slapi_PBlock;
```

Description

For most types of plug-in functions, the server passes in a parameter block (Slapi_PBlock) including data relevant to the operation being processed. Access the data using [“slapi_pblock_get\(\)” on page 462](#).

Plug-in initialization functions register at least the plug-in API version, plug-in descriptions, and other plug-in functions using [“slapi_pblock_set\(\)” on page 464](#).

The specific parameters available in a `Slapi_PBlock` structure depend on the type of plug-in function and the context of the LDAP operation. Refer to the “Parameter Block Reference,” on page 335 for details on the name-value pairs available to specific types of plug-in functions.

See Also

[Table 16–1](#)

[Chapter 15](#)

For examples of `Slapi_PBlock` use, refer to the sample plug-ins under `$INSTALL_DIR/examples/`.

Slapi_PluginDesc

Represents a plug-in description you provide to identify your plug-in.

The plug-in initialization function must register this information with the server.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_plugin_desc {
    char * spd_id;
    char * spd_vendor;
    char * spd_version;
    char * spd_description;
} Slapi_PluginDesc;
```

Fields

This structure has the following fields.

TABLE 15–18 Slapi_PluginDesc Fields

Field	Description
spd_id	Unique (server wide) identifier for the plug-in.
spd_vendor	Name of the vendor supplying the plug-in such as Sun Microsystems, Inc.
spd_version	Plug-in revision number such as 5.2, not to be confused with <code>SLAPI_PLUGIN_VERSION</code> , which specifies the plug-in API version supported by the plug-in.
spd_description	Short description of the plug-in such as Sample post-operation plug-in.

See Also

For examples of `Slapi_PluginDesc` use, refer to the sample plug-ins under `$INSTALL_DIR/examples/`.

slapi_plugin_init_fnptr

Specifies a callback function for registering other plug-ins.

Syntax

```
#include "slapi-plugin.h"
typedef int (*slapi_plugin_init_fnptr)( Slapi_PBlock *pb );
```

Parameters

The callback takes the following parameter.

TABLE 15-19 `slapi_plugin_init_fnptr` Parameter

Parameter	Description
pb	Parameter block passed to the initialization function.

Description

This callback mimics a plug-in initialization function, permitting one plug-in to register other plug-ins. The function is registered with the server using “[slapi_register_plugin\(\)](#)” on [page 489](#) as part of the actual plug-in initialization function.

Returns

This callback returns `0` on success. Otherwise, it returns `-1`.

See Also

For examples of plug-in initialization functions, refer to the sample plug-ins under `$INSTALL_DIR/examples/`.

Slapi_RDN

Opaque structure representing a Relative Distinguished Name (RDN), the part of the DN that differentiates the entry from other entries having the same parent.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_rdn Slapi_RDN;
```

See Also

[Table 16–20](#)

Slapi_Value

Opaque structure representing an individual attribute value.

Use “[Slapi_ValueSet](#)” on [page 270](#) instead when handling all the values of an attribute at once.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_value Slapi_Value;
```

See Also

[Table 16–21](#)

Slapi_ValueSet

Opaque structure representing the set of values of an attribute.

Use “[Slapi_Value](#)” on [page 270](#) instead when handling an individual attribute value.

Syntax

```
#include "slapi-plugin.h"
typedef struct slapi_value_set Slapi_ValueSet;
```

See Also

[Table 16–22](#)

vattr_type_thang

Opaque structure representing both real and virtual attributes of an entry.

Syntax

```
#include "slapi-plugin.h"
typedef struct _vattr_type_thang vattr_type_thang;
```

See Also

[Table 16–25](#)

Function Reference, Part I

This chapter contains the first part of the reference to the public functions for writing plug-ins. The following chapter contains the second part of the reference. This chapter includes the following sections:

- [“Functions by Functional Area” on page 273](#)
- [“Function Descriptions, Part I” on page 291](#)

In addition to a detailed description of each function, the function descriptions detail the function header file and syntax, the function parameters, the function return value, and possible memory concerns.

Functions by Functional Area

This section categorizes plug-in functions by functional area.

TABLE 16-1 Functions for Handling Parameter Blocks

Function	Description
“slapi_pblock_destroy()” on page 461	Frees a parameter block from memory.
“slapi_pblock_get()” on page 462	Gets the value from a parameter block.
“slapi_pblock_new()” on page 464	Creates a new parameter block.
“slapi_pblock_set()” on page 464	Sets the value of a parameter block.

TABLE 16-2 Functions for Handling Memory

Function	Description
“slapi_ch_array_free()” on page 326	Frees an existing array.

TABLE 16-2 Functions for Handling Memory (Continued)

Function	Description
“slapi_ch_bvdup()” on page 327	Makes a copy of an existing “berval” on page 249 structure.
“slapi_ch_bvecdup()” on page 328	Makes a copy of an array of existing “berval” on page 249 structures.
“slapi_ch_calloc()” on page 328	Allocates space for an array of a number of elements of a specified size.
“slapi_ch_free()” on page 329	Frees space allocated by the “slapi_ch_malloc()” on page 331, “slapi_ch_realloc()” on page 332, and “slapi_ch_calloc()” on page 328 functions.
“slapi_ch_free_string()” on page 330	Frees an existing string.
“slapi_ch_malloc()” on page 331	Allocates space in memory.
“slapi_ch_realloc()” on page 332	Changes the size of a block of allocated memory.
“slapi_ch_strdup()” on page 332	Makes a copy of an existing string.

TABLE 16-3 Functions for Handling Access Control

Function	Description
“slapi_access_allowed()” on page 292	Determines if the user requesting the current operation has the access rights to perform an operation on a given entry, attribute, or value.
“slapi_acl_check_mods()” on page 294	Determines if a user has the rights to perform the specified modifications on an entry.
“slapi_acl_verify_aci_syntax()” on page 296	Determines whether or not the access control items (ACIs) on an entry are valid.

TABLE 16-4 Functions for Handling Attributes

Function	Description
“slapi_attr_add_value()” on page 300	Adds a value to an attribute.
“slapi_attr_basetype()” on page 301	Returns the base type of an attribute.
“slapi_attr_dup()” on page 301	Duplicates an attribute.
“slapi_attr_first_value_const()” on page 302	Gets the first value of an attribute.
“slapi_attr_flag_is_set()” on page 303	Determines if certain flags are set.
“slapi_attr_free()” on page 304	Frees an attribute.

TABLE 16–4 Functions for Handling Attributes (Continued)

Function	Description
<code>“slapi_attr_get_bervals_copy()”</code> on page 305	Puts the values contained in an attribute into an array of <code>“berval”</code> on page 249 structures.
<code>“slapi_attr_get_flags()”</code> on page 305	Gets the flags associated with an attribute.
<code>“slapi_attr_get_numvalues()”</code> on page 306	Puts the count of values of an attribute into an integer.
<code>“slapi_attr_get_oid_copy()”</code> on page 307	Searches for an attribute type and gives its OID string.
<code>“slapi_attr_get_type()”</code> on page 308	Gets the name of the attribute type.
<code>“slapi_attr_get_valueset()”</code> on page 309	Copies attribute values into a value set.
<code>“slapi_attr_init()”</code> on page 310	Initializes an empty attribute.
<code>“slapi_attr_new()”</code> on page 311	Creates a new attribute.
<code>“slapi_attr_next_value_const()”</code> on page 311	Gets the next value of an attribute.
<code>“slapi_attr_syntax_normalize()”</code> on page 312	Returns a copy of the normalized attribute types.
<code>“slapi_attr_type_cmp()”</code> on page 313	Compares two attributes.
<code>“slapi_attr_types_equivalent()”</code> on page 314	Compares two attribute names to determine if they represent the same attribute.
<code>“slapi_attr_value_cmp()”</code> on page 315	Compares two attribute values.
<code>“slapi_attr_value_find()”</code> on page 315	Determines if an attribute contains a given value.

TABLE 16–5 Functions for Handling Basic Encoding Rule Values

Function	Description
<code>“slapi_berval_cmp()”</code> on page 323	Compares two <code>“berval”</code> on page 249 structures.
<code>“slapi_ch_bvdup()”</code> on page 327	Makes a copy of an existing <code>“berval”</code> on page 249 structure.
<code>“slapi_ch_bvecdup()”</code> on page 328	Makes a copy of an array of existing <code>“berval”</code> on page 249 structures.

TABLE 16–6 Functions for Handling Controls

Function	Description
<code>“slapi_build_control()”</code> on page 324	Creates an <code>“LDAPControl”</code> on page 250 structure based on a <code>BeRElement</code> , an OID, and a criticality flag.

TABLE 16-6 Functions for Handling Controls *(Continued)*

Function	Description
“slapi_entry_get_uniqueid()” on page 372	Retrieves an allocated array of object identifiers (OIDs) representing the controls supported by Directory Server.
“slapi_control_present()” on page 335	Determines whether or not the specified object identification (OID) identifies a control that is present in a list of controls.
“slapi_dup_control()” on page 346	Makes an allocated copy of an “LDAPControl” on page 250 .
“slapi_register_supported_control()” on page 491	Registers the specified control with the server. This function associates the control with an object identification (OID).

TABLE 16-7 Functions for Handling Distinguished Name Strings

Function	Description
“slapi_dn_beparent()” on page 339	Gets a copy of the DN of the parent of an entry.
“slapi_dn_ignore_case()” on page 339	Converts all characters in a DN to lowercase.
“slapi_dn_isbesuffix()” on page 340	Determines if a DN is the suffix of the local database.
“slapi_dn_isbesuffix_norm()” on page 341	Determines if a DN is the suffix of the local database.
“slapi_dn_isparent()” on page 341	Determines if a DN is the parent of a specific DN.
“slapi_dn_isroot()” on page 342	Determines if a DN is the root DN for the local database.
“slapi_dn_issuffix()” on page 342	Determines if a DN is equal to a specified suffix.
“slapi_dn_normalize()” on page 343	Converts a DN to canonical format.
“slapi_dn_normalize_case()” on page 344	Converts a DN to canonical format and all characters to lower case.
“slapi_dn_normalize_to_end()” on page 344	Normalizes part of a DN value.
“slapi_dn_parent()” on page 345	Gets the DN of the parent of an entry.
“slapi_dn_plus_rdn()” on page 346	Adds an RDN to a DN.

TABLE 16-8 Functions for Handling Entries

Function	Description
“slapi_entry2str()” on page 348	Generates an LDIF string description.

TABLE 16–8 Functions for Handling Entries (Continued)

Function	Description
“slapi_entry2str_with_options()” on page 349	Generates an LDIF string descriptions with options.
“slapi_entry_add_rdn_values()” on page 351	Add components in an entry’s RDN.
“slapi_entry_add_string()” on page 352	Adds a string value to an attribute in an entry.
“slapi_entry_add_values_sv()” on page 353	Adds a data value to an attribute in an entry.
“slapi_entry_add_valueset()” on page 354	Adds a data value to an attribute in an entry.
“slapi_entry_alloc()” on page 355	Allocates memory for a new entry.
“slapi_entry_attr_delete()” on page 356	Deletes an attribute from an entry.
“slapi_entry_attr_find()” on page 356	Checks if an entry contains a specific attribute.
“slapi_entry_attr_get_charptr()” on page 357	Gets the first value as a string.
“slapi_entry_attr_get_int()” on page 358	Gets the first value as an integer.
“slapi_entry_attr_get_long()” on page 358	Gets the first value as a long.
“slapi_entry_attr_get_uint()” on page 359	Gets the first value as an unsigned integer.
“slapi_entry_attr_get_ulong()” on page 359	Gets the first value as an unsigned long.
“slapi_entry_attr_hasvalue()” on page 360	Checks if an attribute in an entry contains a value.
“slapi_entry_attr_merge_sv()” on page 360	Adds an array to the attribute values in an entry.
“slapi_entry_attr_replace_sv()” on page 361	Replaces the attribute values in an entry.
“slapi_entry_attr_set_charptr()” on page 362	Replaces the values of an attribute with a string.
“slapi_entry_attr_set_int()” on page 363	Replaces the value of an attribute with an integer.
“slapi_entry_attr_set_long()” on page 363	Replaces the value of an attribute with a long.
“slapi_entry_attr_set_uint()” on page 364	Replaces the value of an attribute with an unsigned integer.
“slapi_entry_attr_set_ulong()” on page 364	Replaces the value of an attribute with an unsigned long.
“slapi_entry_delete_string()” on page 365	Deletes a string from an attribute.
“slapi_entry_delete_values_sv()” on page 365	Removes a “ Slapi_Value ” on page 270 array from an attribute.
“slapi_entry_dup()” on page 366	Copies an entry, its DN, and its attributes.
“slapi_entry_first_attr()” on page 367	Finds the first attribute in an entry.
“slapi_entry_free()” on page 368	Frees an entry from memory.

TABLE 16–8 Functions for Handling Entries (Continued)

Function	Description
<code>“slapi_entry_get_dn()”</code> on page 369	Gets the DN from an entry.
<code>“slapi_entry_get_dn_const()”</code> on page 369	Returns the DN of an entry as a constant.
<code>“slapi_entry_get_ndn()”</code> on page 370	Returns the NDN of an entry.
<code>“slapi_entry_get_sdn()”</code> on page 370	Returns the <code>“Slapi_DN”</code> on page 263 from an entry.
<code>“slapi_entry_get_sdn_const()”</code> on page 371	Returns a <code>“Slapi_DN”</code> on page 263 from an entry as a constant.
<code>“slapi_entry_get_uniqueid()”</code> on page 372	Gets the unique ID from an entry.
<code>“slapi_entry_has_children()”</code> on page 372	Determines if the specified entry has child entries.
<code>“slapi_entry_init()”</code> on page 373	Initializes the values of an entry.
<code>“slapi_entry_isroot()”</code> on page 374	Determines whether the entry is that of a directory super user.
<code>“slapi_entry_merge_values_sv()”</code> on page 375	Adds an array of data values to an attribute in an entry.
<code>“slapi_entry_next_attr()”</code> on page 376	Finds the next attribute in an entry.
<code>“slapi_entry_rdn_values_present()”</code> on page 376	Checks if values present in an entry’s RDN are also present as attribute values.
<code>“slapi_entry_schema_check()”</code> on page 377	Determines if an entry complies with the schema for its object class.
<code>“slapi_entry_schema_check_ext()”</code> on page 378	Determines if a set of modifications to an entry comply with the schema.
<code>“slapi_entry_set_dn()”</code> on page 378	Sets the DN of an entry.
<code>“slapi_entry_set_sdn()”</code> on page 379	Sets the <code>“Slapi_DN”</code> on page 263 value in an entry.
<code>“slapi_entry_size()”</code> on page 380	Returns the size of an entry.
<code>“slapi_entry_syntax_check()”</code> on page 381	Determines if the attributes of an entry comply with attribute syntax rules.
<code>“slapi_entry_vattr_find()”</code> on page 381	Finds the specified virtual attribute in the entry.
<code>“slapi_is_rootdse()”</code> on page 403	Determines if an entry is the root DSE.
<code>“slapi_str2entry()”</code> on page 528	Converts an LDIF description into an entry.

TABLE 16–9 Functions for Handling Extended Operations

Function	Description
<code>slapi_get_supported_extended_ops_copy()</code> on page 401	Gets a copy of the object IDs (OIDs) of the extended operations.

TABLE 16–10 Functions for Handling Filters

Function	Description
<code>slapi_filter_compare()</code> on page 382	Determines if two filters are identical.
<code>slapi_filter_free()</code> on page 383	Frees the specified filter.
<code>slapi_filter_get_attribute_type()</code> on page 384	Gets the attribute type for all simple filter choices.
<code>slapi_filter_get_ava()</code> on page 385	Gets the attribute type and the value from the filter.
<code>slapi_filter_get_choice()</code> on page 386	Gets the type of the specified filter.
<code>slapi_filter_get_subfilt()</code> on page 387	Gets the substring values from the filter.
<code>slapi_filter_get_type()</code> on page 388	Gets the attribute type specified in the filter.
<code>slapi_filter_join()</code> on page 389	Joins two specified filters.
<code>slapi_filter_list_first()</code> on page 390	Gets the first filter that makes up the specified filter.
<code>slapi_filter_list_next()</code> on page 391	Gets the next filter.
<code>slapi_filter_test()</code> on page 392	Determines if the specified entry matches a particular filter.
<code>slapi_filter_test_ext()</code> on page 393	Determines if an entry matches a given filter.
<code>slapi_filter_test_simple()</code> on page 394	Determines if an entry matches a filter.
<code>slapi_str2filter()</code> on page 529	Converts a string description of a search filter into a filter of the <code>Slapi_Filter</code> on page 265 type.

TABLE 16–11 Functions for Handling Internal Operations

Function	Description
<code>slapi_add_entry_internal_set_pb()</code> on page 297	Prepare a <code>Slapi_PBlock</code> on page 267 structure for an internal add operation involving a <code>Slapi_Entry</code> on page 263 structure.
<code>slapi_add_internal_pb()</code> on page 298	Adds an LDAP add operation based on a parameter block to add a new directory entry.
<code>slapi_add_internal_set_pb()</code> on page 299	Prepare a <code>Slapi_PBlock</code> on page 267 structure for an internal add operation.

TABLE 16–11 Functions for Handling Internal Operations (Continued)

Function	Description
“slapi_delete_internal_pb()” on page 336	Performs an LDAP delete operation based on a parameter block to remove a directory entry
“slapi_delete_internal_set_pb()” on page 337	Prepare a “Slapi_PBlock” on page 267 structure for an internal delete operation.
“slapi_free_search_results_internal()” on page 396	Frees search results.
“slapi_modify_internal_pb()” on page 431	Performs an LDAP modify operation based on a parameter block to modify a directory entry.
“slapi_modify_internal_set_pb()” on page 432	Prepare a “Slapi_PBlock” on page 267 structure for an internal modify operation.
“slapi_modrdn_internal_pb()” on page 433	Performs an LDAP modify RDN operation based on a parameter block to rename a directory entry.
“slapi_rename_internal_set_pb()” on page 493	Prepare a “Slapi_PBlock” on page 267 structure for an internal modify RDN operation.
“slapi_search_internal_callback_pb()” on page 518	Performs an LDAP search operation based on a parameter block to search the directory.
“slapi_search_internal_get_entry()” on page 520	Performs an internal search operation to read one entry
“slapi_search_internal_pb()” on page 521	Performs an LDAP search operation based on a parameter block to search the directory.
“slapi_search_internal_set_pb()” on page 521	Prepare a “Slapi_PBlock” on page 267 structure for an internal search operation.

TABLE 16–12 Functions for Handling Matching Rules

Function	Description
“slapi_matchingrule_free()” on page 413	Free a “Slapi_MatchingRuleEntry” on page 265 after registering the matching rule.
“slapi_matchingrule_get()” on page 414	Access a “Slapi_MatchingRuleEntry” on page 265 .
“slapi_matchingrule_new()” on page 415	Allocate a “Slapi_MatchingRuleEntry” on page 265 structure.
“slapi_matchingrule_register()” on page 415	Register a matching rule with the server.
“slapi_matchingrule_set()” on page 416	Modify a “Slapi_MatchingRuleEntry” on page 265 .
“slapi_mr_filter_index()” on page 456	Call a matching rule filter index function.

TABLE 16–12 Functions for Handling Matching Rules (Continued)

Function	Description
“slapi_mr_indexer_create()” on page 456	Get a pointer to the indexer factory function for a matching rule.

TABLE 16–13 Functions for Handling Modifications

Function	Description
“slapi_entry2mods()” on page 347	Creates an array of “LDAPMod” on page 250 from a “Slapi_Entry” on page 263 .
“slapi_ldapmods_syntax_check()” on page 406	Determines if the proposed modifications to an entry comply with attribute syntax rules.
“slapi_mod_add_value()” on page 417	Adds a value to a “Slapi_Mod” on page 265 structure.
“slapi_mod_done()” on page 418	Frees internals of “Slapi_Mod” on page 265 structure.
“slapi_mod_dump()” on page 418	Dumps the contents of an “LDAPMod” on page 250 to the server log.
“slapi_mod_free()” on page 419	Frees a “Slapi_Mod” on page 265 structure.
“slapi_mod_get_first_value()” on page 419	Initializes a “Slapi_Mod” on page 265 iterator and returns the first attribute value.
“slapi_mod_get_ldapmod_byref()” on page 420	Gets a reference to the “LDAPMod” on page 250 in a “Slapi_Mod” on page 265 structure.
“slapi_mod_get_ldapmod_passout()” on page 421	Retrieves the “LDAPMod” on page 250 contained in a “Slapi_Mod” on page 265 structure.
“slapi_mod_get_next_value()” on page 421	Increments the “Slapi_Mod” on page 265 iterator and returns the next attribute value.
“slapi_mod_get_num_values()” on page 422	Gets the number of values in a “Slapi_Mod” on page 265 structure.
“slapi_mod_get_operation()” on page 423	Gets the operation type of “Slapi_Mod” on page 265 structure.
“slapi_mod_get_type()” on page 423	Gets the attribute type of a “Slapi_Mod” on page 265 structure.
“slapi_mod_init()” on page 424	Initializes a “Slapi_Mod” on page 265 structure.
“slapi_mod_init_byref()” on page 425	Initializes a “Slapi_Mod” on page 265 structure that is a wrapper for an existing “LDAPMod” on page 250 .
“slapi_mod_init_byval()” on page 425	Initializes a “Slapi_Mod” on page 265 structure with a copy of an “LDAPMod” on page 250 .

TABLE 16–13 Functions for Handling Modifications *(Continued)*

Function	Description
<code>"slapi_mod_init_passin()" on page 426</code>	Initializes a "Slapi_Mod" on page 265 from an "LDAPMod" on page 250.
<code>"slapi_mod_isvalid()" on page 427</code>	Determines whether a "Slapi_Mod" on page 265 structure is valid.
<code>"slapi_mod_new()" on page 427</code>	Allocates a new "Slapi_Mod" on page 265 structure.
<code>"slapi_mod_remove_value()" on page 428</code>	Removes the value at the current "Slapi_Mod" on page 265 iterator position.
<code>"slapi_mod_set_operation()" on page 429</code>	Sets the operation type of a "Slapi_Mod" on page 265 structure.
<code>"slapi_mod_set_type()" on page 429</code>	Sets the attribute type of a "Slapi_Mod" on page 265.
<code>"slapi_mods2entry()" on page 434</code>	Creates a "Slapi_Entry" on page 263 from an array of "LDAPMod" on page 250.
<code>"slapi_mods_add()" on page 435</code>	Appends a new mod with a single attribute value to "Slapi_Mods" on page 266 structure.
<code>"slapi_mods_add_ldapmod()" on page 436</code>	Appends an "LDAPMod" on page 250 to a "Slapi_Mods" on page 266 structure.
<code>"slapi_mods_add_mod_values()" on page 437</code>	Appends a new mod to a "Slapi_Mods" on page 266 structure, with attribute values provided as an array of "Slapi_Value" on page 270.
<code>"slapi_mods_add_modbvps()" on page 438</code>	Appends a new mod to a "Slapi_Mods" on page 266 structure, with attribute values provided as an array of "berval" on page 249.
<code>"slapi_mods_add_smod()" on page 439</code>	Appends a "Slapi_Mod" on page 265 to a "Slapi_Mods" on page 266 structure.
<code>"slapi_mods_add_string()" on page 439</code>	Appends a new mod to "Slapi_Mods" on page 266 structure with a single attribute value provided as a string.
<code>"slapi_mods_done()" on page 440</code>	Frees internals of a "Slapi_Mods" on page 266 structure.
<code>"slapi_mods_dump()" on page 441</code>	Dumps the contents of a "Slapi_Mods" on page 266 structure to the server log.
<code>"slapi_mods_free()" on page 442</code>	Frees a "Slapi_Mods" on page 266 structure.
<code>"slapi_mods_get_first_mod()" on page 442</code>	Initializes a "Slapi_Mods" on page 266 iterator and returns the first "LDAPMod" on page 250.

TABLE 16–13 Functions for Handling Modifications *(Continued)*

Function	Description
“slapi_mods_get_first_smod()” on page 443	Initializes a “Slapi_Mods” on page 266 iterator and returns the first mod wrapped in a “Slapi_Mods” on page 266 structure.
“slapi_mods_get_ldapmods_byref()” on page 443	Gets a reference to the array of “LDAPMod” on page 250 in a “Slapi_Mods” on page 266 structure.
“slapi_mods_get_ldapmods_passout()” on page 444	Retrieves the array of “LDAPMod” on page 250 contained in a “Slapi_Mods” on page 266 structure.
“slapi_mods_get_next_mod()” on page 445	Increments the “Slapi_Mods” on page 266 iterator and returns the next “LDAPMod” on page 250.
“slapi_mods_get_next_smod()” on page 446	Increments the “Slapi_Mods” on page 266 iterator and returns the next mod wrapped in a “Slapi_Mods” on page 266.
“slapi_mods_get_num_mods()” on page 447	Gets the number of mods in a “Slapi_Mods” on page 266 structure.
“slapi_mods_init()” on page 447	Initializes a “Slapi_Mods” on page 266.
“slapi_mods_init_byref()” on page 448	Initializes a “Slapi_Mods” on page 266 that is a wrapper for an existing array of “LDAPMod” on page 250.
“slapi_mods_init_passin()” on page 448	Initializes a “Slapi_Mods” on page 266 structure from an array of “LDAPMod” on page 250.
“slapi_mods_insert_after()” on page 449	Inserts an “LDAPMod” on page 250 into a “Slapi_Mods” on page 266 structure after the current iterator position.
“slapi_mods_insert_at()” on page 450	Inserts an “LDAPMod” on page 250 anywhere in a “Slapi_Mods” on page 266.
“slapi_mods_insert_before()” on page 451	Inserts an “LDAPMod” on page 250 into a “Slapi_Mods” on page 266 structure before the current iterator position.
“slapi_mods_insert_smod_at()” on page 452	Inserts a “Slapi_Mod” on page 265 anywhere in a “Slapi_Mods” on page 266.
“slapi_mods_insert_smod_before()” on page 452	Inserts a “Slapi_Mod” on page 265 into a “Slapi_Mods” on page 266 structure before the current iterator position.
“slapi_mods_iterator_backone()” on page 453	Decrements the “Slapi_Mods” on page 266 current iterator position.

TABLE 16–13 Functions for Handling Modifications *(Continued)*

Function	Description
“slapi_mods_new()” on page 454	Allocates a new uninitialized “Slapi_Mods” on page 266 structure.
“slapi_mods_remove()” on page 455	Removes the mod at the current “Slapi_Mods” on page 266 iterator position.
“slapi_mods_remove_at()” on page 455	Removes the mod at the specified “Slapi_Mods” on page 266 iterator position.

TABLE 16–14 Functions for Handling Operations

Function	Description
“slapi_op_abandoned()” on page 460	Determines if the client has abandoned the current operation.
“slapi_op_get_type()” on page 460	Gets the type of a “Slapi_Operation” on page 267 .

TABLE 16–15 Functions for Handling Passwords

Function	Description
“slapi_pw_find_sv()” on page 466	Determines whether or not a specified password matches one of the hashed values of an attribute.
“slapi_pw_find_valueset()” on page 467	Determines whether or not a specified password matches one of the hashed values of an attribute.

TABLE 16–16 Functions for Handling Roles

Function	Description
“slapi_register_role_get_scope()” on page 490	Register a callback to determine the scope of a role.
“slapi_role_check()” on page 494	Checks if the entry pointed to contains the role indicated.
“slapi_role_get_scope()” on page 495	Determine the scope of a role.

TABLE 16–17 Functions for Handling SASL Mechanisms

Function	Description
“slapi_get_supported_saslmechanisms_copy()” on page 402	Gets an array of the names of the supported Simple Authentication and Security Layer (SASL) mechanisms.
“slapi_register_supported_saslmechanism()” on page 492	Registers the specified Simple Authentication and Security Layer (SASL) mechanism with the server.

TABLE 16–18 Functions for Handling Slapi_Backend Structures

Function	Description
“slapi_be_exist()” on page 316	Checks if the backend that contains the specified DN exists.
“slapi_be_get_name()” on page 317	Returns the name of the specified backend.
“slapi_be_get_readonly()” on page 317	Indicates if the database associated with the backend is in read-only mode.
“slapi_be_getsuffix()” on page 318	Returns the n+1 suffix associated with the specified backend.
“slapi_be_gettype()” on page 319	Returns the type of the backend.
“slapi_be_is_flag_set()” on page 319	Checks if a flag is set in the backend configuration.
“slapi_be_issuffix()” on page 320	Verifies that the specified suffix matches a registered backend suffix.
“slapi_be_logchanges()” on page 321	Indicates if the changes applied to the backend should be logged in the change log.
“slapi_be_private()” on page 321	Verifies if the backend is private.
“slapi_be_select()” on page 322	Finds the backend that should be used to service the entry with the specified DN.
“slapi_be_select_by_instance_name()” on page 322	Find the backend used to service the database.
“slapi_get_first_backend()” on page 397	Returns a pointer of the backend structure of the first backend.
“slapi_get_next_backend()” on page 398	Returns a pointer to the next backend.
“slapi_is_root_suffix()” on page 403	Checks if a suffix is a root suffix of the DIT.

TABLE 16–19 Functions for Handling Slapi_DN Structures

Function	Description
“slapi_moddn_get_newdn()” on page 430	Builds the new DN of an entry.
“slapi_sdn_compare()” on page 496	Compares two DNs.
“slapi_sdn_copy()” on page 497	Copies a DN.
“slapi_sdn_done()” on page 497	Clears a “Slapi_DN” on page 263 structure.
“slapi_sdn_dup()” on page 498	Duplicates a “Slapi_DN” on page 263 structure.
“slapi_sdn_free()” on page 498	Frees a “Slapi_DN” on page 263 structure.

TABLE 16–19 Functions for Handling Slapi_DN Structures (Continued)

Function	Description
<code>“slapi_sdn_get_backend_parent()”</code> on page 499	Gets the DN of the parent within a specific backend.
<code>“slapi_sdn_get_dn()”</code> on page 500	Gets the DN from a “Slapi_DN” on page 263 structure.
<code>“slapi_sdn_get_ndn()”</code> on page 500	Gets the normalized DN of a “Slapi_DN” on page 263 structure.
<code>“slapi_sdn_get_ndn_len()”</code> on page 501	Gets the length of the normalized DN of a “Slapi_DN” on page 263 structure.
<code>“slapi_sdn_get_parent()”</code> on page 502	Get the parent DN of a given “Slapi_DN” on page 263 structure.
<code>“slapi_sdn_get_rdn()”</code> on page 502	Gets the RDN from a DN.
<code>“slapi_sdn_get_suffix()”</code> on page 503	Gets the suffix holding the entry specified by DN.
<code>“slapi_sdn_isempty()”</code> on page 504	Checks if there is a DN value stored in a “Slapi_DN” on page 263 structure.
<code>“slapi_sdn_isgrandparent()”</code> on page 504	Checks if a DN is the parent of the parent of a DN.
<code>“slapi_sdn_isparent()”</code> on page 505	Checks if a DN is the parent of a DN.
<code>“slapi_sdn_issuffix()”</code> on page 506	Checks if a “Slapi_DN” on page 263 structure contains a suffix of another.
<code>“slapi_sdn_new()”</code> on page 506	Allocates new “Slapi_DN” on page 263 structure.
<code>“slapi_sdn_new_dn_byref()”</code> on page 507	Creates a new “Slapi_DN” on page 263 structure pointing to an existing DN string.
<code>“slapi_sdn_new_dn_byval()”</code> on page 508	Creates a new “Slapi_DN” on page 263 structure copying an existing DN string.
<code>“slapi_sdn_new_dn_passin()”</code> on page 509	Creates a new “Slapi_DN” on page 263 structure pointing to a new copy of a DN string.
<code>“slapi_sdn_new_ndn_byref()”</code> on page 510	Creates a new “Slapi_DN” on page 263 structure pointing to an existing normalized DN.
<code>“slapi_sdn_new_ndn_byval()”</code> on page 511	Creates a new “Slapi_DN” on page 263 structure copying an existing normalized DN.
<code>“slapi_sdn_scope_test()”</code> on page 511	Checks if an entry is in the scope of a certain base DN.
<code>“slapi_log_info_ex()”</code> on page 409	Sets a DN value in a “Slapi_DN” on page 263 structure pointing to an existing DN string.
<code>“slapi_sdn_set_dn_byval()”</code> on page 513	Sets a DN value in a “Slapi_DN” on page 263 structure copying an existing DN string.

TABLE 16–19 Functions for Handling Slapi_DN Structures (Continued)

Function	Description
“slapi_sdn_set_dn_passin()” on page 514	Sets a DN value in a “ Slapi_DN ” on page 263 structure pointing to a new copy of a DN string.
“slapi_sdn_set_ndn_byref()” on page 515	Sets a normalized DN in a “ Slapi_DN ” on page 263 structure pointing to an existing normalized DN string.
“slapi_sdn_set_ndn_byval()” on page 515	Sets a normalized DN in a “ Slapi_DN ” on page 263 structure copying an existing normalized DN string.
“slapi_sdn_set_parent()” on page 516	Sets a new parent in an entry.
“slapi_sdn_set_rdn()” on page 517	Sets a new RDN for an entry.

TABLE 16–20 Functions for Handling Slapi_RDN Structures

Function	Description
“slapi_rdn_add()” on page 468	Adds a new RDN to an existing RDN structure.
“slapi_rdn_compare()” on page 469	Compares two RDNs.
“slapi_rdn_contains()” on page 469	Checks if a “ Slapi_RDN ” on page 269 structure holds any RDN matching a given type/value pair.
“slapi_rdn_contains_attr()” on page 470	Checks if a “ Slapi_RDN ” on page 269 structure contains any RDN matching a given type.
“slapi_rdn_done()” on page 471	Clears a “ Slapi_RDN ” on page 269 structure.
“slapi_rdn_free()” on page 472	Frees a “ Slapi_RDN ” on page 269 structure.
“slapi_rdn_get_first()” on page 472	Gets the type/value pair of the first RDN.
“slapi_rdn_get_index()” on page 473	Gets the index of the RDN.
“slapi_rdn_get_index_attr()” on page 474	Gets the position and the attribute value of the first RDN.
“slapi_rdn_get_next()” on page 475	Gets the RDN type/value pair from the RDN.
“slapi_rdn_get_num_components()” on page 476	Gets the number of RDN type/value pairs.
“slapi_rdn_get_rdn()” on page 476	Gets the RDN from a “ Slapi_RDN ” on page 269 structure.
“slapi_rdn_init()” on page 477	Initializes a “ Slapi_RDN ” on page 269 structure with NULL values.
“slapi_rdn_init_dn()” on page 477	Initializes a “ Slapi_RDN ” on page 269 structure from an existing DN string.

TABLE 16–20 Functions for Handling Slapi_RDN Structures (Continued)

Function	Description
“slapi_rdn_init_rdn()” on page 478	Initializes a “Slapi_RDN” on page 269 structure from an existing “Slapi_RDN” on page 269 structure.
“slapi_rdn_init_sdn()” on page 479	Initializes a “Slapi_RDN” on page 269 structure from an existing “Slapi_DN” on page 263 structure.
“slapi_rdn_isempty()” on page 479	Checks if an RDN value is stored in a “Slapi_RDN” on page 269 structure.
“slapi_rdn_new()” on page 480	Creates a new “Slapi_RDN” on page 269 structure.
“slapi_rdn_new_dn()” on page 481	Creates a new “Slapi_RDN” on page 269 structure.
“slapi_rdn_new_rdn()” on page 481	Creates a new “Slapi_RDN” on page 269 structure.
“slapi_rdn_new_sdn()” on page 482	Creates a new “Slapi_RDN” on page 269 structure.
“slapi_rdn_syntax_check()” on page 487	Determines if and RDN complies with attribute syntax rules.

TABLE 16–21 Functions for Handling Slapi_Value Structures

Function	Description
“slapi_value_compare()” on page 536	Compares two values.
“slapi_value_done()” on page 537	Frees internals of a value.
“slapi_value_dup()” on page 537	Duplicates a value.
“slapi_value_free()” on page 538	Frees a “Slapi_Value” on page 270 structure from memory.
“slapi_value_get_berval()” on page 539	Gets the “berval” on page 249 structure of the value.
“slapi_value_get_int()” on page 540	Converts the value of an integer.
“slapi_value_get_length()” on page 541	Gets the length of a value.
“slapi_value_get_long()” on page 541	Converts a value into a long integer.
“slapi_value_get_string()” on page 542	Returns the value as a string.
“slapi_value_get_uint()” on page 543	Converts the value into an unsigned integer.
“slapi_value_get_ulong()” on page 544	Converts the value into an unsigned long.
“slapi_value_init()” on page 544	Initializes a “Slapi_Value” on page 270 structure with no values.
“slapi_value_init_berval()” on page 545	Initializes a “Slapi_Value” on page 270 structure from the “berval” on page 249 structure.

TABLE 16–21 Functions for Handling `Slapi_Value` Structures (Continued)

Function	Description
<code>slapi_value_init_string()</code> on page 546	Initializes a “ <code>Slapi_Value</code> ” on page 270 structure from a string.
<code>slapi_value_init_string_passin()</code> on page 547	Initializes a “ <code>Slapi_Value</code> ” on page 270 structure with a value contained in a string.
<code>slapi_value_new()</code> on page 547	Allocates a new “ <code>Slapi_Value</code> ” on page 270 structure.
<code>slapi_value_new_berval()</code> on page 548	Allocates a new “ <code>Slapi_Value</code> ” on page 270 structure from a “ <code>berval</code> ” on page 249 structure.
<code>slapi_value_new_string()</code> on page 549	Allocates a new “ <code>Slapi_Value</code> ” on page 270 structure from a string.
<code>slapi_value_new_string_passin()</code> on page 550	Allocates a new “ <code>Slapi_Value</code> ” on page 270 structure and initializes it from a string.
<code>slapi_value_new_value()</code> on page 551	Allocates a new “ <code>Slapi_Value</code> ” on page 270 from another “ <code>Slapi_Value</code> ” on page 270 structure.
<code>slapi_value_set()</code> on page 552	Sets the value.
<code>slapi_value_set_berval()</code> on page 553	Copies the value from a “ <code>berval</code> ” on page 249 structure into a “ <code>Slapi_Value</code> ” on page 270 structure.
<code>slapi_value_set_int()</code> on page 554	Sets the integer value of a “ <code>Slapi_Value</code> ” on page 270 structure.
<code>slapi_value_set_string()</code> on page 555	Copies a string into the value.
<code>slapi_value_set_string_passin()</code> on page 556	Sets the value.
<code>slapi_value_set_value()</code> on page 557	Copies the value of a “ <code>Slapi_Value</code> ” on page 270 structure into another “ <code>Slapi_Value</code> ” on page 270 structure.

TABLE 16–22 Functions for Handling `Slapi_ValueSet` Structures

Function	Description
<code>slapi_valueset_add_value_optimised()</code> on page 557	Adds a “ <code>Slapi_Value</code> ” on page 270 in the “ <code>Slapi_ValueSet</code> ” on page 270 structure.
<code>slapi_valueset_count()</code> on page 558	Returns the number of values contained in a “ <code>Slapi_ValueSet</code> ” on page 270 structure.
<code>slapi_valueset_done()</code> on page 559	Frees the values contained in the “ <code>Slapi_ValueSet</code> ” on page 270 structure.
<code>slapi_valueset_find_const()</code> on page 559	Finds the value in a value set by using the syntax of an attribute.

TABLE 16–22 Functions for Handling Slapi_ValueSet Structures (Continued)

Function	Description
<code>“slapi_valueset_first_value_const()”</code> on page 560	Gets the first value of a “Slapi_ValueSet” on page 270 structure.
<code>“slapi_valueset_free()”</code> on page 561	Frees the specified “Slapi_ValueSet” on page 270 structure and its members from memory.
<code>“slapi_valueset_init()”</code> on page 562	Resets a “Slapi_ValueSet” on page 270 structure to no values.
<code>“slapi_valueset_new()”</code> on page 562	Allocates a new “Slapi_ValueSet” on page 270 structure.
<code>“slapi_valueset_next_value_const()”</code> on page 563	Gets the next value from a “Slapi_ValueSet” on page 270 structure.
<code>“slapi_valueset_set_from_smod()”</code> on page 564	Copies the values of “Slapi_Mod” on page 265 structure into a “Slapi_ValueSet” on page 270 structure.
<code>“slapi_valueset_set_valueset_optimised()”</code> on page 565	Initializes a “Slapi_ValueSet” on page 270 structure from another “Slapi_ValueSet” on page 270 structure.

TABLE 16–23 Functions for Handling UTF-8 Strings

Function	Description
<code>“slapi_has8thBit()”</code> on page 402	Checks if a string has an 8-bit character.
<code>“slapi_UTF-8CASECMP()”</code> on page 531	Compares two UTF-8 strings.
<code>“slapi_UTF-8NCASECMP()”</code> on page 532	Compares a specified number of UTF-8 characters.
<code>“slapi_UTF-8ISLOWER()”</code> on page 533	Verifies if a UTF-8 character is lower case.
<code>“slapi_UTF-8ISUPPER()”</code> on page 533	Verifies if a single UTF-8 character is upper case.
<code>“slapi_UTF-8STRTOLOWER()”</code> on page 534	Converts a UTF-8 string to lower case.
<code>“slapi_UTF-8STRTOUPPER()”</code> on page 534	Converts a string made up of UTF-8 characters and converts it to upper case.
<code>“slapi_UTF-8TOLOWER()”</code> on page 535	Converts a UTF-8 character to lower case.
<code>“slapi_UTF-8TOUPPER()”</code> on page 536	Converts a lower case UTF-8 character to an upper case character.

TABLE 16–24 Functions for Writing Log Messages

Function	Description
“slapi_log_error_ex()” on page 407	Writes an error message to the server error log
“slapi_log_info_ex()” on page 409	Writes an informational message to the server error log
“slapi_log_warning_ex()” on page 411	Writes a warning message to the server error log

TABLE 16–25 Functions for Handling Virtual Attributes

Function	Description
“slapi_vattr_is_virtual()” on page 569	Determines if the value of the specified attribute type is virtually generated.
“slapi_vattr_value_compare()” on page 570	Compares attribute type and name in a given entry.
“slapi_vattr_values_free()” on page 571	Frees the value set and type names.
“slapi_vattr_values_get_ex()” on page 572	Returns the values for an attribute type from an entry.

TABLE 16–26 Functions for Sending Entries, Referrals, and Results to Clients

Function	Description
“slapi_send_ldap_referral()” on page 523	Processes an entry’s LDAP v3 referrals.
“slapi_send_ldap_result()” on page 524	Sends a result code back to the client.
“slapi_send_ldap_search_entry()” on page 526	Sends an entry found by a search back to the client.

TABLE 16–27 Function for Registering Plug-Ins

Function	Description
“slapi_register_plugin()” on page 489	Register another plug-in.

Function Descriptions, Part I

The following sections cover plug-in API functions in alphabetical order from `slapi_access_allowed()` to `slapi_moddn_get_newdn()`.

Subsequent sections in the following chapter cover plug-in API functions from `slapi_modify_internal_pb()` to `slapi_wait_condvar()`.

Functions Alphabetically, Part 1

slapi_access_allowed()

Determines if the user 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:

pb	Parameter block passed into this function.
e	Entry for which you want to check the access rights.
attr	Attribute for which you want to check the access rights.
val	Pointer to the “ berval ” on page 249 structure containing the value for which you want to check the access rights.
access	Type of access rights that you want to check. For example, to check for write access, pass <code>SLAPI_ACL_WRITE</code> as the value of this argument.

The value of the access argument can be one of the following:

<code>SLAPI_ACL_ADD</code>	Permission to add a specified entry.
<code>SLAPI_ACL_COMPARE</code>	Permission to compare the specified values of an attribute in an entry.
<code>SLAPI_ACL_DELETE</code>	Permission to delete a specified entry.
<code>SLAPI_ACL_READ</code>	Permission to read a specified attribute.
<code>SLAPI_ACL_SEARCH</code>	Permission to search on a specified attribute or value.
<code>SLAPI_ACL_WRITE</code>	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:

<code>LDAP_OPERATIONS_ERROR</code>	<p>An error occurred while executing the operation.</p> <p>This error can occur if, for example, the type of access rights specified are not recognized by the server. In other words, you did not pass a value from the previous table.</p>
<code>LDAP_INVALID_SYNTAX</code>	<p>Invalid syntax was specified.</p> <p>This error can occur if the ACL associated with an entry, attribute, or value uses the wrong syntax.</p>
<code>LDAP_UNWILLING_TO_PERFORM</code>	<p>The DSA (this Directory Server instance) is unable to perform the specified operation.</p> <p>This error can occur if, for example, you are requesting write access to a read-only database.</p>

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 preoperation 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, this function does the following:

- Checks 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 function checks for this condition.

The caller must ensure that the backend specified in the parameter block 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,
        "Obj not 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);
```

slapi_acl_check_mods()

Determines if a user has the rights to perform the specified modifications on an entry.

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:

pb	Parameter block passed into this function.
e	Entry for which you want to check the access rights.
mods	Array of “ LDAPMod ” on page 250 structures that represent the modifications to be made to the entry.
errbuf	Pointer to a string containing an error message if an error occurs during the processing of this function.

Returns

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:

<code>LDAP_OPERATIONS_ERROR</code>	An error occurred while executing the operation.
<code>LDAP_INVALID_SYNTAX</code>	Invalid syntax was specified. This error can occur if the ACL associated with an entry, attribute, or value uses the wrong syntax.
<code>LDAP_UNWILLING_TO_PERFORM</code>	The DSA (this 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.

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.

For example, if 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 function does the following:

- Checks to access control for the directory is disabled.
If access control is disabled, the function returns `LDAP_SUCCESS`.
- For each value in each attribute specified in the [“LDAPMod” on page 250](#) array, the function determines if the user has permissions to write to that value. Specifically, the function calls [“slapi_access_allowed\(\)” on page 292](#) 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.

Memory Concerns

You must free the `errbuf` buffer by calling “[slapi_ch_free\(\)](#)” on page 329 when you are finished using the error message.

See Also

“[slapi_access_allowed\(\)](#)” on page 292

“[slapi_ch_free\(\)](#)” on page 329

slapi_acl_verify_aci_syntax()

Determines whether or not 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:

- | | |
|---------------------|---|
| <code>e</code> | Entry for which you want to check the ACIs. |
| <code>errbuf</code> | Pointer to the error message returned if the ACI syntax is invalid. |

Returns

This function returns 0 if successful, or -1 if an error occurs.

Memory Concerns

You must free the `errbuf` buffer by calling “[slapi_ch_free\(\)](#)” on page 329 when you are finished using the error message.

See Also

“[slapi_ch_free\(\)](#)” on page 329

slapi_add_entry_internal_set_pb()

Prepares a parameter block for an internal add operation involving a “[Slapi_Entry](#)” on [page 263](#) structure.

Syntax

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

pb	Parameter block for the internal add operation
e	Entry to add
controls	Array of controls to request for the add operation; NULL if no controls
plugin_identity	Plug-in identifier obtained from SLAPI_PLUGIN_IDENTITY during plug-in initialization
operation_flags	NULL or SLAPI_OP_FLAG_NEVER_CHAIN

Returns

This function returns 0 if successful. Otherwise it returns an LDAP error code.

Description

This function prepares a parameter block for use with “[slapi_add_internal_pb\(\)](#)” on [page 298](#) using the entry.

Memory Concerns

Allocate the parameter block using “[slapi_pblock_new\(\)](#)” on [page 464](#) before calling this function.

The entry is consumed during the call to “[slapi_add_internal_pb\(\)](#)” on [page 298](#). The LDAPControls are not consumed.

Free the parameter block after calling “[slapi_add_internal_pb\(\)](#)” on [page 298](#).

See Also

[“slapi_add_internal_pb\(\)” on page 298](#)

[“slapi_add_internal_set_pb\(\)” on page 299](#)

[“slapi_pblock_new\(\)” on page 464](#)

slapi_add_internal_pb()

Performs an internal add operation of a new directory entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_add_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

pb A parameter block that has been initialized using [“slapi_add_internal_set_pb\(\)” on page 299](#).

Returns

This function returns -1 if the parameter passed is a NULL pointer. Otherwise, it returns 0.

After your code calls this function, the server sets `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to the appropriate LDAP result code. You can therefore check `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to determine whether an error has occurred.

Description

This function performs an internal add operation based on a parameter block. The parameter block should be initialized by calling [“slapi_add_internal_set_pb\(\)” on page 299](#) or [“slapi_add_entry_internal_set_pb\(\)” on page 297](#).

Memory Concerns

None of the parameters that are passed to [“slapi_add_internal_set_pb\(\)” on page 299](#) or [“slapi_add_entry_internal_set_pb\(\)” on page 297](#) are altered or consumed by this function. The entry parameter that is passed to [“slapi_add_entry_internal_set_pb\(\)” on page 297](#) is consumed by a successful call to this function.

slapi_add_internal_set_pb()

Prepares a parameter block for an internal add operation.

Syntax

```
#include "slapi-plugin.h"
int slapi_add_internal_set_pb(Slapi_PBlock *pb, const char *dn,
    LDAPMod **attrs, LDAPControl **controls,
    Slapi_ComponentId *plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

pb	Parameter block for the internal add operation
dn	Distinguished Name of the entry to add
attrs	Array of attributes of the entry to add
controls	Array of controls to request for the add operation
plugin_identity	Plug-in identifier obtained from SLAPI_PLUGIN_IDENTITY during plug-in initialization
operation_flags	NULL or SLAPI_OP_FLAG_NEVER_CHAIN

Returns

This function returns 0 if successful. Otherwise, it returns an LDAP error code.

Description

This function prepares a parameter block for use with [“slapi_add_internal_pb\(\)” on page 298](#) using the components of the entry.

If the entry has already been prepared as a [“Slapi_Entry” on page 263](#) structure, use [“slapi_add_entry_internal_set_pb\(\)” on page 297](#) instead.

Memory Concerns

Allocate the parameter block using [“slapi_pblock_new\(\)” on page 464](#) before calling this function.

Directory Server does not free the parameters passed to this function.

Free the parameter block after calling [“slapi_pblock_destroy\(\)” on page 461](#).

See Also

[“slapi_add_entry_internal_set_pb\(\)” on page 297](#)

[“slapi_add_internal_pb\(\)” on page 298](#)

[“slapi_pblock_new\(\)” on page 464](#)

slapi_attr_add_value()

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:

a The attribute that will contain the values.

v Values to be added to the attribute.

Returns

This function always returns 0.

Memory Concerns

Directory Server makes a copy of the `Slapi_Value` to be added to the attribute.

See Also

[“slapi_attr_first_value_const\(\)” on page 302](#)

[“slapi_attr_next_value_const\(\)” on page 311](#)

[“slapi_attr_get_numvalues\(\)” on page 306](#)

[“slapi_attr_value_cmp\(\)” on page 315](#)

[“slapi_attr_value_find\(\)” on page 315](#)

slapi_attr_basetype()

Returns the base type of an attribute.

Syntax

```
#include "slapi-plugin.h"
char *slapi_attr_basetype( char *type, char *buf, size_t bufsiz );
```

Parameters

This function takes the following parameters:

type	Attribute type from which you wish to get the base type.
buf	Buffer to hold the returned base type.
bufsiz	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.

Description

This function returns the base type of an attribute (for example, if given `cn;lang-jp`, returns `cn`).

Memory Concerns

You should free the returned base type when you are finished by calling `“slapi_ch_free()”` on [page 329](#).

See Also

[“slapi_attr_get_type\(\)”](#) on [page 308](#)

[“slapi_attr_type_cmp\(\)”](#) on [page 313](#)

[“slapi_attr_types_equivalent\(\)”](#) on [page 314](#)

slapi_attr_dup()

Duplicates an attribute.

Syntax

```
#include "slapi-plugin.h"
Slapi_Attr *slapi_attr_dup(const Slapi_Attr *attr);
```

Parameters

This function takes the following parameter:

attr The attribute to be duplicated.

Returns

This function returns the newly created copy of the attribute.

Description

Use this function to make a copy of an attribute.

Memory Concerns

You must free the returned attribute using [“slapi_attr_free\(\)” on page 304](#).

See Also

[“slapi_attr_new\(\)” on page 311](#)

[“slapi_attr_init\(\)” on page 310](#)

[“slapi_attr_free\(\)” on page 304](#)

slapi_attr_first_value_const()

Gets the first value of an attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_first_value_const( const Slapi_Attr *a, const Slapi_Value **v );
```

Parameters

This function takes the following parameters:

a Attribute containing the desired value.

`v` 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 NULL.

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](#)” on [page 259](#) structure.

Memory Concerns

Do not free the value held in `v`.

See Also

“[slapi_attr_next_value_const\(\)](#)” on [page 311](#)

“[slapi_attr_get_numvalues\(\)](#)” on [page 306](#)

slapi_attr_flag_is_set()

Determines if certain flags are set for a particular attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_flag_is_set( Slapi_Attr *attr, unsigned long flag );
```

Parameters

This function takes the following parameters:

`attr` Attribute that you want to check.

`flag` Flag that you want 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.

Description

This function determines if certain flags are set for the specified attribute. These flags can identify an attribute as a single-valued attribute, an operational attribute, or as a read-only attribute, and are set from the schema when the `Slapi_Attr` structure is initialized.

See Also

[“`slapi_attr_get_flags\(\)`” on page 305](#)

`slapi_attr_free()`

Frees an attribute.

Syntax

```
#include "slapi-plugin.h"
void slapi_attr_free( Slapi_Attr **a );
```

Parameters

This function takes the following parameters:

a Attribute to be freed.

Description

Use this function to free an attribute when you are finished with it.

See Also

[“`slapi_attr_new\(\)`” on page 311](#)

[“`slapi_attr_init\(\)`” on page 310](#)

[“`slapi_attr_free\(\)`” on page 304](#)

slapi_attr_get_berval_copy()

Puts the values contained in an attribute into an array of “[berval](#)” on page 249 structures.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_berval_copy(Slapi_Attr *a,
    struct berval **vals );
```

Parameters

This function takes the following parameters:

- a* Attribute that contains the desired values.
- vals* Pointer to an array of “[berval](#)” on page 249 structure pointers to hold the desired values.

Returns

This function returns one of the following values:

- 0 if values are found.
- -1 if NULL.

Description

This function copies the values from an attribute into an array of “[berval](#)” on page 249 structure pointers.

Memory Concerns

Free this array using `ber_bvecfree(3LDAP)`.

slapi_attr_get_flags()

Gets the flags associated with the specified attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_flags( Slapi_Attr *attr, unsigned long *flags );
```

Parameters

This function takes the following parameters:

- attr** Attribute for which you wish to get the flags.
- flags** When you call this function, 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 use bitwise AND on 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.

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.

See Also

[“`slapi_attr_flag_is_set\(\)`” on page 303](#)

`slapi_attr_get_numvalues()`

Puts the count of values of an attribute into a provided integer.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_numvalues( const Slapi_Attr *a, int *numValues);
```

Parameters

This function takes the following parameters:

- a*** Attribute containing the values to be counted.

numValues Integer to hold the counted values.

Returns

This function always returns 0.

Description

This function counts the number of values in an attribute and places that count in an integer.

See Also

[“slapi_attr_first_value_const\(\)” on page 302](#)

[“slapi_attr_next_value_const\(\)” on page 311](#)

slapi_attr_get_oid_copy()

Searches the syntaxes for an attribute type, and returns a copy of its 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:

- attr* Attribute that contains the desired type.
- oidp* 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.

Description

Use this function to search the syntaxes for an attribute type's OID.

Memory Concerns

You should free this string using “[slapi_ch_free\(\)](#)” on page 329.

slapi_attr_get_plugin()

Gets a pointer to the syntax plug-in used to handle values of the attribute type in question.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_get_plugin( Slapi_Attr *a, void **plugin );
```

Parameters

This function takes the following parameters:

- a Attribute whose associated syntax plug-in you want to access.
- plugin This parameter is set to a pointer to the plug-in registered to handle attributes of the type passed to this function. Do not free this pointer as it is not a copy.

Returns

This function returns 0.

slapi_attr_get_type()

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:

- attr Attribute of which you wish to get the type.
- type When you call this function, 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_type_cmp\(\)” on page 313](#)

[“slapi_attr_types_equivalent\(\)” on page 314](#)

[“slapi_attr_basetype\(\)” on page 301](#)

slapi_attr_get_valueset()

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:

- a* Attribute containing the values to be placed into a value set. This must be a valid attribute, not NULL.
- vs* Receives values from the first parameter.

Returns

This function always returns 0.

Memory Concerns

Free the value set in *vs* using `slapi_valueset_free()`.

See Also

[“slapi_entry_add_valueset\(\)” on page 354](#)

[“slapi_valueset_new\(\)” on page 562](#)

[“slapi_valueset_init\(\)” on page 562](#)

[“slapi_valueset_free\(\)” on page 561](#)

[“slapi_valueset_done\(\)” on page 559](#)

[“slapi_valueset_add_value_optimised\(\)” on page 557](#)

[“slapi_valueset_first_value_const\(\)” on page 560](#)

[“slapi_valueset_next_value_const\(\)” on page 563](#)

[“slapi_valueset_count\(\)” on page 558](#)

slapi_attr_init()

Initializes an empty attribute with a base type.

Syntax

```
#include "slapi-plugin.h"
Slapi_Attr *slapi_attr_init(Slapi_Attr *a, const char *type);
```

Parameters

This function takes the following parameters:

a The empty attribute to be initialized.
type Attribute type to be initialized.

Returns

This function returns the newly initialized attribute, or an empty attribute if the type is not specified in the schema.

Description

Use this function to initialize an empty attribute with an attribute type.

Memory Concerns

Directory Server makes a copy of the type string.

See Also

[“slapi_attr_new\(\)” on page 311](#)

[“slapi_attr_dup\(\)” on page 301](#)

[“slapi_attr_free\(\)” on page 304](#)

slapi_attr_new()

Creates a new 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

Description

Use this function to create an empty attribute.

See Also

[“slapi_attr_dup\(\)” on page 301](#)

[“slapi_attr_free\(\)” on page 304](#)

slapi_attr_next_value_const()

Gets the next value of an attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_next_value_const( const Slapi_Attr *a, int index,
                                const Slapi_Value **v );
```

Parameters

This function takes the following parameters:

- a* Attribute contained the desired value.
- index* Index of the value to be returned, counting from 0.
- v* 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

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” on page 259](#) structure.

See Also

[“slapi_attr_first_value_const\(\)” on page 302](#)

[“slapi_attr_get_numvalues\(\)” on page 306](#)

`slapi_attr_syntax_normalize()`

Searches for an attribute type in the syntaxes, and returns a copy of the normalized attribute types.

Syntax

```
#include "slapi-plugin.h"
char * slapi_attr_syntax_normalize( const char *s );
```

Parameters

This function takes the following parameter:

- s* 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.

Description

Use this function to search the syntaxes for an attribute type and return its normalized form. If the attribute type is not defined in the schema, this function returns a copy of the type folded to lower case.

Memory Concerns

You should free the returned string using “[slapi_ch_free\(\)](#)” on page 329.

See Also

“[slapi_ch_free\(\)](#)” on page 329

slapi_attr_type_cmp()

Compares two attribute types to determine if they are the same.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_type_cmp( char *t1, char *t2, int opt );
```

Parameters

This function takes the following parameters:

- t1* Name of the first attribute type that you want to compare.
- t2* Name of the second attribute type that you want to compare.
- opt* One of the following values:
 - 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 0 if the type names are equal, or a non-zero value if they are not equal.

See Also

[“slapi_attr_type_cmp\(\)” on page 313](#)

[“slapi_attr_types_equivalent\(\)” on page 314](#)

[“slapi_attr_basetype\(\)” on page 301](#)

slapi_attr_types_equivalent()

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:

- t1* Pointer to the first attribute type that you want to compare.
- t2* Pointer to the second attribute type that you want to compare.

Returns

This function returns the 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\(\)” on page 300](#)

[“slapi_attr_first_value_const\(\)” on page 302](#)

[“slapi_attr_next_value_const\(\)” on page 311](#)

[“slapi_attr_get_numvalues\(\)” on page 306](#)

[“slapi_attr_value_find\(\)” on page 315](#)

slapi_attr_value_cmp()

Compares two values for a given attribute to determine if they are equal.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_value_cmp( Slapi_Attr *attr, struct berval *v1,
    struct berval *v2 );
```

Parameters

This function takes the following parameters:

- attr* 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).
- v1* Pointer to the “[berval](#)” on page 249 structure containing the first value that you want to compare.
- v2* Pointer to the “[berval](#)” on page 249 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 they are not equal.

See Also

“[slapi_attr_add_value\(\)](#)” on page 300

“[slapi_attr_first_value_const\(\)](#)” on page 302

“[slapi_attr_next_value_const\(\)](#)” on page 311

“[slapi_attr_get_numvalues\(\)](#)” on page 306

“[slapi_attr_value_find\(\)](#)” on page 315

slapi_attr_value_find()

Determines if an attribute contains a given value, using the equality matching rule.

Syntax

```
#include "slapi-plugin.h"
int slapi_attr_value_find( Slapi_Attr *a, struct berval *v );
```

Parameters

This function takes the following parameters:

- a* Attribute that you wish to check.
- v* Pointer to the “[berval](#)” on [page 249](#) structure containing the value for which you wish to search.

Returns

This function returns one of the following values:

- 0 if the attribute contains a match for the specified value according to the equality matching rule.
As an example for CN, BABS JENSEN and Babs Jensen match.
- -1 if the attribute does not contain the specified value.

See Also

“[slapi_attr_add_value\(\)](#)” on [page 300](#)

“[slapi_attr_first_value_const\(\)](#)” on [page 302](#)

“[slapi_attr_next_value_const\(\)](#)” on [page 311](#)

“[slapi_attr_get_numvalues\(\)](#)” on [page 306](#)

“[slapi_attr_value_cmp\(\)](#)” on [page 315](#)

slapi_be_exist()

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:

sdn 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\(\)” on page 322](#)

slapi_be_get_name()

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:

be 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.

slapi_be_get_readonly()

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:

be 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.

slapi_be_getsuffix()

Returns the DN of the *n*th suffix associated with the specified backend.

Syntax

```
#include "slapi-plugin.h"
const Slapi_DN *slapi_be_getsuffix(Slapi_Backend *be, int n);
```

Parameters

This function takes the following parameters:

be Pointer to the structure containing the backend configuration.

n Index, starting from 0.

Returns

This function returns the DN of the suffix if it exists, or NULL if there is no *n*th suffix in the backend.

Description

This function returns the *n*th suffix, counting from 0, associated with the specified backend. This function is present for compatibility purposes with previous versions of the Directory Server Plug-In API.

Memory Concerns

You should not free the returned pointer.

slapi_be_gettype()

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:

be Pointer to the structure containing the backend configuration.

Returns

This function returns the type of the backend. Backend types include:

- `chaining` (used for chaining, also known as database links)
- `default` (supporting only binds, used for Pass Through Authentication)
- `DSE` (used for configuration parameters)
- `ldbm database` (housing your directory data)
- `schema-internal` (used for schema configuration)

Memory Concerns

You should not free the returned pointer.

slapi_be_is_flag_set()

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:

- be* Pointer to the structure containing the backend configuration.
- flag* Flag to check (SLAPI_BE_FLAG_REMOTE_DATA, SLAPI_BE_FLAG_SUSPENDED).

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.

slapi_be_issuffix()

Verifies that the specified suffix matches a registered backend suffix.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_issuffix(const Slapi_Backend *be,
                     const Slapi_DN *suffix);
```

Parameters

This function takes the following parameters:

- be* Pointer to the structure containing the backend configuration.
- suffix* 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.

Description

This function checks if the specified suffix exactly matches a registered suffix on a specified backend.

slapi_be_logchanges()

Indicates if the changes applied to the LDBM database backend should be logged in the changelog. You can only read this value, not set it.

Syntax

```
#include "slapi-plugin.h"
int slapi_be_logchanges(Slapi_Backend *be);
```

Parameters

This function takes the following parameter:

be 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.

slapi_be_private()

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:

be 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 (for backend types chaining and ldbm database)

- 1 if the backend is hidden from the user (for internal use only, backend types `default`, `DSE`, `schema-internal`).

`slapi_be_select()`

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:

sdn Pointer to the DN of which you wish to get the backend.

Returns

This function returns a pointer to the default backend, which is an empty backend allowing only bind operations, if no backend with the appropriate suffix is configured. Otherwise, it returns a pointer to the backend structure.

Memory Concerns

You should not free the returned pointer.

See Also

[“`slapi_be_select_by_instance_name\(\)`” on page 322](#)

`slapi_be_select_by_instance_name()`

Find the backend used to service the database.

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend *slapi_be_select_by_instance_name( const char *name );
```

Parameters

This function takes the following parameter:

name Pointer to the value of the CN for the backend whose structure you want, such as userRoot.

Returns

This function returns NULL if no backend with the appropriate name is configured. Otherwise, it returns a pointer to the backend structure.

Description

This function finds the backend that should be used to service the database named as the parameter.

Memory Concerns

You should not free the returned pointer.

See Also

[“slapi_be_select\(\)” on page 322](#)

slapi_berval_cmp()

Compare two [“berval” on page 249](#) structures.

Syntax

```
#include "slapi-plugin.h"
int slapi_berval_cmp(const struct berval* L,const struct berval* R);
```

Parameters

This function takes the following parameters:

L One of the [“berval” on page 249](#) structures
R The other [“berval” on page 249](#) structure

Description

This function checks whether two [“berval” on page 249](#) structures are equivalent.

Returns

This function returns 0 if the two “[berval](#)” [on page 249](#) structures are equivalent. It returns a negative value if L is shorter than R, and a positive value if L is longer than R. If L and R are of the same size but their content differs, this function returns a negative value if L is less than R, or a positive value if L is greater than R, where L and R are compared as arrays of bytes.

slapi_build_control()

Creates an “[LDAPControl](#)” [on page 250](#) structure based on a BerElement, an OID, and a criticality flag.

Syntax

```
#include "slapi-plugin.h"
int slapi_build_control( char const *oid, BerElement const *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 ” on page 250 structure. |

Returns

This function returns LDAP_SUCCESS (LDAP result code) if successful.

Description

This function creates an “[LDAPControl](#)” [on page 250](#) structure based on a BerElement, an OID, and a criticality flag. The “[LDAPControl](#)” [on page 250](#) that is created can be used in LDAP client requests or internal operations.

You can construct a BerElement using `ber_init(3LDAP)` for example.

Memory Concerns

Directory Server makes a copy of the oid string.

The contents of the ber parameter are the responsibility of the caller.

You can free the ber parameter of the “[slapi_build_control\(\)](#)” on page 324 using `ber_free(3LDAP)`.

The “[LDAPControl](#)” on page 250 pointer that is returned in `ctrlp` should be freed by calling `ldap_control_free(3LDAP)`, which is an LDAP API function.

See Also

```
ber_free(3LDAP)
ber_init(3LDAP)
ldap_control_free(3LDAP)
```

“[slapi_build_control_from_berval\(\)](#)” on page 325

slapi_build_control_from_berval()

Creates an “[LDAPControl](#)” on page 250 structure based on a “[berval](#)” on page 249 structure, an OID, and a criticality flag.

Syntax

```
#include "slapi-plugin.h"
int slapi_build_control_from_berval( char const *oid,
    struct berval *bvp, 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>bvp</i>	A “ berval ” on page 249 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 ” on page 250 structure.

Returns

This function always returns LDAP_SUCCESS (LDAP result code).

Description

This function creates an “LDAPControl” on page 250 structure based on a “berval” on page 249, an OID, and a criticality flag. The “LDAPControl” on page 250 that is created can be used in LDAP client requests or internal operations.

Memory Concerns

Directory Server makes a copy of the oid string.

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” on page 250 pointer that is returned in ctrlp should be freed by calling ldap_control_free(3LDAP), which is an LDAP API function.

See Also

ldap_control_free(3LDAP)

“slapi_build_control()” on page 324

slapi_ch_array_free()

Frees an existing array.

Syntax

```
#include "slapi-plugin.h"
void slapi_ch_array_free( char **arrayp );
```

Parameters

This function takes the following parameter:

arrayp Pointer to the array to be freed. This parameter can be NULL.

Description

This function frees the char ** pointed to by arrayp . In the following excerpt, for example, both array and a1 are freed:

```
char **array;
char *a1;

array = malloc(2*sizeof(char *));
a1 = strdup("hello");

array[0] = a1;
array[1] = NULL;

slapi_ch_array_free(array);
```

slapi_ch_bvdup()

Makes a copy of an existing “[berval](#)” on page 249 structure.

Syntax

```
#include "slapi-plugin.h"
struct berval* slapi_ch_bvdup( const struct berval *v );
```

Parameters

This function takes the following parameter:

v Pointer to the “[berval](#)” on page 249 structure that you want to copy.

Returns

This function returns a pointer to the new copy of the “[berval](#)” on page 249 structure. If the structure cannot be duplicated (for example, if no more virtual memory exists), the `slapd` program terminates.

Memory Concerns

The contents of the *v* parameter are not altered by this function. The returned “[berval](#)” on page 249 structure should be freed by calling `ber_bvfree(3LDAP)`, which is an LDAP API function.

See Also

`ber_bvfree(3LDAP)`

“[slapi_ch_bvecdup\(\)](#)” on page 328

slapi_ch_bvecdup()

Makes a copy of an array of existing “[berval](#)” on page 249 structures.

Syntax

```
#include "slapi-plugin.h"
extern struct berval** slapi_ch_bvecdup (const struct berval **v);
```

Parameters

This function takes the following parameters:

v Pointer to the array of “[berval](#)” on page 249 structures that you want to copy.

Returns

This function returns a pointer to an array of the new copy of the “[berval](#)” on page 249 structures. If the structures cannot be duplicated (for example, if no more virtual memory exists), the slapd program terminates.

Memory Concerns

The contents of the *v* parameter are not altered by this function. The returned “[berval](#)” on page 249 structure should be freed by calling `ber_bvfree(3LDAP)`, an LDAP API function.

See Also

`ber_bvfree(3LDAP)`

“[slapi_ch_bvecdup\(\)](#)” on page 328

slapi_ch_calloc()

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:

nelem Number of elements for which you wish to allocate memory.

size Size in bytes of the element for which you wish to allocate memory.

Returns

This function returns a pointer to the newly allocated space of memory. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Memory Concerns

This function should be called instead of the standard `calloc()` C function, and 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_ch_free\(\)](#)” on page 329.

See Also

“[slapi_ch_free\(\)](#)” on page 329

“[slapi_ch_malloc\(\)](#)” on page 331

“[slapi_ch_realloc\(\)](#)” on page 332

“[slapi_ch_strdup\(\)](#)” on page 332

`slapi_ch_free()`

Frees space allocated by the “[slapi_ch_malloc\(\)](#)” on page 331, “[slapi_ch_realloc\(\)](#)” on page 332, and “[slapi_ch_calloc\(\)](#)” on page 328 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:

ptr Address of the pointer to the block of memory that you wish to free. If `NULL`, no action occurs.

Memory Concerns

The `ptr` passed to `slapi_ch_free()` should be the address of a pointer to memory allocated using a call to [“`slapi_ch_malloc\(\)`” on page 331](#), [“`slapi_ch_realloc\(\)`” on page 332](#), [“`slapi_ch_calloc\(\)`” on page 328](#), or [“`slapi_ch_strdup\(\)`” on page 332](#).

See Also

[“`slapi_ch_calloc\(\)`” on page 328](#)

[“`slapi_ch_malloc\(\)`” on page 331](#)

[“`slapi_ch_realloc\(\)`” on page 332](#)

[“`slapi_ch_strdup\(\)`” on page 332](#)

`slapi_ch_free_string()`

Frees an existing string allocated by the [“`slapi_ch_malloc\(\)`” on page 331](#), [“`slapi_ch_realloc\(\)`” on page 332](#), and [“`slapi_ch_calloc\(\)`” on page 328](#). Call this function instead of the standard `free()` C function.

Syntax

```
#include "slapi-plugin.h"
void slapi_ch_free_string( char **s );
```

Parameters

This function takes the following parameter:

`s` Address of the string that you wish to free.

Description

This function frees an existing string, and should be used in favor of [“`slapi_ch_free\(\)`” on page 329](#) when using strings. It will perform compile-time error checking for incorrect error arguments, as opposed to [“`slapi_ch_free\(\)`” on page 329](#), which defeats the compile-time checking because you must cast the argument to `(void**)`.

See Also

[“`slapi_ch_free\(\)`” on page 329](#)

[“`slapi_ch_calloc\(\)`” on page 328](#)

[“slapi_ch_malloc\(\)” on page 331](#)

[“slapi_ch_realloc\(\)” on page 332](#)

[“slapi_ch_strdup\(\)” on page 332](#)

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:

size Size in bytes of the space for which you wish to get the memory.

Returns

This function returns a pointer to the newly allocated space of memory. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Memory Concerns

This function should be called instead of the standard `malloc()` C function, and 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\(\)” on page 329](#).

See Also

[“slapi_ch_free\(\)” on page 329](#)

[“slapi_ch_calloc\(\)” on page 328](#)

[“slapi_ch_realloc\(\)” on page 332](#)

[“slapi_ch_strdup\(\)” on page 332](#)

slapi_ch_realloc()

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:

block Pointer to an existing block of allocated memory.

size New size (in bytes) 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 (for example, if no more virtual memory exists), the `slapd` program terminates.

Memory Concerns

This function should be called instead of the standard `realloc()` C function, and terminates the `slapd` server with an “out of memory” error message if memory cannot be allocated.

The `block` parameter passed to this function should be the address of a pointer that was allocated using a `slapi` call such as “[slapi_ch_malloc\(\)](#)” on page 331, “[slapi_ch_calloc\(\)](#)” on page 328, or “[slapi_ch_strdup\(\)](#)” on page 332.

The returned pointer should be freed by calling “[slapi_ch_free\(\)](#)” on page 329.

See Also

“[slapi_ch_free\(\)](#)” on page 329

“[slapi_ch_calloc\(\)](#)” on page 328

“[slapi_ch_malloc\(\)](#)” on page 331

“[slapi_ch_strdup\(\)](#)” on page 332

slapi_ch_strdup()

Makes a copy of an existing string.

Syntax

```
#include "slapi-plugin.h"
char * slapi_ch_strdup( char *s );
```

Parameters

This function takes the following parameter:

s 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 (for example, if no more virtual memory exists), the `slapd` program terminates.

Memory Concerns

This function should be called instead of the standard `strdup()` C function, and 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\(\)](#)” on page 329.

See Also

“[slapi_ch_free\(\)](#)” on page 329

“[slapi_ch_calloc\(\)](#)” on page 328

“[slapi_ch_malloc\(\)](#)” on page 331

“[slapi_ch_realloc\(\)](#)” on page 332

`slapi_compute_add_evaluator()`

Sets a callback for use by the server in evaluating which computed attributes to generate and include in an entry before returning a result to a client.

Syntax

```
#include "slapi-plugin.h"
int slapi_compute_add_evaluator(slapi_compute_callback_t fcn);
```

Parameters

This function takes the following parameters:

fcn Function to call when evaluating computed attributes

Returns

This function returns 0 if successful. Otherwise, it returns ENOMEM indicating that no memory could be allocated for the callback.

Description

For a description of the callback, refer to “[slapi_compute_callback_t](#)” on page 261 . Register the callback as part of plug-in initialization.

See Also

“[computed_attr_context](#)” on page 249

“[slapi_compute_callback_t](#)” on page 261

“[slapi_pblock_new\(\)](#)” on page 464

slapi_compute_add_search_rewriter_ex()

Sets callbacks for use by the server in searching against computed attributes.

Syntax

```
#include "slapi-plugin.h"
int slapi_compute_add_search_rewriter_ex(
    slapi_search_rewrite_callback_t function,
    slapi_search_rewrite_callback_t cleanup_function);
```

Parameters

This function takes the following parameters:

function	Function to call to rewrite a filter for the search
cleanup_function	Function to call to cleanup after performing the rewritten search

Returns

This function returns 0 if successful. Otherwise, it returns ENOMEM indicating that no memory could be allocated for the callback.

Description

For a description of the callbacks, refer to the [Chapter 15](#)

slapi_control_present()

Determines whether or not the specified object identifier (OID) identifies a control that is present in a list of controls.

Syntax

```
#include "slapi-plugin.h"
int slapi_control_present( LDAPControl **controls, char const *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, this function specifies the pointer to the “ berval ” on page 249 structure containing the value of the control. If you do not want to receive a pointer to the control value, pass NULL for this parameter.
<i>iscritical</i>	If the control is present in the list of controls, this function specifies whether or not the control is critical to the operation of the server: <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. ▪ If you do not want to receive an indication of whether the control is critical or not, pass NULL for this parameter.

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_entry_get_uniqueid\(\)” on page 372](#)

[“slapi_register_supported_control\(\)” on page 491](#)

slapi_delete_internal_pb()

Performs an LDAP delete operation based on a parameter block to remove a directory entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_delete_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

pb A parameter block that has been initialized using [“slapi_delete_internal_set_pb\(\)” on page 337](#).

Returns

This function returns -1 if the parameter passed is a NULL pointer. Otherwise, it returns 0.

After your code calls this function, the server sets `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to the appropriate LDAP result code. You can therefore check `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to determine whether an error has occurred.

Description

This function performs an internal delete operation based on a parameter block. The parameter block should be initialized by calling [“slapi_delete_internal_set_pb\(\)” on page 337](#).

Memory Concerns

None of the parameters that are passed to [“slapi_delete_internal_set_pb\(\)” on page 337](#) are altered or consumed by this function.

See Also

[“slapi_delete_internal_set_pb\(\)” on page 337](#)

slapi_delete_internal_set_pb()

Prepares a parameter block for an internal delete operation.

Syntax

```
#include "slapi-plugin.h"
int slapi_delete_internal_set_pb(Slapi_PBlock *pb, const char *dn,
    LDAPControl **controls, const char *uniqueid,
    Slapi_ComponentId *plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

pb	Parameter block for the internal add operation
dn	Distinguished Name of the entry to add
controls	Array of controls to request for the add operation
uniqueid	Unique identifier for the entry if using this rather than DN.
plugin_identity	Plug-in identifier obtained from SLAPI_PLUGIN_IDENTITY during plug-in initialization
operation_flags	NULL or SLAPI_OP_FLAG_NEVER_CHAIN

Returns

This function returns 0 if successful. Otherwise, it returns an LDAP error code.

Description

This function prepares a parameter block for use with “[slapi_delete_internal_pb\(\)](#)” on [page 336](#) using the components of the entry.

Memory Concerns

Allocate the parameter block using “[slapi_pblock_new\(\)](#)” on [page 464](#) before calling this function.

Directory Server does not free the parameters you passed to this function.

Free the parameter block after calling “[slapi_delete_internal_pb\(\)](#)” on [page 336](#).

See Also

[“slapi_delete_internal_pb\(\)” on page 336](#)

[“slapi_pblock_new\(\)” on page 464](#)

slapi_destroy_condvar()

Frees a [“Slapi_CondVar” on page 262](#) structure from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_destroy_condvar( Slapi_CondVar *cvar );
```

Parameters

This function takes the following parameters:

cvar Pointer to the [“Slapi_CondVar” on page 262](#) structure that you want to free from memory.

Description

This function frees a [“Slapi_CondVar” on page 262](#) structure from memory. Before calling this function, you should make sure that this condition variable is no longer in use.

slapi_destroy_mutex()

Frees a [“Slapi_Mutex” on page 266](#) structure from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_destroy_mutex( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

mutex Pointer to the [“Slapi_Mutex” on page 266](#) structure that you want to free from memory.

Description

This function frees a “[Slapi_Mutex](#)” on page 266 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.

slapi_dn_beparent()

Gets a copy of the distinguished name (DN) of the parent of an entry, unless the specified entry’s DN is the suffix of the local database.

If you do not want to check if the entry’s DN is the suffix of the local database, call the “[slapi_dn_parent\(\)](#)” on page 345 function instead.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_beparent( Slapi_PBlock *pb, char *dn );
```

Parameters

This function takes the following parameters:

- pb* Parameter block.
- dn* DN of the entry for which you want to find the parent.

Returns

This function returns the DN of the parent entry; or NULL if the specified DN is NULL, if the DN is an empty string, if the DN has no parent (for example, o=example.com), or if the specified DN is the suffix of the local database.

See Also

“[slapi_dn_parent\(\)](#)” on page 345

slapi_dn_ignore_case()

Converts all characters in a distinguished name (DN) to lowercase.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_ignore_case( char *dn );
```

Parameters

This function takes the following parameters:

dn DN that you want to convert to lowercase.

Returns

This function returns the DN with lowercase characters. Notice that the variable passed in as the *dn* argument is also converted in place.

See Also

[“slapi_dn_normalize\(\)” on page 343](#)

slapi_dn_isbesuffix()

Determines whether or not the specified distinguished name (DN) is the suffix of the local database. Before calling this function, you should call [“slapi_dn_normalize_case\(\)” on page 344](#) to normalize the DN and convert all characters to lowercase.

Syntax

```
#include "slapi-plugin.h"
int slapi_dn_isbesuffix( Slapi_PBlock *pb, char *dn );
```

Parameters

This function takes the following parameters:

pb Parameter block.

dn DN that you want to check.

Returns

This function returns 1 if the specified DN is the suffix for the local database, or 0 if the DN is not the suffix.

slapi_dn_isbesuffix_norm()

Determines whether or not the specified distinguished name (DN) is the suffix of the local database.

Syntax

```
#include "slapi-plugin.h"
int slapi_dn_isbesuffix_norm( Slapi_PBlock *pb, const char *dn );
```

Parameters

This function takes the following parameters:

pb Parameter block.
dn DN that you want to check.

Returns

This function returns 1 if the specified DN is the suffix for the local database, or 0 if the DN is not the suffix.

slapi_dn_isparent()

Determines whether or not a particular DN is the parent of another specified DN. Before calling this function, you should call “[slapi_dn_normalize_case\(\)](#)” on [page 344](#) to normalize the DNs and convert all characters to lowercase.

Syntax

```
#include "slapi-plugin.h"
int slapi_dn_isparent( const char *parentdn, char *childdn );
```

Parameters

This function takes the following parameters:

parentdn Determine if this DN is the parent of *childdn*.
childdn Determine if this DN is the child of *parentdn*.

Returns

This function returns a non-zero value if *parentdn* is the parent of *childdn*, or 0 if the *parentdn* is not the parent of *childdn*.

See Also

[“slapi_dn_issuffix\(\)” on page 342](#)

slapi_dn_isroot()

Determines if the specified DN is the root DN for this local database. Before calling this function, you should call [“slapi_dn_normalize_case\(\)” on page 344](#) to normalize the DN and convert all characters to lowercase.

Syntax

```
#include "slapi-plugin.h"
int slapi_dn_isroot( Slapi_PBlock *pb, char *dn );
```

Parameters

This function takes the following parameters:

pb Parameter block.

dn DN that you want to check.

Returns

This function returns 1 if the specified DN is the root DN of the local database, or 0 if the DN is not the root DN.

slapi_dn_issuffix()

Determines if a DN is equal to the specified suffix. Before calling this function, you should call [“slapi_dn_normalize_case\(\)” on page 344](#) to normalize the DN and convert all characters to lowercase.

If you want to determine if a DN is the same as the suffix for the local database, call the [“slapi_dn_isbesuffix_norm\(\)” on page 341](#) function instead.

Syntax

```
#include "slapi-plugin.h"
int slapi_dn_issuffix( const char *dn, const char *suffix );
```

Parameters

This function takes the following parameters:

dn DN that you want to check.
suffix Suffix that you want compared against the DN.

Returns

This function returns 1 if the specified DN is the same as the specified suffix, or 0 if the DN is not the same as the suffix.

See Also

[“slapi_dn_isparent\(\)” on page 341](#)

slapi_dn_normalize()

Converts a distinguished name (DN) to canonical format (no leading or trailing spaces, no spaces between components, and no spaces around the equals sign). For example, given the following DN:

```
cn = Moxie Cross , ou = Engineering , dc = example , dc = com
```

the function returns:

```
cn=Moxie Cross,ou=Engineering,dc=example,dc=com
```

Syntax

```
#include "slapi-plugin.h"  
char *slapi_dn_normalize( char *dn );
```

Parameters

This function takes the following parameters:

dn DN that you want to normalize.

Returns

This function returns the normalized DN. Notice that the variable passed in as the *dn* argument is also converted in place.

See Also

[“slapi_dn_normalize_to_end\(\)” on page 344](#)

[“slapi_dn_normalize_case\(\)” on page 344](#)

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_dn_normalize\(\)” on page 343](#) function followed by the [“slapi_dn_ignore_case\(\)” on page 339](#) function.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_normalize_case( char *dn );
```

Parameters

This function takes the following parameters:

dn DN that you want to normalize and convert to lowercase.

Returns

This function returns the normalized DN with all lowercase characters. Notice that variable passed in as the *dn* argument is also converted in-place.

See Also

[“slapi_dn_normalize\(\)” on page 343](#)

[“slapi_dn_ignore_case\(\)” on page 339](#)

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*.

Notice that 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_dn_normalize\(\)](#), except for NULL terminating the normalized DN.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_normalize_to_end( char *dn, char *end );
```

Parameters

This function takes the following parameters:

- dn* DN value to be normalized.
- end* 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. In other words, the normalized portion is from **dn* to ** (returnValue - 1)*.

See Also

[“slapi_dn_normalize\(\)” on page 343](#)

slapi_dn_parent()

Gets a copy of the distinguished name (DN) of the parent of an entry. Before calling this function, you should call [“slapi_dn_normalize_case\(\)” on page 344](#) to normalize the DN and convert all characters to lowercase.

If you want to check if the DN is the suffix of the local database, call the [“slapi_dn_beparent\(\)” on page 339](#) function instead.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_parent( char *dn );
```

Parameters

This function takes the following parameter:

- dn* DN of the entry for which you want to find the parent.

Returns

This function returns the DN of the parent entry. If the specified DN is `NULL`, if the DN is an empty string, or if the DN has no parent (for example, `o=example.com`), the function returns `NULL`.

`slapi_dn_plus_rdn()`

Adds an RDN to DN.

Syntax

```
#include "slapi-plugin.h"
char *slapi_dn_plus_rdn( const char *dn, const char *rdn);
```

Parameters

This function takes the following parameters:

- dn* DN value to which a new RDN is to be added.
- rdn* RDN value that is to be added to the DN value in *dn*.

Returns

This function returns the new DN formed by adding the RDN value in *rdn* to the DN value in *dn*.

Memory Concerns

You must free the string returned with `slapi_ch_free_string()`.

See Also

[“`slapi_sdn_add_rdn\(\)`” on page 495](#)

`slapi_dup_control()`

Makes an allocated copy of an [“`LDAPControl`” on page 250](#).

Syntax

```
#include "slapi-plugin.h"
LDAPControl * slapi_dup_control( LDAPControl const *ctrl );
```

Parameters

This function takes the following parameter:

ctrl Pointer to an “[LDAPControl](#)” on page 250 structure whose contents are to be duplicated.

Returns

This function returns a pointer to an allocated “[LDAPControl](#)” on page 250 structure if successful, or NULL if an error occurs.

Description

This function duplicates the contents of an “[LDAPControl](#)” on page 250 structure. All fields within the “[LDAPControl](#)” on page 250 are copied to a new, allocated structure, and a pointer to the new structure is returned.

Memory Concerns

The structure that is returned should be freed by calling `ldap_control_free(3LDAP)`, an LDAP API function.

See Also

`ldap_control_free(3LDAP)`

`slapi_entry2mods()`

Creates an array of “[LDAPMod](#)” on page 250 from a “[Slapi_Entry](#)” on page 263.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry2mods(const Slapi_Entry *e,
    char **dn, LDAPMod ***attrs);
```

Parameters

This function takes the following parameters:

e Pointer to a “[Slapi_Entry](#)” on page 263.

dn Address of a `char*` that will be set on return to the entry DN.

attrs Address of an array of “[LDAPMod](#)” on page 250 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.
- non-0 if not successful.

Description

This function creates an array of “[LDAPMod](#)” on page 250 of type LDAP_MOD_ADD from a “[Slapi_Entry](#)” on page 263. Such structures may be useful for example when performing LDAP add and modify operations as a client from inside a plug-in.

See Also

“[slapi_mods2entry\(\)](#)” on page 434

slapi_entry2str()

Generates an LDIF string description of an LDAP entry.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry2str( Slapi_Entry const *e, int *len );
```

Parameters

This function takes the following parameters:

- e* Entry that you want to convert into an LDIF string.
- len* Length of the returned LDIF string.

Returns

Returns the LDIF string representation of the entry you specify. If an error occurs, the function returns NULL.

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, o=example.com
cn: Jane Doe
sn: Doe
...
```

To convert a string description in LDIF format to an entry of the “[Slapi_Entry](#)” on page 263 data type, call the “[slapi_str2entry\(\)](#)” on page 528 function.

Memory Concerns

When you no longer need to use the string, you should free it from memory by calling the “[slapi_ch_free_string\(\)](#)” on page 330 function.

See Also

“[slapi_entry2str_with_options\(\)](#)” on page 349

“[slapi_str2entry\(\)](#)” on page 528

slapi_entry2str_with_options()

Generates a description of an entry as an LDIF string. This function behaves much like “[slapi_str2entry\(\)](#)” on page 528. You can however specify output options with this function.

Syntax

```
#include "slapi-plugin.h"
char *slapi_entry2str_with_options( Slapi_Entry const *e,
    int *len, int options );
```

Parameters

This function takes the following parameters:

- e* Entry that you want to convert into an LDIF string.
- len* Length of the LDIF string returned by this function.
- options* 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
SLAPI_DUMP_STATEINFO	This is only used internally by replication. This allows access to the internal data used by multi-master replication.
SLAPI_DUMP_UNIQUEID	This option is used when creating an LDIF file to be used to initialize a replica. Each entry will contain the nsuniqueID operational attribute.
SLAPI_DUMP_NOOPATTRS	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.
SLAPI_DUMP_NOWRAP	By default, lines will be wrapped as defined in the LDIF specification. With this option, line wrapping is disabled.

Returns

This function returns the LDIF string representation of the entry you specify or NULL if an error occurs.

Description

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, o=example.com
cn: Jane Doe
sn: Doe
...
```

To convert an entry described in LDIF string format to an LDAP entry using the “[Slapi_Entry](#)” on page 263 data type, call the “[slapi_str2entry\(\)](#)” on page 528 function.

Memory Concerns

When you no longer need to use the string, you should free it from memory by calling the “[slapi_ch_free_string\(\)](#)” on page 330 function.

See Also

[“slapi_entry2str\(\)” on page 348](#)

[“slapi_str2entry\(\)” on page 528](#)

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.

Description

If the attribute type corresponding to the RDN already has a value matching the RDN value for equality, the value is not added. This function does not however examine other attribute types not in the RDN, whose values may match the RDN value for equality.

Memory Concerns

Free the entry from memory by using [“slapi_entry_free\(\)” on page 368](#) if the entry was allocated by the user.

See Also

[“slapi_entry_free\(\)” on page 368](#)

slapi_entry_add_string()

Adds a string value to an attribute in an entry.

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:

- e* Entry to which you want to add a string value.
- type* Attribute to which you want to add a string value.
- value* String value you want to add.

Returns

This function returns 0 when successful; any other value returned signals failure.

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.

This function does *not* check whether the value is already present for the attribute. Use `slapi_entry_attr_delete()` before using this function.

Memory Concerns

This routine makes a copy of the parameter value. If value is NULL, the entry is not changed.

slapi_entry_add_value()

Adds a specified “[Slapi_Value](#)” on [page 270](#) data value to an attribute in an entry.

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:

- e* Entry to which you want to add a value.
- type* Attribute to which you want to add a value.
- value* The “[Slapi_Value](#)” on page 270 data value you want to add to the entry.

Returns

Returns 0 when successful; any other value returned signals failure.

Description

This function adds a “[Slapi_Value](#)” on page 270 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](#)” on page 270 specified.

Memory Concerns

This routine makes a copy of the parameter *value*. If *value* is NULL, the entry is not changed.

`slapi_entry_add_values_sv()`

Adds an array of “[Slapi_Value](#)” on page 270 data values to the specified attribute in an entry.

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:

- e* Entry to which you want to add values.
- type* Attribute type to which you want to add values.
- vals* Array of “[Slapi_Value](#)” on page 270 data values that you want to add.

Returns

Returns one of the following values:

- LDAP_SUCCESS if the “[Slapi_Value](#)” on page 270 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.

Description

This function adds an array of “[Slapi_Value](#)” on page 270 data values to an attribute. If the attribute does not exist, it is created and given the value contained in the “[Slapi_Value](#)” on page 270 array.

This function replaces the deprecated `slapi_entry_add_values()` function. This function uses “[Slapi_Value](#)” on page 270 attribute values instead of the “[berval](#)” on page 249 attribute values.

Memory Concerns

This routine makes a copy of the parameter `vals`. `vals` can be NULL.

`slapi_entry_add_valueset()`

Add a “[Slapi_ValueSet](#)” on page 270 data value to the specified attribute in an entry.

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> | “ Slapi_ValueSet ” on page 270 data value that you want to add to the entry. |

Returns

Returns 0 when successful; any other value returned signals failure.

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](#)” on page 270 data type. If the entry does not contain the attribute specified, it is created with the specified “[Slapi_ValueSet](#)” on page 270 value.

Memory Concerns

This routine makes a copy of the parameter `vs`. `vs` can be `NULL`.

`slapi_entry_alloc()`

Allocates memory for a new entry of the data type “[Slapi_Entry](#)” on page 263 .

Syntax

```
#include "slapi-plugin.h"
Slapi_Entry *slapi_entry_alloc();
```

Returns

Returns a pointer to the newly allocated entry of the data type “[Slapi_Entry](#)” on page 263. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Description

This function returns an empty “[Slapi_Entry](#)” on page 263 structure.

Memory Concerns

When you are no longer using the entry, you should free it from memory by calling the “[slapi_entry_free\(\)](#)” on page 368 function.

See Also

“[slapi_entry_add_string\(\)](#)” on page 352

“[slapi_entry_add_value\(\)](#)” on page 352

“[slapi_entry_add_values_sv\(\)](#)” on page 353

“[slapi_entry_add_valueset\(\)](#)” on page 354

“[slapi_entry_dup\(\)](#)” on page 366

[“slapi_entry_free\(\)” on page 368](#)

[“slapi_entry_set_dn\(\)” on page 378](#)

[“slapi_entry_set_sdn\(\)” on page 379](#)

[“slapi_str2entry\(\)” on page 528](#)

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:

e Entry from which you want to delete the attribute.
type 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.

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:

- e* Entry that you want to check.
- type* Name of the attribute that you want to check.
- attr* Pointer to the attribute, if the attribute is in the entry.

Returns

This function returns 0 if the entry contains the specified attribute; otherwise it returns -1.

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\(\)](#)” on page 301, to avoid dangling pointers if the entry is freed while the pointer to *attr* is still being used.

See Also

“[slapi_attr_dup\(\)](#)” on page 301

slapi_entry_attr_get_charptr()

Gets the first value of an attribute in 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:

- e* Entry from which you want to get the string value.
- type* Attribute type from which you want to get the value.

Returns

This function returns a copy of the first value in the attribute, or NULL if the entry does not contain the attribute.

Memory Concerns

When you are done working with this value, you should free it from memory by calling the “[slapi_ch_free\(\)](#)” on [page 329](#) function.

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:

- e* Entry from which you want to get the integer value.
- type* Attribute type from which you want to get the value.

Returns

Returns the first value in the attribute converted to an integer or 0 if the entry does not contain the attribute.

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:

- e* Entry from which you want to get the long value.
- type* Attribute type from which you want to get the value.

Returns

This function returns the first value in the attribute converted to a long type. The function returns 0 if the entry does not contain the attribute specified.

slapi_entry_attr_get_uint()

Gets the first value of an attribute in an entry as a 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:

- e* Entry from which you want to get the value.
- type* Attribute type from which you want to get the value.

Returns

This function returns the first value in the attribute converted to an unsigned integer. The function returns 0 if the entry does not contain the attribute specified.

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:

- e* Entry from which you want to get the value.
- type* Attribute type from which you want to get the value.

Returns

This function returns the first value in the attribute converted to an unsigned long. The function returns 0 if the entry does not contain the attribute specified.

slapi_entry_attr_hasvalue()

This function is deprecated. It determines if an attribute in an entry contains a specified value by comparing the specified value as a string with the existing values, and does not compare using the equality matching rule.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_attr_hasvalue(Slapi_Entry *e, const char *type,
    const char *value);
```

Parameters

This function takes the following parameters:

- e* Entry that you want to check.
- type* Attribute type that you want to test for the value specified.
- value* Value that you want to find in the attribute.

Returns

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.

slapi_entry_attr_merge_sv()

Adds an array of “[Slapi_Value](#)” on page 270 data values to the existing attribute values in an entry. If the attribute does not exist, it is created with the “[Slapi_Value](#)” on page 270 specified.

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:

- e* Entry to which you want to add values.
- type* Attribute to which you want to add values.
- vals* Array of “[Slapi_Value](#)” on page 270 data values you want to add.

Returns

Returns 0 if successful; any other value returned signals failure.

Description

This function replaces the deprecated `slapi_entry_attr_merge()` function. This function uses “[Slapi_Value](#)” on page 270 attribute values instead of the “[berval](#)” on page 249 attribute values.

Memory Concerns

This function makes a copy of the parameter `vals`. `vals` can be NULL.

`slapi_entry_attr_replace_sv()`

Replaces the values of an attribute with the “[Slapi_Value](#)” on page 270 data value you specify.

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:

- e* Entry in which you want to replace values.
- type* Attribute type which will receive the replaced values.

vals Array containing the “[Slapi_Value](#)” on page 270 values that should replace the existing values of the attribute.

Returns

This function returns 0 when successful; any other value returned signals failure.

Description

This function replaces existing attribute values in a specified entry with a single “[Slapi_Value](#)” on page 270 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](#)” on page 270 attribute values instead of the “[berval](#)” on page 249 attribute values.

Memory Concerns

This function makes a copy of the parameter `vals`. `vals` can be NULL.

`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:

e Entry in which you want to set the value.
type Attribute type in which you want to set the value.
value 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, and if so, this function is roughly equivalent to “[slapi_entry_attr_delete\(\)](#)” on page 356.

See Also

[“slapi_entry_attr_delete\(\)” on page 356](#)

slapi_entry_attr_set_int()

Replaces the value or values of an attribute in an entry with a specified integer data value.

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:

- e* Entry in which you want to set the value.
- type* Attribute type in which you want to set the value.
- l* Integer value that you want assigned to the attribute.

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.

slapi_entry_attr_set_long()

Replaces the value or values of an attribute in an entry with a specified long data type value.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_attr_set_long(Slapi_Entry* e, const char *type,
                              unsigned long l);
```

Parameters

This function takes the following parameters:

- e* Entry in which you want to set the value.
- type* Attribute type in which you want to set the value.

l Long integer value that you want assigned to the attribute.

slapi_entry_attr_set_uint()

Replaces the value or values of an attribute in an entry with a specified unsigned integer data type value.

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:

e Entry in which you want to set the value.

type Attribute type in which you want to set the value.

l Unsigned integer value that you want assigned to the attribute.

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.

slapi_entry_attr_set_ulong()

Replaces the value or values of an attribute in an entry with a specified unsigned long data type value.

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:

e Entry in which you want to set the value.

type Attribute type in which you want to set the value.
l Unsigned long value that you want assigned to the attribute.

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.

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:

e Entry from which you want the string deleted.
type Attribute type from which you want the string deleted.
value Value of string to delete.

Returns

Returns 0 when successful; any other value returned signals failure.

slapi_entry_delete_values_sv()

Removes an array of “[Slapi_Value](#)” on page 270 data values from an attribute in an entry.

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:

- e* Entry from which you want to delete values.
- type* Attribute from which you want to delete values.
- vals* Array of “[Slapi_Value](#)” on page 270 data values that you want to delete.

Returns

Returns LDAP_SUCCESS if the specified attribute and the array of “[Slapi_Value](#)” on page 270 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 . Additionally, 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 Sun support services.

Description

This function removes an attribute/value set from an entry. Notice that both the attribute and its “[Slapi_Value](#)” on page 270 data values are removed from the entry. If you supply a “[Slapi_Value](#)” on page 270 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](#)” on page 270 attribute values instead of the “[berval](#)” on page 249 attribute values.

Memory Concerns

The `vals` parameter can be NULL, in which case, this function does nothing.

`slapi_entry_dup()`

Makes a copy of an entry, its DN, and its attributes.

Syntax

```
#include "slapi-plugin.h"
Slapi_Entry *slapi_entry_dup( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

e 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.

Description

This function returns a copy of an existing “[Slapi_Entry](#)” on page 263 structure. You can call other front-end functions to change the DN and attributes of this entry.

Memory Concerns

When you are no longer using the entry, you should free it from memory by calling the “[slapi_entry_free\(\)](#)” on page 368 function.

See Also

“[slapi_entry_alloc\(\)](#)” on page 355

“[slapi_entry_free\(\)](#)” on page 368

`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\(\)](#)” on page 376 function.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_first_attr( Slapi_Entry *e, Slapi_Attr **attr );
```

Parameters

This function takes the following parameters:

e Entry from which you want to get the attribute.

attr 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\(\)`” on page 301](#).

See Also

[“`slapi_attr_dup\(\)`” on page 301](#)

slapi_entry_free()

Frees an entry, its DN, and its attributes from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_free( Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

e Entry that you want to free. If NULL, no action occurs.

Description

Call this function to free an entry that you have allocated by using the [“`slapi_entry_alloc\(\)`” on page 355](#) function or the [“`slapi_entry_dup\(\)`” on page 366](#) function.

Memory Concerns

To free entries, always use this function, as opposed to using [“`slapi_ch_free\(\)`” on page 329](#), or `free()`.

See Also

[“`slapi_entry_alloc\(\)`” on page 355](#)

[“`slapi_entry_dup\(\)`” on page 366](#)

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. Notice that 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\(\)” on page 378](#).

Memory Concerns

Use [“slapi_ch_free\(\)” on page 329](#) if you are replacing the DN with [“slapi_entry_set_dn\(\)” on page 378](#).

See Also

[“slapi_ch_free\(\)” on page 329](#)

[“slapi_entry_set_dn\(\)” on page 378](#)

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 the DN of the entry that you specify. The DN is returned as a `const`; you are not able to modify the DN value. If the DN of the “[Slapi_DN](#)” on page 263 object is `NULL`, the NDN value of “[Slapi_DN](#)” on page 263 is returned.

Memory Concerns

Never free this value.

`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.

`slapi_entry_get_sdn()`

Returns the “[Slapi_DN](#)” on page 263 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](#)” on page 263 object.

Returns

This function returns the “[Slapi_DN](#)” on page 263 object from the entry that you specify.

Memory Concerns

Never free this value. If you need a copy, use “[slapi_sdn_dup\(\)](#)” on page 498.

See Also

“[slapi_sdn_dup\(\)](#)” on page 498

slapi_entry_get_sdn_const()

Returns as a const the value of the “[Slapi_DN](#)” on page 263 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:

e Entry from which you want to get the “[Slapi_DN](#)” on page 263 object.

Returns

Returns as a const the value of the “[Slapi_DN](#)” on page 263 object from the entry that you specify.

Memory Concerns

Never free this value. If you need a copy, use [“slapi_sdn_dup\(\)”](#) on page 498.

See Also

[“slapi_sdn_dup\(\)”](#) on page 498

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:

e 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\(\)”](#) on page 332.

See Also

[“slapi_ch_strdup\(\)”](#) on page 332

slapi_entry_has_children()

This function determines if the specified entry has child entries in the backend where it resides.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_has_children( const Slapi_Entry *e );
```

Parameters

This function takes the following parameter:

e Entry that you want to test for child entries.

Returns

This function returns 1 if the entry you supply has child entries in the backend where it resides; otherwise it returns 0. Notice that if a subsuffix is in another backend, this function does not find children contained in that subsuffix.

slapi_entry_init()

Initializes the values of an entry with the DN and attribute value pairs you supply.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_init(Slapi_Entry *e, char *dn, Slapi_Attr *a);
```

Parameters

This function takes the following parameters:

e The entry you want to initialize.

dn The DN of the entry you are initializing.

a Initialization list of attribute value pairs, supplied as a “[Slapi_Attr](#)” on page 259 data value.

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](#)” on page 263 pointer.

Memory Concerns

This function should always be used after “[slapi_entry_alloc\(\)](#)” on page 355, 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:

```
char *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 is eventually be freed when “[slapi_entry_free\(\)](#)” on [page 368](#) is called.

See Also

“[slapi_ch_strdup\(\)](#)” on [page 332](#)

“[slapi_entry_alloc\(\)](#)” on [page 355](#)

“[slapi_entry_free\(\)](#)” on [page 368](#)

slapi_entry_isroot()

Identifies whether the entry having the specified DN is a root DN (directory super user).

Syntax

```
#include "slapi-plugin.h"
into slapi_entry_isroot( const char *dn );
```

Parameters

This function takes the following parameter:

dn The DN of the entry to check.

Returns

This function returns one of the following values:

`0` The entry with the specified DN is that of a root user (directory super user).

- 1 The entry with the specified DN is not that of a root user.

slapi_entry_merge_values_sv()

Merges (adds) an array of “[Slapi_Value](#)” on page 270 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.

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:

- e* Entry into which you want to merge values.
- type* Attribute type that contains the values you want to merge.
- vals* Values that you want to merge into the entry. Values are of type “[Slapi_Value](#)” on page 270.

Returns

This function returns either LDAP_SUCCESS or LDAP_NO_SUCH_ATTRIBUTE.

Description

This function adds additional “[Slapi_Value](#)” on page 270 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](#)” on page 270 specified and returns LDAP_NO_SUCH_ATTRIBUTE.

Notice that if this function fails, it leaves the values for type within a pointer to e in an indeterminate state. The present value set may be truncated.

Memory Concerns

This function makes a copy of vals. vals can be NULL.

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\(\)](#)” on page 367 function.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_next_attr( 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 <code>prevattr</code> in the entry. |

Returns

This function returns 0 if successful or -1 if `prevattr` was the last attribute in the entry.

Memory Concerns

Never free the returned `attr`. Use “[slapi_attr_dup\(\)](#)” on page 301 to make a copy if a copy is needed.

See Also

“[slapi_attr_dup\(\)](#)” on page 301

slapi_entry_rdn_values_present()

Determines if 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( Slapi_Entry *e );
```


Parameters

This function takes the following parameter:

e Entry from which you want to get the attribute.

Returns

This function returns 1 if the values in the RDN are present in attributes of the entry or 0 if the values are not present.

slapi_entry_schema_check()

Determines whether or not 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:

pb Parameter block.

e Entry of which you want to check the schema.

Returns

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 used only to get the `SLAPI_IS_REPLICATED_OPERATION` flag. If that flag is present, no schema checking is done.

slapi_entry_schema_check_ext()

Determines whether or not the proposed modifications to the specified entry comply with the schema for the entry's object class. This function does not check existing attributes not affected by the modifications.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_schema_check_ext( Slapi_PBlock *pb, Slapi_Entry *e,
    LDAPMod **mods );
```

Parameters

This function takes the following parameters:

- pb* Parameter block.
- e* Entry of which you want to check the schema.
- mods* Pointer to the modify structure whose attribute values are to be checked.

Returns

Returns one of the following values:

- 0 if the proposed modifications to the entry comply 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 modifications cause the `objectClass` attribute or other required attributes to be missing, if the modifications add any attributes not allowed by the schema, or if the modifications result in multiple values for a single-valued attribute.

Memory Concerns

The *pb* argument can be NULL. It is used only to get the `SLAPI_IS_REPLICATED_OPERATION` flag. If that flag is present, no schema checking is done.

slapi_entry_set_dn()

Sets the distinguished name (DN) of an entry.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_set_dn( Slapi_Entry *e, char *dn );
```

Parameters

This function takes the following parameters:

- e* Entry to which you want to assign the DN.
- dn* Distinguished name you want assigned to the entry.

Description

This function sets a pointer to the DN supplied in the specified entry.

Memory Concerns

dn is freed when “[slapi_entry_free\(\)](#)” on page 368 is called.

A copy of *dn* should be passed, for example:

```
char *dn = slapi_ch_strdup(some_dn);
slapi_entry_set_dn(e, dn);
```

The old *dn* is freed as a result of this call. Do not pass in a NULL value.

See Also

“[slapi_entry_free\(\)](#)” on page 368

“[slapi_entry_get_dn\(\)](#)” on page 369

“[slapi_entry_set_dn\(\)](#)” on page 378

slapi_entry_set_sdn()

Sets the “[Slapi_DN](#)” on page 263 value in an entry.

Syntax

```
#include "slapi-plugin.h"
void slapi_entry_set_sdn( Slapi_Entry *e, const Slapi_DN *sdn );
```

Parameters

This function takes the following parameters:

- e* Entry to which you want to set the value of the “[Slapi_DN](#)” on page 263.
- sdn* The specified “[Slapi_DN](#)” on page 263 value that you want to set.

Description

This function sets the value for the “[Slapi_DN](#)” on page 263 object in the entry you specify.

Memory Concerns

This function makes a copy of the *sdn* argument.

`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.

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 the size of the entry, rounded to the nearest 1k. The value returned is a `size_t` data type, with is a `u_long` value. If the entry is empty, a size of 1k is returned.

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 DN's, or any metadata) are not included in the size returned. It is assumed that the size of the metadata 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).

Notice that when determining the size of the entry, both deleted values and deleted attributes are included in the count.

slapi_entry_syntax_check()

This function determines whether the values of attributes present on the specified entry comply with attribute syntax rules.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_syntax_check( Slapi_PBlock *pb, Slapi_Entry *e );
```

Parameters

This function takes the following parameters:

- pb* Parameter block.
- e* Entry whose attributes to check for syntax compliance.

Returns

Returns one of the following values:

- 0 if the entry attribute values comply or if syntax checking is turned off.
- 1 if the entry attribute values do not comply with attribute syntax rules.

Memory Concerns

The *pb* argument can be NULL. It is used only to get the SLAPI_IS_REPLICATED_OPERATION flag. If that flag is present, no syntax checking is done.

slapi_entry_vattr_find()

This function determines whether the specified virtual attribute is present, and returns that attribute if available.

Syntax

```
#include "slapi-plugin.h"
int slapi_entry_vattr_find(const Slapi_Entry *e,
    const char *type, Slapi_Attr **a, int *buffer_flags);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to check for the virtual attribute.
<i>type</i>	Attribute type of the virtual attribute.
<i>a</i>	Structure to hold the virtual attribute.
<i>buffer_flags</i>	Bitmask indicated whether the caller needs to free the attribute.

Returns

Returns one of the following values:

- 0 if the entry contains the virtual attribute specified by *type*.
In this case *a* points to the virtual attribute, and *buffer_flags* holds `SLAPI_VIRTUALATTRS_RETURNED_COPIES` meaning the structure *a* must be freed, or `SLAPI_VIRTUALATTRS_RETURNED_POINTERS` and it should not be freed.
- -1 if the entry does not contain the attribute.

slapi_filter_compare()

Determines if two filters are identical.

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 0 if the two filters are identical, or a value other than 0 if they are not.

Description

This function allows you to determine if two filters are identical, and/or are allowed to be in a different order.

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).

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 <i>f</i> . |

Description

This function frees the filter in parameter *f*.

Memory Concerns

Filters created using “[slapi_str2filter\(\)](#)” on page 529 must be freed after using this function. Filters extracted from a parameter block using:

```
slapi_pblock_get( pb, SLAPI_SEARCH_FILTER, &filter );
```

must not be freed.

See Also

“[slapi_pblock_get\(\)](#)” on page 462

“[slapi_str2filter\(\)](#)” on page 529

slapi_filter_get_attribute_type()

Gets the attribute type for all simple filter choices.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_attribute_type( Slapi_Filter *f, char **type );
```

Parameters

This function takes the following parameters:

- f* Filter from which you wish to get the substring values.
- type* Pointer to the attribute type of the filter.

Returns

This function returns the attribute type of the filter.

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.

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](#)” on [page 265](#) structure when “[slapi_filter_free\(\)](#)” on [page 383](#) is called.

See Also

[“slapi_filter_get_choice\(\)” on page 386](#)

[“slapi_filter_get_ava\(\)” on page 385](#)

[“slapi_filter_get_type\(\)” on page 388](#)

[“slapi_filter_free\(\)” on page 383](#)

slapi_filter_get_ava()

(Applies only to filters of the types LDAP_FILTER_EQUALITY, LDAP_FILTER_GE, LDAP_FILTER_LE, LDAP_FILTER_APPROX) Gets the attribute type and the value from the filter.

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:

- f* Filter from which you wish to get the attribute and value.
- type* Pointer to the attribute type of the filter.
- bval* Pointer to the address of the [“berval” on page 249](#) structure containing the value of the filter.

Returns

This function returns 0 if successful, or -1 if the filter is not one of the types listed above.

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*.

Memory Concerns

The strings within the parameters `type` and `bval` are direct pointers to memory inside the “[Slapi_Filter](#)” on page 265 , and therefore should not be freed after usage. They will be freed when a server entity calls “[slapi_filter_free\(\)](#)” on page 383 after usage of the “[Slapi_Filter](#)” on page 265 structure.

See Also

“[slapi_filter_get_choice\(\)](#)” on page 386

“[slapi_filter_get_type\(\)](#)” on page 388

“[slapi_filter_get_attribute_type\(\)](#)” on page 384

`slapi_filter_get_choice()`

Gets the type of the specified filter such as `LDAP_FILTER_EQUALITY`, for example.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_choice( Slapi_Filter *f );
```

Parameters

This function takes the following parameter:

f Filter type that you wish to get.

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\(\)” on page 388](#)

[“slapi_filter_get_attribute_type\(\)” on page 384](#)

[“slapi_filter_get_ava\(\)” on page 385](#)

slapi_filter_get_subfilt()

(Applies only to filters of the type LDAP_FILTER_SUBSTRINGS) Gets the substring values from the filter.

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 from which you wish to get the substring values.
<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.

final 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.

Description

Filters of the type `LDAP_FILTER_SUBSTRINGS` generally compare a set of substrings against an attribute. For example:

```
(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\(\)” on page 384](#)

[“slapi_filter_get_ava\(\)” on page 385](#)

[“slapi_filter_get_choice\(\)” on page 386](#)

slapi_filter_get_type()

(Applies only to filters of the type `LDAP_FILTER_PRESENT`) Gets the attribute type specified in the filter.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_get_type( Slapi_Filter *f, char **type );
```

Parameters

This function takes the following parameters:

f Filter from which you want to get the substring values.

type Pointer to the attribute type of the filter.

Returns

This function returns 0 if successful, or -1 if the filter is not one of the types listed above.

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`.

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](#)” on page 265 is freed by calling “[slapi_filter_free\(\)](#)” on page 383.

See Also

“[slapi_filter_get_attribute_type\(\)](#)” on page 384

“[slapi_filter_get_ava\(\)](#)” on page 385

“[slapi_filter_get_choice\(\)](#)” on page 386

`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`.

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:

- f_{type}* Type of composite filter you want to create.
- f₁* First filter that you want to join.
- f₂* Second filter that you want to join. If *f_{type}* is LDAP_FILTER_NOT, specify NULL for this argument.

Returns

This function returns the new filter constructed from the other two filters.

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.

Memory Concerns

The *f₁* and *f₂* filters are not copied, nor freed, during the join process, but the resulting filter will have references pointing to these two filters.

`slapi_filter_list_first()`

(Applies only to filters of the types LDAP_FILTER_EQUALITY, LDAP_FILTER_GE, LDAP_FILTER_LE, LDAP_FILTER_APPROX) Gets the first filter that makes up the specified filter.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_list_first( Slapi_Filter *f );
```

Parameters

This function takes the following parameter:

f Filter of which you wish to get the first component.

Returns

The first filter 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_next\(\)” on page 391](#) 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 this function to get the first filter in the 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_next\(\)” on page 391](#)

slapi_filter_list_next()

(Applies only to filters of the types LDAP_FILTER_EQUALITY, LDAP_FILTER_GE, LDAP_FILTER_LE, LDAP_FILTER_APPROX) Gets the next filter (following *fprev*) that makes up the specified filter *f*.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_filter_list_next(Slapi_Filter *f,
    Slapi_Filter *fprev);
```

Parameters

This function takes the following parameters:

- f* Filter from which you want to get the next component (after *fprev*).
- fprev* Filter within the specified filter *f*.

Returns

The next filter (after *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\(\)`” on page 390](#) 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 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 the filter should not be freed independently of the original filter.

See Also

[“`slapi_filter_list_first\(\)`” on page 390](#)

`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

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\(\)” on page 394](#)

[“slapi_filter_test_ext\(\)” on page 393](#)

slapi_filter_test_ext()

Determines if an entry matches a given filter.

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>	Parameter block 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.

only_test_access 1 when access checking must be done.
 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.

Description

This function allows you to determine if an entry matches a given filter, or that the current user has the permission to access the entry.

See Also

[“slapi_filter_test_simple\(\)” on page 394](#)

[“slapi_filter_test\(\)” on page 392](#)

slapi_filter_test_simple()

Determines if an entry matches a filter.

Syntax

```
#include "slapi-plugin.h"
int slapi_filter_test_simple( Slapi_Entry *e, Slapi_Filter *f);
```

Parameters

This function takes the following parameters:

e Entry that you wish to test.
f 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.

Description

This function allows you to check if entry *e* matches filter *f*.

See Also

[“slapi_filter_test\(\)” on page 392](#)

[“slapi_filter_test_ext\(\)” on page 393](#)

slapi_find_matching_paren()

Find a right parenthesis matching a left parenthesis.

Syntax

```
#include "slapi-plugin.h"
char *slapi_find_matching_paren( const char *str );
```

Parameters

This function takes the following parameters:

str Pointer to a string starting with a left parenthesis

Returns

This function returns a pointer to the right parenthesis if successful. Otherwise, it returns NULL indicating that no matching right parenthesis was found.

Description

This function takes a pointer to a string starting with a left parenthesis, (, and returns a pointer to the matching right parenthesis,). It may be useful when evaluating complex search filter strings.

slapi_free_search_results_internal()

Frees search results returned by the “[slapi_search_internal_pb\(\)](#)” on page 521 and “[slapi_search_internal_callback_pb\(\)](#)” on page 518 functions.

Syntax

```
#include "slapi-plugin.h"
void slapi_free_search_results_internal(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

pb Parameter block returned by the “[slapi_search_internal_pb\(\)](#)” on page 521 and “[slapi_search_internal_callback_pb\(\)](#)” on page 518 functions.

Description

This function must be called when you are finished with the entries before freeing the parameter block.

slapi_free_suffix_list()

Free a list of directory suffixes, such as a list obtained using “[slapi_get_suffix_list\(\)](#)” on page 399.

Syntax

```
#include "slapi-plugin.h"
void slapi_free_suffix_list(Slapi_DN ** suffix_list);
```

Parameters

This function takes the following parameters:

suffix_list Array of Distinguished Names for the suffixes to free.

Description

This function frees each entry in *suffix_list*, and the *suffix_list* itself. It does not remove data from a database associated with the suffix.

slapi_get_first_backend()

Returns a pointer of the backend structure of the first backend.

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend* slapi_get_first_backend(char **cookie);
```

Parameters

This function takes the following parameter:

cookie Output parameter containing the index of the returned backed. This is useful for calls to [“slapi_get_next_backend\(\)” on page 398](#). Contains 0 in output if no backend is returned.

Returns

This function returns a pointer to the backend structure of the first backend, and its index, in the cookie parameter, or NULL if there is no 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\(\)” on page 398](#). 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);
```

Memory Concerns

Free the cookie parameter after the iteration using [“slapi_ch_free\(\)” on page 329](#).

See Also

[“slapi_get_next_backend\(\)” on page 398](#)

slapi_get_object_extension()

Access an object extension.

Syntax

```
#include "slapi-plugin.h"
void *slapi_get_object_extension(int objecttype, void *object,
    int extensionhandle);
```

Parameters

This function takes the following parameters:

<code>objecttype</code>	Type set by the server
<code>object</code>	Pointer to the object you extended
<code>extensionhandle</code>	Handle set by the server

Description

This function returns a pointer to the object extension registered using [“slapi_register_object_extension\(\)” on page 488](#).

See Also

[“slapi_register_object_extension\(\)” on page 488](#)

slapi_get_next_backend()

Returns a pointer to the next backend.

Syntax

```
#include "slapi-plugin.h"
Slapi_Backend* slapi_get_next_backend(char *cookie);
```

Parameters

This function takes the following parameters:

<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 a pointer to the next backend, if it exists, and updates the cookie parameter. Otherwise, it returns NULL and cookie is not changed.

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\(\)](#)” on page 397. 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);
```

Memory Concerns

Free the cookie parameter after the iteration using “[slapi_ch_free\(\)](#)” on page 329.

See Also

“[slapi_get_first_backend\(\)](#)” on page 397

“[slapi_ch_free\(\)](#)” on page 329

slapi_get_suffix_list()

Returns an array of suffix DN's handled by the server.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN ** slapi_get_suffix_list(int show_private, int *count);
```

Parameters

This function takes the following parameters:

<i>show_private</i>	If set to 1, this list of DNs returned includes suffixes used by Directory Server that do not contain user data. Otherwise, only DNs of suffixes containing user data are returned.
<i>count</i>	Placeholder to contain the number of suffixes in the list.

Returns

This function returns a pointer to an array of “[Slapi_DN](#)” on page 263 structures containing the base DNs of the suffixes. The *count* parameter contains the number of suffixes whose DNs are returned.

Memory Concerns

Free the list returned using “[slapi_free_suffix_list\(\)](#)” on page 396.

slapi_get_supported_controls_copy()

Retrieves an allocated array of object identifiers (OIDs) representing the controls supported by Directory Server. You can register new controls by calling “[slapi_register_supported_control\(\)](#)” on page 491.

Syntax

```
#include "slapi-plugin.h"
int slapi_get_supported_controls_copy( char ***ctrloidsp,
    unsigned long **ctrllosp );
```

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>ctrllosp</i>	Pointer to an unsigned long array that will receive the supported operation values for each control in the <i>ctrloidsp</i> array. Pass NULL for this parameter if you do not wish to receive the supported operation values.

Returns

This function returns 0 if successful, or a non-zero value if an error occurs.

Description

This function replaces the deprecated `slapi_get_supported_controls()` function from previous releases, as it was not multithread safe.

When you call “[slapi_register_supported_control\(\)](#)” on page 491 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_entry_get_uniqueid\(\)](#)” on page 372.

For each OID returned in the `ctrloidsp` array, the corresponding array element (with the same index) in the `ctrllosp` array identifies the operations that support the control. For a list of the possible IDs for the operations, see “[slapi_register_supported_control\(\)](#)” on page 491.

Memory Concerns

The returned `ctrloidsp` array should be freed by calling “[slapi_ch_array_free\(\)](#)” on page 326. The returned `ctrllosp` array should be freed by calling “[slapi_ch_free\(\)](#)” on page 329.

See Also

“[slapi_register_supported_control\(\)](#)” on page 491

“[slapi_ch_array_free\(\)](#)” on page 326

slapi_get_supported_extended_ops_copy()

Gets a copy of the object IDs (OIDs) of the extended operations.

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.

Description

This function replaces the deprecated `slapi_get_supported_extended_ops()` function from earlier releases, as `slapi_get_supported_extended_ops()` was not multithread 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_pblock_set\(\)`” on page 464](#).

Memory Concerns

The array returned by this function should be freed by calling the [“`slapi_ch_array_free\(\)`” on page 326](#) function.

See Also

[“`slapi_pblock_set\(\)`” on page 464](#)

[“`slapi_ch_array_free\(\)`” on page 326](#)

`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_register_supported_saslmechanism\(\)`” on page 492](#) 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.

`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:

s Pointer to the NULL terminated string to test.

Returns

This function returns 1 if the string contains an 8-bit character, or 0 if it does not.

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 char *dn );
```

Parameters

This function takes the following parameters:

dn The DN that you want to test to see if it is the root DSE entry.

Returns

This function returns 1 if *dn* is the root DSE; otherwise the function returns 0.

slapi_is_root_suffix()

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:

dn DN that you wish 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

slapi_ldap_init()

Get a thread-safe handle to an LDAP connection.

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>	Host on which the LDAP server is running
<i>ldapport</i>	Port on which the LDAP server is listening
<i>secure</i>	1 for a secure connection over SSL, NULL otherwise
<i>shared</i>	If not NULL, then the connection may be shared between threads

Description

This function allows a plug-in to retrieve a thread-safe handle to an LDAP connection. When done with the handle, call `slapi_ldap_unbind()`.

A timeout may be set for the connection using the Directory SDK for C provided as part of Directory Server Resource Kit. [Example 16–1](#) demonstrates how to set a timeout.

EXAMPLE 16–1 Setting a Timeout

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

EXAMPLE 16-1 Setting a Timeout (Continued)

```

void
my_ldap_function(void)
{
    LDAP * ld;
    int    to = 5000;                /* 5000 ms == 5 s 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 for a search for example. */

    slapi_ldap_unbind(ld);
    return 0;
}

```

Returns

This function returns an LDAP connection handle if successful. Otherwise, it returns NULL.

See Also

[“slapi_ldap_unbind\(\)” on page 405](#)

slapi_ldap_unbind()

Release an LDAP connection obtained using [“slapi_ldap_init\(\)” on page 404](#).

Syntax

```

#include "slapi-plugin.h"
void slapi_ldap_unbind( LDAP *ld );

```

Parameters

This function takes the following parameters:

`ld` Handle to the LDAP connection

Description

This function allows a plug-in to release an LDAP connection obtained using [“`slapi_ldap_init\(\)`” on page 404](#).

See Also

[“`slapi_ldap_init\(\)`” on page 404](#)

slapi_ldapmods_syntax_check()

Determines whether the proposed modifications comply with attribute syntax rules.

Syntax

```
#include "slapi-plugin.h"
int slapi_ldapmods_syntax_check( Slapi_PBlock *pb, LDAPMod **mods );
```

Parameters

This function takes the following parameters:

pb Parameter block.

mods Pointer to the modify structure whose attribute values are to be checked.

Returns

Returns one of the following values:

- 0 if the proposed modifications comply with attribute syntax rules, or if syntax checking is turned off.
- 1 if the modifications do not comply with attribute syntax rules.

Memory Concerns

The *pb* argument can be NULL. It is used only to get the `SLAPI_IS_REPLICATED_OPERATION` flag. If that flag is present, no syntax checking is done.

slapi_lock_mutex()

Lock a mutex.

Syntax

```
#include "slapi-plugin.h"
void slapi_lock_mutex( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

`mutex` Mutex for thread synchronization

Description

This function allows thread synchronization. Once a thread has locked a mutex using this function, other threads attempting to acquire the lock are blocked until the thread holding the mutex calls `slapi_UTF-8STRTOLOWER()` on page 534.

See Also

[“slapi_destroy_mutex\(\)” on page 338](#)

[“slapi_new_mutex\(\)” on page 458](#)

[“slapi_UTF-8STRTOLOWER\(\)” on page 534](#)

slapi_log_error_ex()

Write an error message to the server error log.

Syntax

```
#include "slapi-plugin.h"
int slapi_log_error_ex(long errorId,
    long msgId, int connId, int opId, char const * subsystem,
    char const * humanReadableMsg, char const * fmt, /* args */ ...);
```

Parameters

This function takes the following parameters:

`errorId` Unique identifier you provide for this error message

msgId	Identifier for the current message obtained using SLAPI_OPERATION_MSGID
connId	Identifier for the current connection obtained using SLAPI_CONN_ID
opId	Identifier for the current operation obtained using SLAPI_OPERATION_ID
subsystem	String indicating the context in which the warning arose such as the name of the plug-in function logging the message
humanReadableMsg	String message identifying this warning
fmt	Format specification in the style of printf()
args	Arguments for the format specification in fmt

Description

This function writes the specified error message to the server error log in synchronous fashion. This function does not return until the log message has been flushed to disk, thus blocking the server for the duration of the write operation. By default, the error log is \$INSTANCE_PATH/logs/errors.

Unlike “[slapi_log_info_ex\(\)](#)” on [page 409](#), this function cannot be turned off.

Error messages typically concern fatal errors. For warnings, use “[slapi_log_warning_ex\(\)](#)” on [page 411](#). For informational log messages, use “[slapi_log_info_ex\(\)](#)” on [page 409](#).

Example

[Example 16–2](#) shows a call to `slapi_log_error_ex()`.

EXAMPLE 16–2 Logging an Error

```
#include "slapi-plugin.h"
#include "example-com-error-ids.h" /* example.com unique
                                   error IDs file */

int
foobar(Slapi_PBlock * pb)
{
    char * error_cause;
    int    apocalypse = 1;          /* Expect the worst. */

    /* ... */

    if (apocalypse) {               /* Server to crash soon */
        slapi_log_error_ex(
            EXCOM_SERVER_MORIBUND, /* Unique error ID */

```


EXAMPLE 16-2 Logging an Error (Continued)

```

        SLAPI_LOG_NO_MSGID,
        SLAPI_LOG_NO_CONNID,
        SLAPI_LOG_NO_OPID,
        "example.com: foobar in baz plug-in",
        "cannot write to file system: %s\n",
        error_cause
    );
    return -1;
}
return 0;
}

```

Returns

This function returns 0 if successful. Otherwise, it returns -1.

See Also

[“slapi_log_info_ex\(\)” on page 409](#)

[“slapi_log_warning_ex\(\)” on page 411](#)

slapi_log_info_ex()

Write an informational message to the server error log.

Syntax

```

#include "slapi-plugin.h"
int slapi_log_info_ex(slapi_log_info_area_t area,
    slapi_log_info_level_t level,
    long msgId, int connId, int opId, char const * subsystem,
    char const * fmt, /* args */, ...);

```

Parameters

This function takes the following parameters:

area	Identifies the server component so logging can be turned on by adding the decimal value of the area to the value for <code>nsslapd-info-log-area</code> . Subtract from the value to turn informational logging off.
level	Identifies whether the server should log this message when informational logging is activated for area.

When informational logging is activated, setting level to:

- `SLAPI_LOG_INFO_LEVEL_DEFAULT` means always log the message.
- `SLAPI_LOG_INFO_LEVEL_EXTRA` means only log if the value of `nsslapd-info-log-level` is greater than 0.

<code>msgId</code>	Identifier for the current message obtained using <code>SLAPI_OPERATION_MSGID</code>
<code>connId</code>	Identifier for the current connection obtained using <code>SLAPI_CONN_ID</code>
<code>opId</code>	Identifier for the current operation obtained using <code>SLAPI_OPERATION_ID</code>
<code>subsystem</code>	String indicating the context in which the warning arose such as the name of the plug-in function logging the message
<code>fmt</code>	Format specification in the style of <code>printf()</code>
<code>args</code>	Arguments for the format specification in <code>fmt</code>

Description

This function writes the specified error message to the server error log in synchronous fashion. This function does not return until the log message has been flushed to disk, thus blocking the server for the duration of the write operation. By default, the error log is `$INSTANCE_PATH/logs/errors`.

This function is turned off by default. Activate logging of the message with the `dsconf set-log-prop` command.

You can also manage logs using Directory Service Control Center.

Informational message are typically those that system administrators may ignore unless trying to debug server behavior. For errors, use “[slapi_log_error_ex\(\)](#)” on page 407. For warnings, use “[slapi_log_warning_ex\(\)](#)” on page 411.

Example

[Example 16-3](#) shows a call to `slapi_log_info_ex()`.

EXAMPLE 16-3 Logging an Informational Message

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

int
hello()
{
    slapi_log_info_ex(
        SLAPI_LOG_INFO_AREA_PLUGIN,
        SLAPI_LOG_INFO_LEVEL_DEFAULT,
```

EXAMPLE 16-3 Logging an Informational Message *(Continued)*

```

        SLAPI_LOG_NO_MSGID,
        SLAPI_LOG_NO_CONNID,
        SLAPI_LOG_NO_OPID,
        "hello() from a plug-in",
        "Hello, World!\n"
    );
    return 0;
}

```

Returns

This function returns 0 if successful. Otherwise, it returns -1.

See Also

[“slapi_log_error_ex\(\)” on page 407](#)

[“slapi_log_warning_ex\(\)” on page 411](#)

slapi_log_warning_ex()

Write a warning message to the server error log.

Syntax

```

#include "slapi-plugin.h"
int slapi_log_warning_ex(long warningId,
    long msgId, int connId, int opId, char const * subsystem,
    char const * humanReadableMsg, char const * fmt, /* args */ ...);

```

Parameters

This function takes the following parameters:

warningId	Unique identifier you provide for this warning message
msgId	Identifier for the current message obtained using SLAPI_OPERATION_MSGID
connId	Identifier for the current connection obtained using SLAPI_CONN_ID
opId	Identifier for the current operation obtained using SLAPI_OPERATION_ID

subsystem	String indicating the context in which the warning arose such as the name of the plug-in function logging the message
humanReadableMsg	String message identifying this warning
fmt	Format specification in the style of <code>printf()</code>
args	Arguments for the format specification in <code>fmt</code>

Description

This function writes the specified error message to the server error log in synchronous fashion. This function does not return until the log message has been flushed to disk, thus blocking the server for the duration of the write operation. By default, the error log is `$INSTANCE_PATH/logs/errors`.

Unlike [“`slapi_log_info_ex\(\)`” on page 409](#), this function cannot be turned off.

Warning messages typically concern potentially serious situations, but not fatal errors. For fatal errors, use [“`slapi_log_error_ex\(\)`” on page 407](#). For informational log messages, use [“`slapi_log_info_ex\(\)`” on page 409](#).

Example

[Example 16–4](#) shows a call to `slapi_log_warning_ex()`.

EXAMPLE 16–4 Logging a Warning

```
#include "slapi-plugin.h"
#include "example-com-warning-ids.h" /* example.com unique
                                     warning IDs file */

int
foobar()
{
    int disk_use_percentage;

    /* ... */

    if (disk_use_percentage >= 95){
        slapi_log_warning_ex(
            EXCOM_DISK_FULL_WARN, /* unique warning ID */
            SLAPI_LOG_NO_MSGID,
            SLAPI_LOG_NO_CONNID,
            SLAPI_LOG_NO_OPID,
            "example.com: foobar in baz plug-in",
            "disk %.0f%% full, find more space\n",
            (float)disk_use_percentage
        );
    }
```

EXAMPLE 16-4 Logging a Warning (Continued)

```

    }
    return 0;
}

```

Returns

This function returns 0 if successful. Otherwise, it returns -1.

See Also

[“slapi_log_error_ex\(\)” on page 407](#)

[“slapi_log_info_ex\(\)” on page 409](#)

slapi_matchingrule_free()

Free a [“Slapi_MatchingRuleEntry” on page 265](#) after registering the matching rule.

Syntax

```

#include "slapi-plugin.h"
void slapi_matchingrule_free(Slapi_MatchingRuleEntry **mrEntry,
    int freeMembers);

```

Parameters

This function takes the following parameters:

mrEntry	Matching rule registration object
freeMembers	Whether to free the members of the “Slapi_MatchingRuleEntry” on page 265
	If 0, do not free the members of the “Slapi_MatchingRuleEntry” on page 265 .

Description

This function frees memory allocated to a [“Slapi_MatchingRuleEntry” on page 265](#) after the structure has been used to register a matching rule.

See Also

[“slapi_matchingrule_new\(\)” on page 415](#)

[“slapi_matchingrule_register\(\)” on page 415](#)

slapi_matchingrule_get()

Access a [“Slapi_MatchingRuleEntry” on page 265](#) member.

Syntax

```
#include "slapi-plugin.h"
int slapi_matchingrule_get(Slapi_MatchingRuleEntry *mr, int arg,
    void *value);
```

Parameters

This function takes the following parameters:

mr	Matching rule registration object
arg	Identifier for the “Slapi_MatchingRuleEntry” on page 265 member: <ul style="list-style-type: none">▪ SLAPI_MATCHINGRULE_DESC, a string describing the matching rule▪ SLAPI_MATCHINGRULE_NAME, a string identifying the matching rule▪ SLAPI_MATCHINGRULE_OID, a string representing the matching rule object identifier▪ SLAPI_MATCHINGRULE_SYNTAX, the matching rule syntax OID string 1.3.6.1.4.1.1466.115.121.1.15▪ SLAPI_MATCHINGRULE_OBSOLETE, an int identifying whether the rule is obsolete
value	Value retrieved from the member

Description

This function accesses a [“Slapi_MatchingRuleEntry” on page 265](#) member based on the identifier in arg.

Returns

This function returns 0 if successful. Otherwise, it returns -1.

See Also

[“slapi_matchingrule_register\(\)” on page 415](#)

[“slapi_matchingrule_set\(\)” on page 416](#)

slapi_matchingrule_new()

Allocate a [“Slapi_MatchingRuleEntry” on page 265](#).

Syntax

```
#include "slapi-plugin.h"
Slapi_MatchingRuleEntry *slapi_matchingrule_new(void);
```

Description

This function allocates a [“Slapi_MatchingRuleEntry” on page 265](#) used to register a matching rule.

Returns

This function returns a pointer to the matching rule registration object if successful. Otherwise, it returns NULL.

See Also

[“slapi_matchingrule_free\(\)” on page 413](#)

[“slapi_matchingrule_register\(\)” on page 415](#)

slapi_matchingrule_register()

Register a matching rule with the server.

Syntax

```
#include "slapi-plugin.h"
int slapi_matchingrule_register(Slapi_MatchingRuleEntry *mrEntry);
```

Parameters

This function takes the following parameters:

mrEntry Matching rule registration object

Description

This function registers a “[Slapi_MatchingRuleEntry](#)” on page 265 with the server. Register matching rules as part of the plug-in initialization function.

First, allocate the structure using “[slapi_matchingrule_new\(\)](#)” on page 415. Next, set the members of the matching rule entry using “[slapi_matchingrule_set\(\)](#)” on page 416. After setting the members, register the matching rule with the server using this function. Finally, free the memory allocated using “[slapi_matchingrule_free\(\)](#)” on page 413.

Returns

This function returns 0 if successful. Otherwise, it returns -1.

See Also

“[slapi_matchingrule_free\(\)](#)” on page 413

“[slapi_matchingrule_get\(\)](#)” on page 414

“[slapi_matchingrule_new\(\)](#)” on page 415

“[slapi_matchingrule_set\(\)](#)” on page 416

slapi_matchingrule_set()

Modify a “[Slapi_MatchingRuleEntry](#)” on page 265 member.

Syntax

```
#include "slapi-plugin.h"
int slapi_matchingrule_set(Slapi_MatchingRuleEntry *mr, int arg,
    void *value);
```

Parameters

This function takes the following parameters:

- | | |
|-----|---|
| mr | Matching rule registration object |
| arg | Identifier for the “ Slapi_MatchingRuleEntry ” on page 265 member: <ul style="list-style-type: none">▪ SLAPI_MATCHINGRULE_DESC, a string describing the matching rule▪ SLAPI_MATCHINGRULE_NAME, a string identifying the matching rule |

- `SLAPI_MATCHINGRULE_OID`, a string representing the matching rule object identifier
- `SLAPI_MATCHINGRULE_SYNTAX`, the matching rule syntax OID string `1.3.6.1.4.1.1466.115.121.1.15`
- `SLAPI_MATCHINGRULE_OBSOLETE`, an `int` identifying whether the rule is obsolete

value Value to affect to the member

Description

This function modifies a “[Slapi_MatchingRuleEntry](#)” on page 265 member based on the identifier in *arg*.

Returns

This function returns 0 if successful. Otherwise, it returns -1.

See Also

“[slapi_matchingrule_get\(\)](#)” on page 414

“[slapi_matchingrule_register\(\)](#)” on page 415

`slapi_mod_add_value()`

Adds a value to a “[Slapi_Mod](#)” on page 265 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:

smod Pointer to an initialized “[Slapi_Mod](#)” on page 265.

val Pointer to a “[berval](#)” on page 249 representing the attribute value.

Description

Adds a copy of a given attribute to the “[Slapi_Mod](#)” on page 265.

slapi_mod_done()

Frees the internals of “[Slapi_Mod](#)” on page 265 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_done(Slapi_Mod *mod);
```

Parameters

This function takes the following parameter:

mod Pointer to a “[Slapi_Mod](#)” on page 265.

Description

This function frees the internals of a “[Slapi_Mod](#)” on page 265, leaving it in the uninitialized state.

Memory Concerns

Use this function on a stack-allocated “[Slapi_Mod](#)” on page 265 when you have finished with it, or wish to reuse it.

See Also

“[slapi_mod_init\(\)](#)” on page 424

“[slapi_mod_init_byref\(\)](#)” on page 425

“[slapi_mod_init_byval\(\)](#)” on page 425

“[slapi_mod_init_passin\(\)](#)” on page 426

slapi_mod_dump()

Dumps the contents of an “[LDAPMod](#)” on page 250 to the server log.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_dump(LDAPMod *mod, int n);
```

Parameters

This function takes the following parameters:

- mod* Pointer to an “[LDAPMod](#)” on page 250.
- n* Numeric label that will be included in the log.

Description

This function uses the LDAP_DEBUG_ANY log level to dump the contents of an “[LDAPMod](#)” on page 250 to \$INSTANCE_PATH/logs/errors for debugging.

slapi_mod_free()

Frees a “[Slapi_Mod](#)” on page 265 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_free(Slapi_Mod **smod);
```

Parameters

This function takes the following parameter:

- smod* Pointer to an initialized “[Slapi_Mod](#)” on page 265.

Description

This function frees a “[Slapi_Mod](#)” on page 265 structure that was allocated by “[slapi_mod_new\(\)](#)” on page 427.

See Also

“[slapi_mod_new\(\)](#)” on page 427

slapi_mod_get_first_value()

Initializes a “[Slapi_Mod](#)” on page 265 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:

smod Pointer to an initialized “[Slapi_Mod](#)” on page 265.

Returns

This function returns a pointer to the first attribute value in the “[Slapi_Mod](#)” on page 265, or NULL if no values exist.

Description

Use this function with “[slapi_mod_get_next_value\(\)](#)” on page 421 to iterate through the attribute values in a “[Slapi_Mod](#)” on page 265 structure.

See Also

“[slapi_mod_get_next_value\(\)](#)” on page 421

slapi_mod_get_ldapmod_byref()

Gets a reference to the “[LDAPMod](#)” on page 250 in a “[Slapi_Mod](#)” on page 265 structure.

Syntax

```
#include "slapi-plugin.h"
const LDAPMod *slapi_mod_get_ldapmod_byref(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

smod Pointer to an initialized “[Slapi_Mod](#)” on page 265.

Returns

This function returns a pointer to a read-only “[LDAPMod](#)” on page 250 owned by the “[Slapi_Mod](#)” on page 265.

Description

Use this function to get direct access to the “[LDAPMod](#)” on page 250 contained in a “[Slapi_Mod](#)” on page 265.

Memory Concerns

Responsibility for the “LDAPMod” on page 250 remains with the “Slapi_Mod” on page 265.

See Also

“slapi_mod_get_ldapmod_passout()” on page 421

slapi_mod_get_ldapmod_passout()

Retrieves the “LDAPMod” on page 250 contained in a “Slapi_Mod” on page 265 structure.

Syntax

```
#include "slapi-plugin.h"
LDAPMod *slapi_mod_get_ldapmod_passout(Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

smod Pointer to an initialized “Slapi_Mod” on page 265.

Returns

This function returns a pointer to an “LDAPMod” on page 250 owned by the caller.

Description

Use this function to get the “LDAPMod” on page 250 out of a “Slapi_Mod” on page 265.

Memory Concerns

Responsibility for the “LDAPMod” on page 250 transfers to the caller. The “Slapi_Mod” on page 265 is left in the uninitialized state.

See Also

“slapi_mod_get_ldapmod_byref()” on page 420

slapi_mod_get_next_value()

Increments the “Slapi_Mod” on page 265 iterator and returns the next attribute value.

Syntax

```
#include "slapi-plugin.h"
struct berval *slapi_mod_get_next_value(Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

smod Pointer to an initialized [“Slapi_Mod” on page 265](#).

Returns

This function returns a pointer to the next attribute value in the [“Slapi_Mod” on page 265](#), or NULL if there are no more.

Description

Use this function with [“slapi_mod_get_first_value\(\)” on page 419](#) to iterate through the attribute values in a [“Slapi_Mod” on page 265](#).

See Also

[“slapi_mod_get_first_value\(\)” on page 419](#)

slapi_mod_get_num_values()

Gets the number of values in a [“Slapi_Mod” on page 265](#) structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_mod_get_num_values(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

smod Pointer to an initialized [“Slapi_Mod” on page 265](#).

Returns

This function returns the number of attribute values in the [“Slapi_Mod” on page 265](#).

slapi_mod_get_operation()

Gets the operation type of “[Slapi_Mod](#)” on page 265 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_mod_get_operation(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

smod Pointer to an initialized “[Slapi_Mod](#)” on page 265.

Returns

This function returns one of LDAP_MOD_ADD, LDAP_MOD_DELETE, LDAP_MOD_REPLACE, combined using the bitwise or operator with LDAP_MOD_BVALUES.

See Also

“[slapi_mod_set_operation\(\)](#)” on page 429

slapi_mod_get_type()

Gets the attribute type of a “[Slapi_Mod](#)” on page 265 structure.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_mod_get_type(const Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

smod Pointer to an initialized “[Slapi_Mod](#)” on page 265.

Returns

This function returns a read-only pointer to the attribute type in the “[Slapi_Mod](#)” on page 265.

Description

Gets the LDAP attribute type of a “[Slapi_Mod](#)” on page 265.

See Also

“[slapi_mod_set_type\(\)](#)” on page 429

slapi_mod_init()

Initializes a “[Slapi_Mod](#)” on page 265 structure.

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 ” on page 265. |
| <i>initCount</i> | Suggested number of attribute values for which to make room. Minimum value is 0. |

Description

This function initializes a “[Slapi_Mod](#)” on page 265 so that it is empty, but initially has room for the given number of attribute values.

Memory Concerns

If you are unsure of the room you will need, you may use an `initCount` of 0. The “[Slapi_Mod](#)” on page 265 expands as necessary.

See Also

“[slapi_mod_done\(\)](#)” on page 418

“[slapi_mod_init_byref\(\)](#)” on page 425

“[slapi_mod_init_byval\(\)](#)” on page 425

“[slapi_mod_init_passin\(\)](#)” on page 426

slapi_mod_init_byref()

Initializes a “[Slapi_Mod](#)” on page 265 structure that is a wrapper for an existing “[LDAPMod](#)” on page 250.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init_byref(Slapi_Mod *smod, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

smod Pointer to an uninitialized “[Slapi_Mod](#)” on page 265.

mod Pointer to an “[LDAPMod](#)” on page 250.

Description

This function initializes a “[Slapi_Mod](#)” on page 265 containing a reference to an “[LDAPMod](#)” on page 250. Use this function when you have an “[LDAPMod](#)” on page 250 and would like the convenience of the “[Slapi_Mod](#)” on page 265 functions to access it.

See Also

“[slapi_mod_done\(\)](#)” on page 418

“[slapi_mod_init\(\)](#)” on page 424

“[slapi_mod_init_byval\(\)](#)” on page 425

“[slapi_mod_init_passin\(\)](#)” on page 426

slapi_mod_init_byval()

Initializes a “[Slapi_Mod](#)” on page 265 structure with a copy of an “[LDAPMod](#)” on page 250.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init_byval(Slapi_Mod *smod, const LDAPMod *mod);
```

Parameters

This function takes the following parameters:

smod Pointer to an uninitialized “[Slapi_Mod](#)” on page 265.

mod Pointer to an “[LDAPMod](#)” on page 250.

See Also

[“slapi_mod_done\(\)” on page 418](#)

[“slapi_mod_init\(\)” on page 424](#)

[“slapi_mod_init_byref\(\)” on page 425](#)

[“slapi_mod_init_passin\(\)” on page 426](#)

slapi_mod_init_passin()

Initializes a “[Slapi_Mod](#)” on page 265 from an “[LDAPMod](#)” on page 250.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_init_passin(Slapi_Mod *smod, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

smod Pointer to an uninitialized “[Slapi_Mod](#)” on page 265.

mod Pointer to an “[LDAPMod](#)” on page 250.

Description

This function initializes a “[Slapi_Mod](#)” on page 265 by passing in an “[LDAPMod](#)” on page 250. Use this function to convert an “[LDAPMod](#)” on page 250 to a “[Slapi_Mod](#)” on page 265.

Memory Concerns

Responsibility for the “[LDAPMod](#)” on page 250 is transferred to the “[Slapi_Mod](#)” on page 265. The “[LDAPMod](#)” on page 250 is destroyed when the “[Slapi_Mod](#)” on page 265 is destroyed.

See Also

[“slapi_mod_done\(\)” on page 418](#)

[“slapi_mod_init\(\)” on page 424](#)

[“slapi_mod_init_byref\(\)” on page 425](#)

[“slapi_mod_init_byval\(\)” on page 425](#)

slapi_mod_isvalid()

Determines whether a [“Slapi_Mod” on page 265](#) structure is valid.

Syntax

```
#include "slapi-plugin.h"
int slapi_mod_isvalid(const Slapi_Mod *mod);
```

Parameters

This function takes the following parameters:

smod Pointer to a [“Slapi_Mod” on page 265](#).

Returns

This function returns one of the following values:

- 1 if the [“Slapi_Mod” on page 265](#) is valid.
- 0 if the [“Slapi_Mod” on page 265](#) is not valid.

Description

Use this function to verify that the contents of [“Slapi_Mod” on page 265](#) are valid. It is considered valid if the operation type is one of LDAP_MOD_ADD, LDAP_MOD_DELETE, LDAP_MOD_REPLACE, combined using the bitwise or operator with LDAP_MOD_BVALUES; the attribute type is not NULL; and there is at least one attribute value for add and replace operations.

slapi_mod_new()

Allocates a new [“Slapi_Mod” on page 265](#) structure.

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” on page 265](#).

Description

This function allocates a new uninitialized [“Slapi_Mod” on page 265](#). Use this function when you need to a [“Slapi_Mod” on page 265](#) allocated from the heap, rather than from the stack.

See Also

[“slapi_mod_free\(\)” on page 419](#)

slapi_mod_remove_value()

Removes the value at the current [“Slapi_Mod” on page 265](#) iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_remove_value(Slapi_Mod *smod);
```

Parameters

This function takes the following parameter:

smod Pointer to an initialized [“Slapi_Mod” on page 265](#).

See Also

[“slapi_mod_get_first_value\(\)” on page 419](#)

[“slapi_mod_get_next_value\(\)” on page 421](#)

slapi_mod_set_operation()

Sets the operation type of a “[Slapi_Mod](#)” on page 265 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_set_operation(Slapi_Mod *smod, int op);
```

Parameters

This function takes the following parameters:

- smod* Pointer to an initialized “[Slapi_Mod](#)” on page 265.
- op* One of LDAP_MOD_ADD, LDAP_MOD_DELETE, LDAP_MOD_REPLACE, combined using the bitwise or operator with LDAP_MOD_BVALUES.

See Also

“[slapi_mod_get_operation\(\)](#)” on page 423

slapi_mod_set_type()

Sets the attribute type of a “[Slapi_Mod](#)” on page 265.

Syntax

```
#include "slapi-plugin.h"
void slapi_mod_set_type(Slapi_Mod *smod, const char *type);
```

Parameters

This function takes the following parameters:

- smod* Pointer to an initialized “[Slapi_Mod](#)” on page 265.
- type* An attribute type.

Description

Sets the attribute type of the “[Slapi_Mod](#)” on page 265 to a copy of the given value.

See Also

“[slapi_mod_get_type\(\)](#)” on page 423

slapi_moddn_get_newdn()

Builds the new DN of an entry.

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*.

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 *newsuperiordn*, 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).

Memory Concerns

You must free the DN returned using [“slapi_ch_free_string\(\)”](#) on page 330.

Function Reference, Part II

This chapter contains the second part of the reference to the public functions for writing plug-ins. The previous chapter contains the first part of the reference.

Sections in the previous chapter cover plug-in API functions from `slapi_access_allowed()` to `slapi_moddn_get_newdn()`.

The following sections cover plug-in API functions in alphabetical order from `slapi_modify_internal_pb()` to `slapi_wait_condvar()`.

Functions Alphabetically, Part 1

`slapi_modify_internal_pb()`

Performs an LDAP modify operation based on a parameter block to modify a directory entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_modify_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

pb A parameter block that has been initialized using “[slapi_modify_internal_set_pb\(\)](#)” on [page 432](#).

Returns

This function returns -1 if the parameter passed is a NULL pointer. Otherwise, it returns 0.

After your code calls this function, the server sets `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to the appropriate LDAP result code. You can therefore check `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to determine whether an error has occurred.

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\(\)](#)” on [page 432](#).

Memory Concerns

None of the parameters that are passed to “[slapi_modify_internal_set_pb\(\)](#)” on [page 432](#) are altered or consumed by this function.

slapi_modify_internal_set_pb()

Prepares a parameter block for an internal modify operation.

Syntax

```
#include "slapi-plugin.h"
int slapi_modify_internal_set_pb(Slapi_PBlock *pb,
    const char *dn, LDAPMod **mods, LDAPControl **controls,
    const char *uniqueid, Slapi_ComponentId *plugin_identity,
    int operation_flags);
```

Parameters

This function takes the following parameters:

<code>pb</code>	Parameter block for the internal modify operation
<code>dn</code>	Distinguished Name of the entry to modify
<code>mods</code>	Array of modifications to apply
<code>controls</code>	Array of controls to request for the modify operation
<code>uniqueid</code>	Unique identifier for the entry if using this rather than DN
<code>plugin_identity</code>	Plug-in identifier obtained from <code>SLAPI_PLUGIN_IDENTITY</code> during plug-in initialization
<code>operation_flags</code>	<code>NULL</code> or <code>SLAPI_OP_FLAG_NEVER_CHAIN</code>

Returns

This function returns 0 if successful. Otherwise, it returns an LDAP error code.

Description

This function prepares a parameter block for use with “[slapi_modify_internal_pb\(\)](#)” on [page 431](#).

Memory Concerns

Allocate the parameter block using “[slapi_pblock_new\(\)](#)” on [page 464](#) before calling this function.

Directory Server does not free the parameters you passed to this function.

Free the parameter block after calling “[slapi_modify_internal_pb\(\)](#)” on [page 431](#).

See Also

“[slapi_modify_internal_pb\(\)](#)” on [page 431](#)

“[slapi_pblock_new\(\)](#)” on [page 464](#)

slapi_modrdn_internal_pb()

Performs an LDAP modify RDN operation based on a parameter block to rename a directory entry.

Syntax

```
#include "slapi-plugin.h"
int slapi_modrdn_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

pb A parameter block that has been initialized using “[slapi_rename_internal_set_pb\(\)](#)” on [page 493](#).

Returns

This function returns -1 if the parameter passed is a NULL pointer. Otherwise, it returns 0.

After your code calls this function, the server sets `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to the appropriate LDAP result code. You can therefore check `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to determine whether an error has occurred.

Description

This function performs an internal modify RDN operation based on a parameter block. The parameter block should be initialized by calling “[slapi_rename_internal_set_pb\(\)](#)” on [page 493](#).

Memory Concerns

None of the parameters that are passed to “[slapi_rename_internal_set_pb\(\)](#)” on [page 493](#) are altered or consumed by this function.

See Also

“[slapi_rename_internal_set_pb\(\)](#)” on [page 493](#)

slapi_mods2entry()

Creates a “[Slapi_Entry](#)” on [page 263](#) from an array of “[LDAPMod](#)” on [page 250](#).

Syntax

```
#include "slapi-plugin.h"
int slapi_mods2entry(Slapi_Entry **e, const char *dn,
    LDAPMod **attrs);
```

Parameters

This function takes the following parameters:

- e* Address of a pointer that will be set on return to the created entry.
- dn* The LDAP DN of the entry.
- attrs* An array of “[LDAPMod](#)” on [page 250](#) of type `LDAP_MOD_ADD` representing the entry attributes.

Returns

This function returns `LDAP_SUCCESS` if successful, or an LDAP return code if not successful.

Description

This function creates a “[Slapi_Entry](#)” on page 263 from a copy of an array of “[LDAPMod](#)” on page 250 of type `LDAP_MODD_ADD`.

See Also

“[slapi_entry2mods\(\)](#)” on page 347

slapi_mods_add()

Appends a new mod with a single attribute value to “[Slapi_Mods](#)” on page 266 structure.

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>smods</i>	Pointer to an initialized “ Slapi_Mods ” on page 266.
<i>modtype</i>	One of <code>LDAP_MOD_ADD</code> , <code>LDAP_MOD_DELETE</code> , <code>LDAP_MOD_REPLACE</code> .
<i>type</i>	The LDAP attribute type.
<i>len</i>	The length in bytes of the attribute value.
<i>val</i>	The attribute value.

Description

This function appends a new mod with a single attribute value to a “[Slapi_Mods](#)” on page 266. The mod is constructed from copies of the values of `modtype`, `type`, `len`, and `val`.

Memory Concerns

This function must not be used on “[Slapi_Mods](#)” on page 266 initialized with “[slapi_mods_init_byref\(\)](#)” on page 448.

See Also

[“slapi_mods_add_ldapmod\(\)” on page 436](#)

[“slapi_mods_add_mod_values\(\)” on page 437](#)

[“slapi_mods_add_modbvps\(\)” on page 438](#)

[“slapi_mods_add_string\(\)” on page 439](#)

slapi_mods_add_ldapmod()

Appends an “LDAPMod” on page 250 to a “Slapi_Mods” on page 266 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_ldapmod(Slapi_Mods *smods, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

smods Pointer to an initialized “Slapi_Mods” on page 266.

mod Pointer to a the “LDAPMod” on page 250 to be appended.

Description

Appends an “LDAPMod” on page 250 to a “Slapi_Mods” on page 266.

Memory Concerns

Responsibility for the “LDAPMod” on page 250 is transferred to the “Slapi_Mods” on page 266.

This function must not be used on a “Slapi_Mods” on page 266 initialized with “slapi_mods_init_byref()” on page 448.

See Also

[“slapi_mods_add_ldapmod\(\)” on page 436](#)

[“slapi_mods_add_mod_values\(\)” on page 437](#)

[“slapi_mods_add_modbvps\(\)” on page 438](#)

[“slapi_mods_add_string\(\)” on page 439](#)

slapi_mods_add_mod_values()

Appends a new mod to a “[Slapi_Mods](#)” on page 266 structure, with attribute values provided as an array of “[Slapi_Value](#)” on page 270.

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:

- smods* Pointer to an initialized “[Slapi_Mods](#)” on page 266.
- modtype* One of LDAP_MOD_ADD, LDAP_MOD_DELETE, LDAP_MOD_REPLACE.
- type* The LDAP attribute type.
- va* A NULL terminated array of “[Slapi_Value](#)” on page 270 representing the attribute values.

Description

This function appends a new mod to a “[Slapi_Mods](#)” on page 266. 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](#)” on page 270.

Memory Concerns

This function must not be used on a “[Slapi_Mods](#)” on page 266 initialized with “[slapi_mods_init_byref\(\)](#)” on page 448.

See Also

“[slapi_mods_add\(\)](#)” on page 435

“[slapi_mods_add_ldapmod\(\)](#)” on page 436

“[slapi_mods_add_modbvps\(\)](#)” on page 438

“[slapi_mods_add_string\(\)](#)” on page 439

slapi_mods_add_modbvps()

Appends a new mod to a “[Slapi_Mods](#)” on page 266 structure, with attribute values provided as an array of “[berval](#)” on page 249.

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>smods</i> | Pointer to an initialized “ Slapi_Mods ” on page 266. |
| <i>modtype</i> | One of LDAP_MOD_ADD, LDAP_MOD_DELETE, LDAP_MOD_REPLACE. |
| <i>type</i> | The LDAP attribute type. |
| <i>bvps</i> | A NULL terminated array of “ berval ” on page 249 representing the attribute values. |

Description

This function appends a new mod to “[Slapi_Mods](#)” on page 266. 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](#)” on page 249

Memory Concerns

This function must not be used on a “[Slapi_Mods](#)” on page 266 initialized with “[slapi_mods_init_byref\(\)](#)” on page 448.

See Also

- “[slapi_mods_add\(\)](#)” on page 435
- “[slapi_mods_add_ldapmod\(\)](#)” on page 436
- “[slapi_mods_add_mod_values\(\)](#)” on page 437
- “[slapi_mods_add_string\(\)](#)” on page 439

slapi_mods_add_smod()

Appends a “[Slapi_Mod](#)” on page 265 to a “[Slapi_Mods](#)” on page 266 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_add_smod(Slapi_Mods *smods, Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

- smods* Pointer to an initialized “[Slapi_Mods](#)” on page 266.
- smod* Pointer to a the “[Slapi_Mod](#)” on page 265 to be appended.

Description

Appends a “[Slapi_Mod](#)” on page 265 to a “[Slapi_Mods](#)” on page 266.

Memory Concerns

Responsibility for the “[Slapi_Mod](#)” on page 265 is transferred to the “[Slapi_Mods](#)” on page 266.

This function must not be used on a “[Slapi_Mods](#)” on page 266 initialized with “[slapi_mods_init_byref\(\)](#)” on page 448.

See Also

- “[slapi_mods_add\(\)](#)” on page 435
- “[slapi_mods_add_mod_values\(\)](#)” on page 437
- “[slapi_mods_add_modbvps\(\)](#)” on page 438
- “[slapi_mods_add_string\(\)](#)” on page 439

slapi_mods_add_string()

Appends a new mod to “[Slapi_Mods](#)” on page 266 structure with a single attribute value provided as a string.

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>smods</i>	Pointer to an initialized “Slapi_Mods” on page 266 .
<i>modtype</i>	One of LDAP_MOD_ADD, LDAP_MOD_DELETE, LDAP_MOD_REPLACE.
<i>type</i>	The LDAP attribute type.
<i>val</i>	The attribute value represented as a NULL terminated string.

Description

This function appends a new mod with a single string attribute value to a [“Slapi_Mods” on page 266](#). The mod is constructed from copies of the values of *modtype*, *type* and *val*.

Memory Concerns

This function must not be used on a [“Slapi_Mods” on page 266](#) initialized with [“slapi_mods_init_byref\(\)” on page 448](#).

See Also

[“slapi_mods_add\(\)” on page 435](#)

[“slapi_mods_add_ldapmod\(\)” on page 436](#)

[“slapi_mods_add_mod_values\(\)” on page 437](#)

[“slapi_mods_add_modbvps\(\)” on page 438](#)

slapi_mods_done()

Frees internals of a [“Slapi_Mods” on page 266](#) structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_done(Slapi_Mods *smods);
```


Parameters

This function takes the following parameter:

smods Pointer to a “[Slapi_Mods](#)” on page 266 structure.

Description

This function frees the internals of a “[Slapi_Mods](#)” on page 266, leaving it in the uninitialized state. Use this function on a stack-allocated “[Slapi_Mods](#)” on page 266 when you are finished with it, or when you wish to reuse it.

See Also

“[slapi_mods_init\(\)](#)” on page 447

“[slapi_mods_init_byref\(\)](#)” on page 448

“[slapi_mods_init_passin\(\)](#)” on page 448

`slapi_mods_dump()`

Dumps the contents of a “[Slapi_Mods](#)” on page 266 structure to the server log.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_dump(const Slapi_Mods *smods, const char *text);
```

Parameters

This function takes the following parameters:

smods Pointer to a “[Slapi_Mods](#)” on page 266

text Descriptive text that will be included in the log, preceding the “[Slapi_Mods](#)” on page 266 content.

Description

This function uses the LDAP_DEBUG_ANY log level to dump the contents of a “[Slapi_Mods](#)” on page 266 to \$INSTANCE_PATH/logs/errors for debugging.

slapi_mods_free()

Frees a “[Slapi_Mods](#)” on page 266 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_free(Slapi_Mods **smods);
```

Parameters

This function takes the following parameter:

smods Pointer to an allocated “[Slapi_Mods](#)” on page 266.

Description

This function frees a “[Slapi_Mods](#)” on page 266 that was allocated by “[slapi_mods_new\(\)](#)” on page 454.

See Also

“[slapi_mods_new\(\)](#)” on page 454

slapi_mods_get_first_mod()

Initializes a “[Slapi_Mods](#)” on page 266 iterator and returns the first “[LDAPMod](#)” on page 250.

Syntax

```
#include "slapi-plugin.h"
LDAPMod *slapi_mods_get_first_mod(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266.

Returns

This function returns a pointer to the first “[LDAPMod](#)” on page 250 in the “[Slapi_Mods](#)” on page 266, or NULL if there are no mods.

slapi_mods_get_first_smod()

Initializes a “[Slapi_Mods](#)” on page 266 iterator and returns the first mod wrapped in a “[Slapi_Mods](#)” on page 266 structure.

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:

- smods* A pointer to an initialized “[Slapi_Mods](#)” on page 266.
- smod* Pointer to a “[Slapi_Mod](#)” on page 265 that used to hold the mod.

Returns

This function returns a pointer to the “[Slapi_Mod](#)” on page 265, wrapping the first mod, or NULL if no mod exists.

Description

Use this function in conjunction with “[slapi_mods_get_next_smod\(\)](#)” on page 446 to iterate through the mods in a “[Slapi_Mods](#)” on page 266 using a “[Slapi_Mods](#)” on page 266 wrapper.

Memory Concerns

Only one thread may be iterating through a particular “[Slapi_Mods](#)” on page 266 at any given time.

See Also

“[slapi_mods_get_next_smod\(\)](#)” on page 446

slapi_mods_get_ldapmods_byref()

Gets a reference to the array of “[LDAPMod](#)” on page 250 in a “[Slapi_Mods](#)” on page 266 structure.

Syntax

```
#include "slapi-plugin.h"
LDAPMod **slapi_mods_get_ldapmods_byref(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266.

Returns

This function returns a NULL terminated array of “[LDAPMod](#)” on page 250 owned by the “[Slapi_Mods](#)” on page 266.

Description

Use this function to get direct access to the array of “[LDAPMod](#)” on page 250 contained in a “[Slapi_Mods](#)” on page 266.

Memory Concerns

Responsibility for the array remains with the “[Slapi_Mods](#)” on page 266 .

See Also

“[slapi_mods_get_ldapmods_passout\(\)](#)” on page 444

slapi_mods_get_ldapmods_passout()

Retrieves the array of “[LDAPMod](#)” on page 250 contained in a “[Slapi_Mods](#)” on page 266 structure.

Syntax

```
#include "slapi-plugin.h"
LDAPMod **slapi_mods_get_ldapmods_passout(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266.

Returns

A NULL terminated array “[LDAPMod](#)” on page 250 owned by the caller.

Description

Gets the array of “[LDAPMod](#)” on page 250 out of a “[Slapi_Mods](#)” on page 266. Responsibility for the array transfers to the caller. The “[Slapi_Mods](#)” on page 266 is left in the uninitialized state.

See Also

“[slapi_mods_get_ldapmods_byref\(\)](#)” on page 443

slapi_mods_get_next_mod()

Increments the “[Slapi_Mods](#)” on page 266 iterator and returns the next “[LDAPMod](#)” on page 250.

Syntax

```
#include "slapi-plugin.h"
LDAPMod *slapi_mods_get_next_mod(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266.

Returns

This function returns a pointer to the next “[LDAPMod](#)” on page 250 , or NULL if there are no more.

Description

Use this function in conjunction with “[slapi_mods_get_first_mod\(\)](#)” on page 442 to iterate through the mods in a “[Slapi_Mods](#)” on page 266. This will return an “[LDAPMod](#)” on page 250 each time until the end.

Memory Concerns

Only one thread may be iterating through a particular “[Slapi_Mods](#)” on page 266 at any given time.

See Also

[“slapi_mods_get_first_mod\(\)” on page 442](#)

slapi_mods_get_next_smod()

Increments the [“Slapi_Mods” on page 266](#) iterator and returns the next mod wrapped in a [“Slapi_Mod” on page 266](#).

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:

- smods* A pointer to a an initialized [“Slapi_Mods” on page 266](#).
- smod* Pointer to a [“Slapi_Mod” on page 265](#) that used to hold the mod.

Returns

This function returns a pointer to the [“Slapi_Mod” on page 265](#), wrapping the next mod, or NULL if there are no more mods.

Description

Use this function in conjunction with [“slapi_mods_get_first_smod\(\)” on page 443](#) to iterate through the mods in a [“Slapi_Mods” on page 266](#) using a [“Slapi_Mods” on page 266](#) wrapper.

Memory Concerns

Only one thread may be iterating through a particular [“Slapi_Mods” on page 266](#) at any given time.

See Also

[“slapi_mods_get_first_smod\(\)” on page 443](#)

slapi_mods_get_num_mods()

Gets the number of mods in a “[Slapi_Mods](#)” on page 266 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_mods_get_num_mods(const Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266.

Returns

The number of mods in the “[Slapi_Mods](#)” on page 266.

slapi_mods_init()

Initializes a “[Slapi_Mods](#)” on page 266.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_init(Slapi_Mods *smods, int initCount);
```

Parameters

This function takes the following parameters:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266.

initCount Number of modifications to allocate initially.

Description

Initializes a “[Slapi_Mods](#)” on page 266 so that it is empty, but initially has room for the given number of mods.

Memory Concerns

If you are unsure of how much room you will need, you may use an `initCount` of 0. The “[Slapi_Mods](#)” on page 266 expands as necessary.

See Also

[“slapi_mods_done\(\)” on page 440](#)

slapi_mods_init_byref()

Initializes a [“Slapi_Mods” on page 266](#) that is a wrapper for an existing array of [“LDAPMod” on page 250](#).

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_init_byref(Slapi_Mods *smods, LDAPMod **mods);
```

Parameters

This function takes the following parameters:

smods Pointer to an uninitialized [“Slapi_Mods” on page 266](#).

mods A NULL terminated array of [“LDAPMod” on page 250](#).

Description

Initializes a [“Slapi_Mods” on page 266](#) containing a reference to an array of [“LDAPMod” on page 250](#). This function provides the convenience of using [“Slapi_Mods” on page 266](#) functions to access [“LDAPMod” on page 250](#) array items.

Memory Concerns

The array is not destroyed when the [“Slapi_Mods” on page 266](#) is destroyed. Notice that you cannot insert new mods in a [“Slapi_Mods” on page 266](#) that has been initialized by reference.

See Also

[“slapi_mods_done\(\)” on page 440](#)

slapi_mods_init_passin()

Initializes a [“Slapi_Mods” on page 266](#) structure from an array of [“LDAPMod” on page 250](#).

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_init_passin(Slapi_Mods *smods, LDAPMod **mods);
```

Parameters

This function takes the following parameters:

smods Pointer to an uninitialized “[Slapi_Mods](#)” on page 266.

mods A NULL terminated array of “[LDAPMod](#)” on page 250.

Description

This function initializes a “[Slapi_Mods](#)” on page 266 by passing in an array of “[LDAPMod](#)” on page 250. This function converts an array of “[LDAPMod](#)” on page 250 to a “[Slapi_Mods](#)” on page 266.

Memory Concerns

The responsibility for the array and its elements is transferred to the “[Slapi_Mods](#)” on page 266. The array and its elements are destroyed when the “[Slapi_Mods](#)” on page 266 is destroyed.

See Also

“[slapi_mods_done\(\)](#)” on page 440

slapi_mods_insert_after()

Inserts an “[LDAPMod](#)” on page 250 into a “[Slapi_Mods](#)” on page 266 structure after the current iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_after(Slapi_Mods *smods, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266 with a valid iterator position.

mod Pointer to the “[LDAPMod](#)” on page 250 to be inserted.

Description

This function inserts an “[LDAPMod](#)” on page 250 in a “[Slapi_Mods](#)” on page 266 immediately after the current position of the “[Slapi_Mods](#)” on page 266 iterator. The iterator position is unchanged.

Memory Concerns

Responsibility for the “[LDAPMod](#)” on page 250 is transferred to the “[Slapi_Mods](#)” on page 266.

This function must not be used on a “[Slapi_Mods](#)” on page 266 initialized with “[slapi_mods_init_byref\(\)](#)” on page 448.

See Also

“[slapi_mods_get_first_mod\(\)](#)” on page 442

“[slapi_mods_get_first_smod\(\)](#)” on page 443

“[slapi_mods_get_next_mod\(\)](#)” on page 445

“[slapi_mods_get_next_smod\(\)](#)” on page 446

slapi_mods_insert_at()

Inserts an “[LDAPMod](#)” on page 250 anywhere in a “[Slapi_Mods](#)” on page 266.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_at(Slapi_Mods *smods, LDAPMod *mod, int pos);
```

Parameters

This function takes the following parameters:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266.

mod Pointer to the “[LDAPMod](#)” on page 250 to be inserted.

pos Position at which to insert the new mod. Minimum value is 0. Maximum value is the current number of mods.

Description

This function inserts an “LDAPMod” on page 250 at a given position in “Slapi_Mods” on page 266. Position 0 (zero) refers to the first mod. A position equal to the current number of 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.

Memory Concerns

Responsibility for the “LDAPMod” on page 250 is transferred to the “Slapi_Mods” on page 266.

This function must not be used on a “Slapi_Mods” on page 266 initialized with “slapi_mods_init_byref()” on page 448.

slapi_mods_insert_before()

Inserts an “LDAPMod” on page 250 into a “Slapi_Mods” on page 266 structure before the current iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_before(Slapi_Mods *smods, LDAPMod *mod);
```

Parameters

This function takes the following parameters:

smods Pointer to an initialized “Slapi_Mods” on page 266 with a valid iterator position.
mod Pointer to the “LDAPMod” on page 250 to be inserted.

Description

Inserts an “LDAPMod” on page 250 into a “Slapi_Mods” on page 266 immediately before the current position of the “Slapi_Mods” on page 266 iterator. The iterator position is unchanged.

Memory Concerns

The responsibility for the “LDAPMod” on page 250 is transferred to the “Slapi_Mods” on page 266.

This function must not be used on a “Slapi_Mods” on page 266 initialized with “slapi_mods_init_byref()” on page 448.

See Also

[“slapi_mods_get_first_mod\(\)” on page 442](#)

[“slapi_mods_get_next_mod\(\)” on page 445](#)

slapi_mods_insert_smod_at()

Inserts a [“Slapi_Mod” on page 265](#) anywhere in a [“Slapi_Mods” on page 266](#).

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:

- smods* Pointer to an initialized [“Slapi_Mods” on page 266](#).
- smod* Pointer to the [“Slapi_Mod” on page 265](#) to be inserted.
- pos* Position at which to insert the [“Slapi_Mod” on page 265](#). Minimum value is 0. Maximum value is the current number of mods.

Description

This function inserts a [“Slapi_Mod” on page 265](#) at a given position in [“Slapi_Mods” on page 266](#). Position 0 (zero) refers to the first mod. A position equal to the current number of 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.

Memory Concerns

Responsibility for the [“Slapi_Mod” on page 265](#) is transferred to the [“Slapi_Mods” on page 266](#).

This function must not be used on a [“Slapi_Mods” on page 266](#) initialized with [“slapi_mods_init_byref\(\)” on page 448](#).

slapi_mods_insert_smod_before()

Inserts a [“Slapi_Mod” on page 265](#) into a [“Slapi_Mods” on page 266](#) structure before the current iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_insert_smod_before(Slapi_Mods *smods,
    Slapi_Mod *smod);
```

Parameters

This function takes the following parameters:

- smods* Pointer to an initialized “[Slapi_Mods](#)” on page 266 with a valid iterator position.
- smod* Pointer to the “[Slapi_Mod](#)” on page 265 to be inserted.

Description

Inserts a “[Slapi_Mod](#)” on page 265 into a “[Slapi_Mods](#)” on page 266 immediately before the current position of the “[Slapi_Mods](#)” on page 266 iterator. The iterator position is unchanged.

Memory Concerns

The responsibility for the “[Slapi_Mod](#)” on page 265 is transferred to the “[Slapi_Mods](#)” on page 266.

This function must not be used on a “[Slapi_Mods](#)” on page 266 initialized with “[slapi_mods_init_byref\(\)](#)” on page 448.

See Also

“[slapi_mods_get_first_mod\(\)](#)” on page 442

“[slapi_mods_get_next_mod\(\)](#)” on page 445

slapi_mods_iterator_backone()

Decrements the “[Slapi_Mods](#)” on page 266 current iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_iterator_backone(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266.

Description

This function moves the iterator back one position.

See Also

“[slapi_mods_get_first_mod\(\)](#)” on page 442

“[slapi_mods_get_first_smod\(\)](#)” on page 443

“[slapi_mods_get_next_mod\(\)](#)” on page 445

“[slapi_mods_get_next_smod\(\)](#)” on page 446

slapi_mods_new()

Allocates a new uninitialized “[Slapi_Mods](#)” on page 266 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mods* slapi_mods_new( void );
```

Parameters

This function takes no parameters.

Returns

A pointer to an allocated uninitialized “[Slapi_Mods](#)” on page 266.

Description

This function allocates a new initialized “[Slapi_Mods](#)” on page 266 .

Memory Concerns

Use this function when you need a “[Slapi_Mods](#)” on page 266 allocated from the heap, rather than from the stack.

See Also

“[slapi_mods_free\(\)](#)” on page 442

slapi_mods_remove()

Removes the mod at the current “[Slapi_Mods](#)” on page 266 iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_remove(Slapi_Mods *smods);
```

Parameters

This function takes the following parameter:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266.

Description

This function removes the mod at the current iterator position.

See Also

“[slapi_mods_get_first_mod\(\)](#)” on page 442

“[slapi_mods_get_first_smod\(\)](#)” on page 443

“[slapi_mods_get_next_mod\(\)](#)” on page 445

“[slapi_mods_get_next_smod\(\)](#)” on page 446

slapi_mods_remove_at()

Removes the mod at the specified “[Slapi_Mods](#)” on page 266 iterator position.

Syntax

```
#include "slapi-plugin.h"
void slapi_mods_remove_at(Slapi_Mods *smods, int pos);
```

Parameters

This function takes the following parameters:

smods Pointer to an initialized “[Slapi_Mods](#)” on page 266.

pos Position of the mod to remove.

Description

This function removes the `mod` at the *pos* iterator position, obtained by counting as you iterate through the modifications in a “[Slapi_Mods](#)” on page 266.

See Also

“[slapi_mods_get_first_mod\(\)](#)” on page 442

“[slapi_mods_get_first_smod\(\)](#)” on page 443

“[slapi_mods_get_next_mod\(\)](#)” on page 445

“[slapi_mods_get_next_smod\(\)](#)” on page 446

`slapi_mr_filter_index()`

Call a matching rule filter index function.

Syntax

```
#include "slapi-plugin.h"
int slapi_mr_filter_index(Slapi_Filter* f, Slapi_PBlock* pb);
```

Parameters

This function takes the following parameters:

- `f` Filter containing the matching rule OID
- `pb` Parameter block to pass to the filter index function

Description

This function enables plug-ins to call matching rule filter index functions.

Returns

This function returns 0 if successful. It returns `LDAP_UNAVAILABLE_CRITICAL_EXTENSION` if no filter index function is available for the rule in the filter. It returns -1 if something goes horribly wrong setting the parameter block arguments.

`slapi_mr_indexer_create()`

Set a pointer in the parameter block to the indexer factory function for a matching rule.

Syntax

```
#include "slapi-plugin.h"
int slapi_mr_indexer_create(Slapi_PBlock * pb);
```

Parameters

This function takes the following parameters:

pb Parameter block containing the matching rule object identifier affected to
 SLAPI_PLUGIN_MR_OID

Description

This function enables plug-ins to call matching rule indexer factory functions. If successful, the function sets SLAPI_PLUGIN_MR_INDEXER_CREATE_FN in pb to point to the indexer factory function.

Returns

This function returns 0 if successful. It returns LDAP_UNAVAILABLE_CRITICAL_EXTENSION if no indexer factory function is available for the matching rule. It returns -1 if something goes horribly wrong getting or setting the parameter block arguments.

slapi_new_condvar()

Allocate a condition variable.

Syntax

```
#include "slapi-plugin.h"
Slapi_CondVar *slapi_new_condvar( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameter:

mutex Mutex used for the lock

Description

This function enables thread synchronization using a wait/notify mechanism.

Returns

This function returns a pointer to the new condition variable if successful. Otherwise, it returns `NULL`.

Memory Considerations

Call this function to create the condition variable and [“`slapi_destroy_condvar\(\)`” on page 338](#) to free the condition variable.

See Also

[“`slapi_destroy_condvar\(\)`” on page 338](#)

[“`slapi_notify_condvar\(\)`” on page 459](#)

[“`slapi_wait_condvar\(\)`” on page 575](#)

`slapi_new_mutex()`

Allocate a mutex.

Syntax

```
#include "slapi-plugin.h"
Slapi_Mutex *slapi_new_mutex( void );
```

Returns

This function returns a pointer to the new mutex if successful. Otherwise, it returns `NULL`.

Description

This function enables thread synchronization. Once a thread has locked the mutex using [“`slapi_lock_mutex\(\)`” on page 407](#), other threads attempting to acquire the lock are blocked until the thread holding the mutex calls [“`slapi_UTF-8STRTOLOWER\(\)`” on page 534](#).

Memory Considerations

Call [“`slapi_destroy_mutex\(\)`” on page 338](#) to free the mutex.

See Also

[“slapi_destroy_mutex\(\)” on page 338](#)

[“slapi_lock_mutex\(\)” on page 407](#)

slapi_notify_condvar()

Notify a change in a condition variable.

Syntax

```
#include "slapi-plugin.h"
int slapi_notify_condvar( Slapi_CondVar *cvar, int notify_all );
```

Parameters

This function takes the following parameter:

<i>cvar</i>	Condition variable changed
<i>notify_all</i>	NULL means notify only the next thread waiting for notification. Otherwise, all threads are notified.

Description

This function enables thread synchronization using a wait/notify mechanism.

Returns

This function returns 1 if successful. Otherwise, it returns NULL.

Memory Considerations

Call [“slapi_wait_condvar\(\)” on page 575](#) to wait on a change to the condition variable.

See Also

[“slapi_destroy_condvar\(\)” on page 338](#)

[“slapi_new_condvar\(\)” on page 457](#)

[“slapi_wait_condvar\(\)” on page 575](#)

slapi_op_abandoned()

Determines whether or not the client has abandoned the current operation (the operation that passes in the parameter block).

Syntax

```
#include "slapi-plugin.h"
int slapi_op_abandoned( Slapi_PBlock *pb );
```

Parameters

This function takes the following parameter:

pb Parameter block passed in from the current operation.

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 periodically check the operations status from long-running plug-ins.

Returns

This function returns one of the following values:

- 1 if the operation has been abandoned.
- 0 if the operation has not been abandoned.

slapi_op_get_type()

Gets the type of a “[Slapi_Operation](#)” on [page 267](#).

Syntax

```
#include "slapi-plugin.h"
unsigned long slapi_op_get_type(Slapi_Operation * op);
```

Parameters

This function takes the following parameter:

op The operation of which you wish to get the type.

Description

This function returns the type of an operation. The “[Slapi_Operation](#)” on page 267 structure can be extracted from a pblock structure using “[slapi_pblock_get\(\)](#)” on page 462 with the “[Slapi_Operation](#)” on page 267 parameter. For example:

```
slapi_pblock_get (pb, SLAPI_OPERATION, &op);
```

Returns

This function will return 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\(\)](#)” on page 462

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 parameters:

pb Parameter block that you wish to free.

Memory Concerns

The parameter block that you wish to free must have been created using [“`slapi_pblock_new\(\)`” on page 464](#). Use of this function with parameter blocks allocated on the stack, with `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 parameter block 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\(\)`” on page 464](#)

`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 that you want to get. For a list of IDs that you can specify, see Chapter 15 |

value 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/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\(\)`” on page 461](#). The exception is if the value is explicitly set by the caller through [“`slapi_pblock_set\(\)`” on page 464](#). 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\(\)`” on page 464](#) with a value of NULL. For example:

```
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
                                         references this. */
```

```
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 previous example, which sets `someparam` to `NULL` after setting it in the parameter block.

See Also

[“`slapi_pblock_destroy\(\)`” on page 461](#)

[“`slapi_pblock_set\(\)`” on page 464](#)

`slapi_pblock_new()`

Creates a new parameter block.

Syntax

```
#include "slapi-plugin.h"  
Slapi_PBlock *slapi_pblock_new();
```

Returns

Pointer to the new parameter block.

Memory Concerns

The parameter block pointer allocated with `slapi_pblock_new()` must always be freed by [“`slapi_pblock_destroy\(\)`” on page 461](#). The use of other memory liberators such as `free()` is not supported and may lead to crashes or memory leaks.

`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:

- pb* Parameter block.
- arg* ID of the name-value pair that you want to set. For a list of IDs that you can specify, see [Chapter 15](#)
- value* Pointer to the value that you want to set in the parameter block.

Returns

This function returns 0 if successful, or -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 such as `char * strings`, `char **arrays`, [“Slapi_Backend” on page 259](#)*, and so forth, 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\(\)” on page 462](#) to retrieve the value to free rather than relying on a potentially dangling pointer. See the [“slapi_pblock_get\(\)” on page 462](#) 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\(\)” on page 462](#)

slapi_pw_find_sv()

Determines whether a specified password matches one of the hashed values of an attribute. For example, you can call this function to determine if a given password matches a value in 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:

- vals* Pointer to the array of [“Slapi_Value” on page 270](#) structure pointers to hold the values of an attribute that stores passwords such as `userpassword`.
- v* Pointer to the [“Slapi_Value” on page 270](#) 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” on page 270](#) using [“slapi_value_init_berval\(\)” on page 545](#).

Returns

This function returns 0 if the password specified by *v* was found in *vals*, or a non-zero value if the password *v* was not found in *vals*.

Memory Concerns

You must allocate and release an array of [“Slapi_Value” on page 270](#) structures for *vals* sized to hold the exact number of password values for the `userpassword` attribute on the entry to check. The resulting array is not NULL terminated. For a simpler memory allocation model, use [“slapi_pw_find_valueset\(\)” on page 467](#) instead.

Description

When the server stores the password for an entry in the `userpassword` attribute, it hashes the password using different schemes.

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 hashed values of the `userpassword` attribute.

See Also

[“`slapi_pw_find_valueset\(\)`” on page 467](#)

`slapi_pw_find_valueset()`

Determines whether or not a specified password matches one of the hashed values of an attribute. For example, you can call this function to determine if a given password matches a value in the `userpassword` attribute.

Syntax

```
#include "slapi-plugin.h"
int slapi_pw_find_valueset(Slapi_ValueSet *valset,
    const Slapi_Value *v);
```

Parameters

This function takes the following parameters:

- valset* Pointer to the “[Slapi_ValueSet](#)” on page 270 structure containing the values of an attribute that stores passwords such as `userpassword`.
- v* Pointer to the “[Slapi_Value](#)” on page 270 structure containing the password to check.

For example, you can get this value from the `SLAPI_BIND_CREDENTIALS` parameter in the parameter block and create the “[Slapi_Value](#)” on page 270 using “[slapi_value_init_berval\(\)](#)” on page 545.

Returns

This function returns 0 if the password specified by *v* was found in *valset*, or a non-zero value if the password *v* was not found in *valset*.

Description

When the server stores the password for an entry in the `userpassword` attribute, it hashes the password using different schemes.

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 hashed values of the `userpassword` attribute.

See Also

[“`slapi_pw_find_sv\(\)`” on page 466](#)

`slapi_rdn_add()`

Adds a new RDN to an existing [“`Slapi_RDN`” on page 269](#) structure.

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:

- rdn* The target [“`Slapi_RDN`” on page 269](#) structure.
- type* The type (cn, o, ou, etc.) of the RDN to be added. This parameter cannot be NULL.
- value* The value of the RDN to be added. This parameter cannot be NULL.

Returns

This function always returns 1.

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`” on page 269](#) structure is no longer current with the new RDN.

See Also

[“slapi_rdn_get_num_components\(\)” on page 476](#)

slapi_rdn_compare()

Compares two RDNs.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_compare(Slapi_RDN *rdn1, Slapi_RDN *rdn2);
```

Parameters

This function takes the following parameters:

rdn1 The first RDN to compare.

rdn2 The second RDN to compare.

Returns

This function returns 0 if *rdn1* and *rdn2* have the same RDN components, or -1 if they do not.

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.

slapi_rdn_contains()

Checks whether a [“Slapi_RDN” on page 269](#) structure holds any RDN matching a given type/value pair.

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 ” on page 269 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 <i>value</i> that should be taken into account for the string operation when searching for the RDN.

Returns

This function returns 1 if *rdn* contains an RDN that matches the *type*, *value* and *length*, or 0 if no RDN matches the desired *type/value*.

Description

This function searches for an RDN inside of the “[Slapi_RDN](#)” on page 269 structure *rdn* that matches both *type* and *value* as given in the parameters. This function makes a call to “[slapi_rdn_get_index\(\)](#)” on page 473 and verifies that the returned value is anything but -1.

See Also

“[slapi_rdn_contains_attr\(\)](#)” on page 470

“[slapi_rdn_get_index\(\)](#)” on page 473

slapi_rdn_contains_attr()

Checks whether a “[Slapi_RDN](#)” on page 269 structure contains any RDN matching a given *type*, and if true, gets 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 ” on page 269 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 *rdn*.

Returns

This function returns 1 if *rdn* contains a RDN that matches the given type, or 0 if there is no match.

Description

This function looks for an RDN inside the “[Slapi_RDN](#)” on page 269 structure *rdn* that matches the type given in the parameters. This function makes a call to “[slapi_rdn_get_index_attr\(\)](#)” on page 474 and verifies that the returned value is anything but -1. If successful, it also returns the corresponding attribute value.

See Also

“[slapi_rdn_contains\(\)](#)” on page 469

“[slapi_rdn_get_index_attr\(\)](#)” on page 474

`slapi_rdn_done()`

Clears an instance of a “[Slapi_RDN](#)” on page 269 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_done(Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

rdn Pointer to the structure to be cleared.

Description

This function clears the contents of a “[Slapi_RDN](#)” on page 269 structure. It frees both the RDN value and the array of split RDNs. Those pointers are then set to NULL.

See Also

“[slapi_rdn_free\(\)](#)” on page 472

“[slapi_rdn_init\(\)](#)” on page 477

slapi_rdn_free()

Frees a “[Slapi_RDN](#)” on page 269 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_free(Slapi_RDN **rdn);
```

Parameters

This function takes the following parameter:

rdn Pointer to the pointer of the “[Slapi_RDN](#)” on page 269 structure to be freed.

Description

This function frees both the contents of the “[Slapi_RDN](#)” on page 269 structure and the structure itself pointed to by the content of *rdn*.

See Also

“[slapi_rdn_done\(\)](#)” on page 471

“[slapi_rdn_new\(\)](#)” on page 480

slapi_rdn_get_first()

Gets the type/value pair corresponding to the first RDN stored in a “[Slapi_RDN](#)” on page 269 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_first(Slapi_RDN *rdn, char **type, char **value);
```

Parameters

This function takes the following parameters:

rdn The “[Slapi_RDN](#)” on page 269 structure containing the RDN value(s).

type Repository that will hold the type of the first RDN. If this parameter is NULL at the return of the function, it means *rdn* is empty.

value Repository that will hold the type of the first RDN. If this parameter is NULL at the return of the function, it means *rdn* is empty.

Returns

This function returns -1 if *rdn* is empty, or 1 if the operation is successful.

Description

This function gets the type/value pair corresponding to the first RDN stored in *rdn*. For example, if the RDN is *cn=Joey*, the function will place *cn* in the type return parameter, and *Joey* in *value*.

See Also

[“slapi_rdn_get_next\(\)” on page 475](#)

[“slapi_rdn_get_rdn\(\)” on page 476](#)

slapi_rdn_get_index()

Gets the index of the RDN that follows the RDN with a given type and value.

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” on page 269 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 <i>value</i> that should be taken into account for the string comparisons when searching for the RDN. A matching RDN value must not exceed the length specified.

Returns

This function returns the index of the RDN that follows the RDN matching the contents of the parameters type and value. If no RDN stored in *rdn* matches the given type/value pair, -1 is returned.

Description

This function searches for an RDN inside the “[Slapi_RDN](#)” on page 269 structure *rdn* that matches both type and value as given in the parameters. If it succeeds, the position of the matching RDN is returned.

See Also

“[slapi_rdn_contains\(\)](#)” on page 469

“[slapi_rdn_get_index_attr\(\)](#)” on page 474

`slapi_rdn_get_index_attr()`

Gets the position and the attribute value of the first RDN in a “[Slapi_RDN](#)” on page 269 structure that matches a given type.

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 ” on page 269 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 exists within <i>rdn</i> . |

Returns

This function returns -1 if there is no RDN that matches the content type, or the real position of the first RDN within RDN that matches the content of type.

Description

This function searches for an RDN inside of the “[Slapi_RDN](#)” on page 269 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.

See Also

“[slapi_rdn_get_index\(\)](#)” on page 473

`slapi_rdn_get_next()`

Gets a certain RDN type/value pair from within the RDNs stored in a “[Slapi_RDN](#)” on page 269 structure.

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 ” on page 269 structure containing the RDN value(s).
<i>index</i>	Indicates the position of the RDN that 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 -1 if there is no RDN in the index position, or the real position of the retrieved RDN if the operation was successful.

Description

This function gets the type/value pair corresponding to the RDN stored in the next (index+1) position inside `rdn`. Notice that the index of an element within an array of values is always one unit below its real position in the array.

See Also

[“slapi_rdn_get_first\(\)” on page 472](#)

[“slapi_rdn_get_rdn\(\)” on page 476](#)

slapi_rdn_get_num_components()

Gets the number of RDN type/value pairs present in a [“Slapi_RDN” on page 269](#) structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_get_num_components(Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

rdn The target [“Slapi_RDN” on page 269](#) structure.

Returns

This function returns the number of RDN type/value pairs present in *rdn*.

See Also

[“slapi_rdn_add\(\)” on page 468](#)

slapi_rdn_get_rdn()

Gets the RDN from a [“Slapi_RDN” on page 269](#) structure.

Syntax

```
#include "slapi-plugin.h"
const char *slapi_rdn_get_rdn(const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

rdn The [“Slapi_RDN” on page 269](#) structure holding the RDN value.

Returns

This function returns the RDN value.

slapi_rdn_init()

Initializes a “[Slapi_RDN](#)” on page 269 structure with NULL values.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init(Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

rdn The “[Slapi_RDN](#)” on page 269 structure to be initialized.

Description

This function initializes a given “[Slapi_RDN](#)” on page 269 structure with NULL values (both the RDN value and the array of split RDNs are set to NULL).

See Also

“[slapi_rdn_done\(\)](#)” on page 471

“[slapi_rdn_free\(\)](#)” on page 472

“[slapi_rdn_new\(\)](#)” on page 480

slapi_rdn_init_dn()

Initializes a “[Slapi_RDN](#)” on page 269 structure with an RDN value taken from a given DN.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init_dn(Slapi_RDN *rdn, const char *dn);
```

Parameters

This function takes the following parameters:

- rdn* The “[Slapi_RDN](#)” on page 269 structure to be initialized.
- dn* The DN value whose RDN will be used to initialize the new “[Slapi_RDN](#)” on page 269 structure.

Description

This function initializes a given “[Slapi_RDN](#)” on page 269 structure with the RDN value taken from the DN passed in the *dn* parameter.

See Also

“[slapi_rdn_init_rdn\(\)](#)” on page 478

“[slapi_rdn_init_sdn\(\)](#)” on page 479

`slapi_rdn_init_rdn()`

Initializes a “[Slapi_RDN](#)” on page 269 structure with an RDN value.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init_rdn(Slapi_RDN *rdn, const Slapi_RDN *fromrdn);
```

Parameters

This function takes the following parameters:

- rdn* The “[Slapi_RDN](#)” on page 269 structure to be initialized.
- fromrdn* The RDN value to be set in the new “[Slapi_RDN](#)” on page 269 structure.

Description

This function initializes a given “[Slapi_RDN](#)” on page 269 structure with the RDN value in *fromrdn*.

See Also

“[slapi_rdn_init_dn\(\)](#)” on page 477

“[slapi_rdn_init_sdn\(\)](#)” on page 479

slapi_rdn_init_sdn()

Initializes a “[Slapi_RDN](#)” on page 269 structure with an RDN value taken from the DN contained in a given “[Slapi_DN](#)” on page 263 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_init_sdn(Slapi_RDN *rdn, const Slapi_DN *sdn);
```

Parameters

This function takes the following parameters:

- rdn* The “[Slapi_RDN](#)” on page 269 structure to be initialized.
- sdn* The “[Slapi_DN](#)” on page 263 structure containing the DN value whose RDN will be used to initialize the new “[Slapi_RDN](#)” on page 269 structure.

Description

This function initializes a given “[Slapi_RDN](#)” on page 269 structure with the RDN value taken from the DN passed within the “[Slapi_DN](#)” on page 263 structure of the *sdn* parameter.

See Also

“[slapi_rdn_init_dn\(\)](#)” on page 477

“[slapi_rdn_init_rdn\(\)](#)” on page 478

slapi_rdn_isempty()

Checks whether an RDN value is stored in a “[Slapi_RDN](#)” on page 269 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_isempty(const Slapi_RDN *rdn);
```

Parameters

This function takes the following parameter:

- rdn* The target “[Slapi_RDN](#)” on page 269 structure.

Returns

This function returns 1 if there is no RDN value present, or 0 if `rdn` contains a value.

See Also

[“`slapi_rdn_init\(\)`” on page 477](#)

[“`slapi_rdn_done\(\)`” on page 471](#)

[“`slapi_rdn_free\(\)`” on page 472](#)

`slapi_rdn_new()`

Allocates a new “[Slapi_RDN](#)” on page 269 structure and initializes the values 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](#)” on page 269 structure.

Description

This function creates a new “[Slapi_RDN](#)” on page 269 structure by allocating the necessary memory and initializing both the RDN value and the array of split RDNs to NULL.

See Also

[“`slapi_rdn_init\(\)`” on page 477](#)

[“`slapi_rdn_done\(\)`” on page 471](#)

[“`slapi_rdn_free\(\)`” on page 472](#)

slapi_rdn_new_dn()

Creates a new “[Slapi_RDN](#)” on page 269 structure and sets an RDN value taken from a given DN.

Syntax

```
#include "slapi-plugin.h"
Slapi_RDN *slapi_rdn_new_dn(const char *dn);
```

Parameters

This function takes the following parameter:

dn The DN value whose RDN will be used to initialize the new “[Slapi_RDN](#)” on page 269 structure.

Returns

This function returns a pointer to the new “[Slapi_RDN](#)” on page 269 structure initialized with the TDN taken from the DN value in *dn*.

Description

This function creates a new “[Slapi_RDN](#)” on page 269 structure and initializes its RDN with the value taken from the DN passed in the *dn* parameter.

Memory Concerns

The memory is allocated by the function itself.

See Also

“[slapi_rdn_new_rdn\(\)](#)” on page 481

“[slapi_rdn_new_sdn\(\)](#)” on page 482

slapi_rdn_new_rdn()

Creates a new “[Slapi_RDN](#)” on page 269 structure and sets an RDN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_RDN * slapi_rdn_new_rdn(const Slapi_RDN *fromrdn);
```

Parameters

This function takes the following parameters:

fromrdn The RDN value to be set in the new “[Slapi_RDN](#)” on page 269 structure.

Returns

This function returns a pointer to the new “[Slapi_RDN](#)” on page 269 structure with an RDN set to the content of *fromrdn*.

Description

This function creates a new “[Slapi_RDN](#)” on page 269 structure and initializes its RDN with the value of *fromrdn*.

Memory Concerns

The memory is allocated by the function itself.

See Also

“[slapi_rdn_new_dn\(\)](#)” on page 481

“[slapi_rdn_new_sdn\(\)](#)” on page 482

`slapi_rdn_new_sdn()`

Creates a new “[Slapi_RDN](#)” on page 269 structure and sets an RDN value taken from the DN contained in a given “[Slapi_RDN](#)” on page 269 structure.

Syntax

```
#include "slapi-plugin.h"
vSlapi_RDN *slapi_rdn_new_sdn(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

sdn “[Slapi_RDN](#)” on page 269 structure containing the DN value whose RDN will be used to initialize the new “[Slapi_RDN](#)” on page 269 structure.

Returns

This function returns a pointer to the new “[Slapi_RDN](#)” on page 269 structure initialized with the RDN taken from the DN value in `dn`.

Description

This function creates a new “[Slapi_RDN](#)” on page 269 structure and initializes its RDN with the value taken from the DN passed within the “[Slapi_RDN](#)” on page 269 structure of the `sdn` parameter.

Memory Concerns

The memory is allocated by the function itself.

See Also

“[slapi_rdn_new_dn\(\)](#)” on page 481

“[slapi_rdn_new_rdn\(\)](#)” on page 481

`slapi_rdn_remove()`

Removes an RDN type/value pair from a “[Slapi_RDN](#)” on page 269 structure.

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 ” on page 269 structure.
<i>type</i>	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 <i>value</i> that should be taken into account for the string comparisons when searching for the RDN.

Returns

This function returns 1 if the RDN is removed from *rdn*, or 0 if no RDN is removed.

Description

This function removes the RDN from *rdn* that matches the given criteria (type, value and length).

See Also

[“slapi_rdn_remove_attr\(\)” on page 484](#)

[“slapi_rdn_remove_index\(\)” on page 485](#)

slapi_rdn_remove_attr()

Removes an RDN type/value pair from a “[Slapi_RDN](#)” on [page 269](#) structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_remove_attr(Slapi_RDN *rdn, const char *type);
```

Parameters

This function takes the following parameters:

rdn The target “[Slapi_RDN](#)” on [page 269](#) structure.

type Type (cn, o, ou, etc.) of the RDN searched.

Returns

This function returns 1 if the RDN is removed from *rdn*, or 0 if no RDN is removed.

Description

This function removes the first RDN from *rdn* that matches the given type.

See Also

[“slapi_rdn_remove\(\)” on page 483](#)

[“slapi_rdn_remove_index\(\)” on page 485](#)

slapi_rdn_remove_index()

Removes an RDN type/value pair from a “[Slapi_RDN](#)” on page 269 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_remove_index(Slapi_RDN *rdn, int atindex);
```

Parameters

This function takes the following parameters:

rdn The target “[Slapi_RDN](#)” on page 269 structure.
atindex The index of the RDN type/value pair to remove.

Returns

This function returns 1 if the RDN is removed from *rdn*, or 0 if no RDN is removed because either *rdn* is empty, or the index goes beyond the number of RDNs present.

Description

This function removes the RDN from *rdn* with *atindex* index (placed in the *atindex*+1 position).

See Also

“[slapi_rdn_remove\(\)](#)” on page 483

“[slapi_rdn_remove_attr\(\)](#)” on page 484

slapi_rdn_set_dn()

Sets an RDN value in a “[Slapi_RDN](#)” on page 269 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_set_dn(Slapi_RDN *rdn, const char *dn);
```

Parameters

This function takes the following parameters:

rdn The target “[Slapi_RDN](#)” on page 269 structure.

dn The DN value whose RDN will be set in *rdn*.

Description

This function sets an RDN value in a “[Slapi_RDN](#)” on page 269 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.

See Also

“[slapi_rdn_set_rdn\(\)](#)” on page 486

“[slapi_rdn_set_sdn\(\)](#)” on page 487

slapi_rdn_set_rdn()

Sets an RDN in a “[Slapi_RDN](#)” on page 269 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_rdn_set_rdn(Slapi_RDN *rdn, const Slapi_RDN *fromrdn);
```

Parameters

This function takes the following parameters:

rdn The target “[Slapi_RDN](#)” on page 269 structure.

fromrdn The “[Slapi_RDN](#)” on page 269 structure from which to take the RDN.

Description

This function sets an RDN value in a “[Slapi_RDN](#)” on page 269 structure. The structure is freed from memory and freed of any previous content before setting the new RDN.

See Also

“[slapi_rdn_set_dn\(\)](#)” on page 485

“[slapi_rdn_set_sdn\(\)](#)” on page 487

slapi_rdn_set_sdn()

Sets an RDN value in a “[Slapi_RDN](#)” on page 269 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:

rdn The target “[Slapi_RDN](#)” on page 269 structure.

sdn The “[Slapi_DN](#)” on page 263 structure from which to take the RDN.

Description

This function sets an RDN value in a “[Slapi_RDN](#)” on page 269 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](#)” on page 263 structure.

See Also

“[slapi_rdn_set_dn\(\)](#)” on page 485

“[slapi_rdn_set_rdn\(\)](#)” on page 486

slapi_rdn_syntax_check()

This function determines whether the value of the RDN complies with attribute syntax rules.

Syntax

```
#include "slapi-plugin.h"
int slapi_rdn_syntax_check( Slapi_PBlock *pb, const char *rdn );
```

Parameters

This function takes the following parameters:

pb Parameter block.

rdn Value whose compliance is to be checked.

Returns

Returns one of the following values:

- 0 if *rdn* complies or if syntax checking is turned off.
- 1 if *rdn* does not comply with attribute syntax rules.

Memory Concerns

The *pb* argument can be NULL. It is used only to get the SLAPI_IS_REPLICATED_OPERATION flag. If that flag is present, no syntax checking is done.

slapi_register_object_extension()

Register an object extension.

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:

pluginname	String identifying the plug-in
objectname	Name of the object to extend such as SLAPI_EXT_CONNECTION to add private data to a connection or SLAPI_EXT_OPERATION to add private data to an operation
constructor	Constructor to allocate memory for the object extension and create the extension
destructor	Destructor to free memory used for the object extension
objecttype	Set by the server and used to retrieve the extension
extensionhandle	Set by the server and used to retrieve the extension

Description

This function registers an extension to an object such as a connection or an operation. This mechanism enables a plug-in to store private data with an operation that is passed from a preoperation to a postoperation plug-in for example, something not possible using parameter blocks.

Register object extensions as part of the plug-in initialization function.

Returns

This function returns 0 if successful. Otherwise, it returns -1.

See Also

[“slapi_extension_constructor_fnptr” on page 264](#)

[“slapi_extension_destructor_fnptr” on page 264](#)

[“slapi_get_object_extension\(\)” on page 398](#)

[“slapi_set_object_extension\(\)” on page 527](#)

slapi_register_plugin()

Register another plug-in.

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:

plugintype	String identifying the plug-in type as described under “Plug-In Registration” on page 584
enabled	1 to enable the plug-in on registration, 0 otherwise
initsymbol	String representation of the plug-in initialization function <code>initfunc</code> such as <code>"my_init_fcn"</code>
initfunc	Pointer to the plug-in initialization function

name	Common Name for the plug-in
argv	Arguments passed to the plug-in
group_identity	Plug-in group identifier, typically obtained from SLAPI_PLUGIN_IDENTITY for the plug-in calling this function

Description

This function registers another plug-in. Register plug-ins as part of the plug-in initialization function.

Returns

This function returns 0 if successful. Otherwise, it returns -1.

See Also

[“slapi_plugin_init_fnptr” on page 269](#)

slapi_register_role_get_scope()

Register a callback to determine the scope of a role.

Syntax

```
#include "slapi-plugin.h"
void slapi_register_role_get_scope(
    roles_get_scope_fn_type get_scope_fn);
```

Parameters

This function takes the following parameters:

get_scope_fn Callback to determine the scope of a role

Description

This function registers the callback that evaluates the scope of a role. Register the callback as part of the plug-in initialization function.

See Also

[“roles_get_scope_fn_type” on page 256](#)

[“slapi_role_get_scope\(\)” on page 495](#)

slapi_register_supported_control()

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 const *controloid,
    unsigned long controlops );
```

Parameters

This function takes the following parameters:

controloid OID of the control you want to register.

controlops Operation that the control is applicable to.

The *controlops* argument can have one or more of the following values:

ID	Description
SLAPI_OPERATION_BIND	The specified control applies to the LDAP bind operation.
SLAPI_OPERATION_UNBIND	The specified control applies to the LDAP unbind operation.
SLAPI_OPERATION_SEARCH	The specified control applies to the LDAP search operation.
SLAPI_OPERATION_MODIFY	The specified control applies to the LDAP modify operation.
SLAPI_OPERATION_ADD	The specified control applies to the LDAP add operation.
SLAPI_OPERATION_DELETE	The specified control applies to the LDAP delete operation.
SLAPI_OPERATION_MODDN	The specified control applies to the LDAP modify DN operation.
SLAPI_OPERATION_MODRDN	The specified control applies to the LDAPv3 modify RDN operation.

ID	Description
SLAPI_OPERATION_COMPARE	The specified control applies to the LDAP compare operation.
SLAPI_OPERATION_ABANDON	The specified control applies to the LDAP abandon operation.
SLAPI_OPERATION_EXTENDED	The specified control applies to the LDAP v3 extended operation.
SLAPI_OPERATION_ANY	The specified control applies to any LDAP operation.
SLAPI_OPERATION_NONE	The specified control applies to none of the LDAP operations.

You can specify a combination of values by bitwise ORing the values together (for example, `SLAPI_OPERATION_ADD | SLAPI_OPERATION_DELETE`).

See Also

[“slapi_control_present\(\)” on page 335](#)

[“slapi_entry_get_uniqueid\(\)” on page 372](#)

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:

mechanism String identifying the SASL mechanism to both the server and to clients requesting the mechanism during a SASL bind

Description

Use this function in the plug-in initialization function to register the name of a SASL mechanism supported by the plug-in. The preoperation function handling the SASL bind can then check the SASL bind mechanism name provided by the client to determine whether to attempt to handle the bind.

See Also

The sample `$INSTALL_DIR/plugins/slapd/slapi/examples/testsaslbinding.c` plug-in demonstrates the use of this function.

slapi_rename_internal_set_pb()

Prepares a parameter block for an internal modify RDN operation.

Syntax

```
#include "slapi-plugin.h"
int slapi_rename_internal_set_pb(Slapi_PBlock *pb,
    const char *olddn, const char *newrdn,
    const char *newsuperior, int delolddn,
    LDAPControl **controls, const char *uniqueid,
    Slapi_ComponentId *plugin_identity, int operation_flags);
```

Parameters

This function takes the following parameters:

<code>pb</code>	Parameter block for the internal modify RDN operation
<code>olddn</code>	Distinguished Name of the entry to rename
<code>newrdn</code>	New Relative Distinguished name to apply
<code>newsuperior</code>	DN of parent after renaming
<code>delolddn</code>	1 to delete the old RDN, 0 to retain the old RDN
<code>controls</code>	Array of controls to request for the modify RDN operation
<code>uniqueid</code>	Unique identifier for the entry if using this rather than DN
<code>plugin_identity</code>	Plug-in identifier obtained from <code>SLAPI_PLUGIN_IDENTITY</code> during plug-in initialization
<code>operation_flags</code>	NULL or <code>SLAPI_OP_FLAG_NEVER_CHAIN</code>

Returns

This function returns 0 if successful. Otherwise, it returns an LDAP error code.

Description

This function prepares a parameter block for use with “`slapi_modrdn_internal_pb()`” on [page 433](#).

Memory Concerns

Allocate the parameter block using “[slapi_pblock_new\(\)](#)” on page 464 before calling this function.

Directory Server does not free the parameters you passed to this function.

Free the parameter block after calling “[slapi_modrdn_internal_pb\(\)](#)” on page 433 .

See Also

“[slapi_modrdn_internal_pb\(\)](#)” on page 433

“[slapi_pblock_new\(\)](#)” on page 464

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 is placed. For results: <ul style="list-style-type: none">▪ non 0 means present▪ 0 means not present

Returns

This function returns one of the following values:

- 0 for success if `role_dn` is present in `entry_to_check` and `present` is set to non-zero. Otherwise it is 0.
- non-zero (error condition) if the presence of the role is undetermined.

slapi_role_get_scope()

Determine the scope of a role.

Syntax

```
#include "slapi-plugin.h"
int slapi_role_get_scope(Slapi_Entry *role_entry,
                        Slapi_DN ***scope_dn, int *nb_scope);
```

Parameters

This function takes the following parameters:

<code>role_entry</code>	Entry defining the role
<code>scope_dn</code>	Set by the callback to the Distinguished Name of the entry at the base of the scope
<code>nb_scope</code>	Set by the callback to a value such as <code>LDAP_SCOPE_BASE</code> , <code>LDAP_SCOPE_ONELEVEL</code> , or <code>LDAP_SCOPE_SUBTREE</code>

Description

This function triggers a callback to evaluate the scope of a role.

Memory Concerns

Directory Server does not free or copy the parameters passed to this function.

See Also

[“roles_get_scope_fn_type” on page 256](#)

[“slapi_register_role_get_scope\(\)” on page 490](#)

slapi_sdn_add_rdn()

Adds the RDN contained in a [“Slapi_RDN” on page 269](#) structure to the DN contained in a [“Slapi_DN” on page 263](#) 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:

sdn [“Slapi_DN” on page 263](#) structure containing the value to which a new RDN is to be added.

rdn [“Slapi_RDN” on page 269](#) structure containing the RDN value that is to be added to the DN value.

Returns

This function returns the [“Slapi_DN” on page 263](#) 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\(\)” on page 517](#)

slapi_sdn_compare()

Compares two DNs.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_compare( const Slapi_DN *sdn1, const Slapi_DN *sdn2 );
```

Parameters

This function takes the following parameters:

sdn1 DN to compare with the value in *sdn2*.

sdn2 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.

Description

This function compares two DNs, *sdn1* and *sdn2*. The comparison is case sensitive.

slapi_sdn_copy()

Makes a copy of a DN.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_copy(const Slapi_DN *from, Slapi_DN *to);
```

Parameters

This function takes the following parameters:

from The original DN.

to Destination of the copied DN, containing the copy of the DN in *from*.

Description

This function copies the DN in *from* to the structure pointed by *to*.

Memory Concerns

to must be allocated in advance of calling this function.

See Also

[“slapi_sdn_dup\(\)” on page 498](#)

slapi_sdn_done()

Clears an instance of a [“Slapi_DN” on page 263](#) structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_done(Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

sdn Pointer to the structure to clear.

Description

This function clears the contents of a “[Slapi_DN](#)” on page 263 structure. It frees both the DN and the normalized DN, if any, and sets those pointers to NULL.

See Also

“[slapi_sdn_free\(\)](#)” on page 498

slapi_sdn_dup()

Duplicates a “[Slapi_DN](#)” on page 263 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN * slapi_sdn_dup(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

sdn Pointer to the “[Slapi_DN](#)” on page 263 structure to duplicate.

Returns

This function returns a pointer to a duplicate of *sdn*.

See Also

“[slapi_sdn_copy\(\)](#)” on page 497

“[slapi_sdn_new\(\)](#)” on page 506

slapi_sdn_free()

Frees a “[Slapi_DN](#)” on page 263 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_free(Slapi_DN **sdn);
```

Parameters

This function takes the following parameter:

sdn Pointer to the “[Slapi_DN](#)” on page 263 structure to free.

Description

This function frees the “[Slapi_DN](#)” on page 263 structure and its contents pointed to by the contents of *sdn*.

See Also

“[slapi_sdn_done\(\)](#)” on page 497

“[slapi_sdn_new\(\)](#)” on page 506

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:

sdn DN of the entry whose parent is searched.

sdn_parent Parent DN of *sdn*.

backend Backend of which the parent of *sdn* is to be searched.

Description

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](#)” on [page 263](#) structure for `sdn_parent` must be allocated before calling this function.

See Also

“[slapi_sdn_get_parent\(\)](#)” on [page 502](#)

`slapi_sdn_get_dn()`

Gets the DN from a “[Slapi_DN](#)” on [page 263](#) structure.

Syntax

```
#include "slapi-plugin.h"
const char * slapi_sdn_get_dn(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

sdn The “[Slapi_DN](#)” on [page 263](#) structure containing the DN value.

Returns

This function returns the DN value.

Description

This function retrieves the DN value of a “[Slapi_DN](#)” on [page 263](#) structure. The returned value can be the normalized DN (in a canonical format and in lower case) if no other value is present.

See Also

“[slapi_sdn_get_ndn\(\)](#)” on [page 500](#)

“[slapi_sdn_get_parent\(\)](#)” on [page 502](#)

“[slapi_sdn_get_rdn\(\)](#)” on [page 502](#)

`slapi_sdn_get_ndn()`

Gets the normalized DN of a “[Slapi_DN](#)” on [page 263](#) structure.

Syntax

```
#include "slapi-plugin.h"
const char * slapi_sdn_get_ndn(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

sdn The “[Slapi_DN](#)” on page 263 structure containing the DN value.

Returns

This function returns the normalized DN value.

Description

This function retrieves the normalized DN (in a canonical format and lower case) from a “[Slapi_DN](#)” on page 263 structure and normalizes *sdn* if it has not already been normalized.

See Also

“[slapi_sdn_get_dn\(\)](#)” on page 500

slapi_sdn_get_ndn_len()

Gets the length of the normalized DN of a “[Slapi_DN](#)” on page 263 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_get_ndn_len(const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

sdn The “[Slapi_DN](#)” on page 263 structure containing the DN value.

Returns

This function returns the length of the normalized DN.

Description

This function contains the length of the normalized DN and normalizes sdn if it has not already been normalized.

slapi_sdn_get_parent()

Gets the parent DN of a given “[Slapi_DN](#)” on [page 263](#) structure.

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 initialized “ Slapi_DN ” on page 263 structure containing the DN whose parent is searched.
<i>sdn_parent</i>	Pointer to the “ Slapi_DN ” on page 263 structure where the parent DN is returned.

Description

This function returns a “[Slapi_DN](#)” on [page 263](#) structure containing the parent DN of the DN kept in the structure pointed to by sdn.

See Also

“[slapi_sdn_get_backend_parent\(\)](#)” on [page 499](#)

slapi_sdn_get_rdn()

Gets the RDN from a DN.

Syntax

```
#include "slapi-plugin.h"
void slapi_sdn_get_rdn(const Slapi_DN *sdn, Slapi_RDN *rdn);
```

Parameters

This function takes the following parameters:

- sdn* Pointer to the “[Slapi_DN](#)” on page 263 structure containing the DN.
- rdn* Pointer to the “[Slapi_RDN](#)” on page 269 structure where the RDN is returned.

Description

This function takes the DN stored in the “[Slapi_DN](#)” on page 263 structure pointed to by *sdn* and retrieves its returned RDN within the “[Slapi_RDN](#)” on page 269 structure pointed to by *rdn*.

See Also

“[slapi_sdn_add_rdn\(\)](#)” on page 495

“[slapi_sdn_get_dn\(\)](#)” on page 500

slapi_sdn_get_suffix()

Returns a pointer to the DN of the suffix containing the target.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN* slapi_sdn_get_suffix(const Slapi_DN *target_sdn);
```

Parameters

This function takes the following parameter:

- target_sdn* DN of the target entry whose suffix you want.

Returns

This function returns a pointer to a “[Slapi_DN](#)” on page 263 structure containing the base DN of the suffix for the entry with *target_sdn*, or NULL if that suffix is not available.

Memory Concerns

Free the structure returned using “[slapi_sdn_free\(\)](#)” on page 498.

slapi_sdn_isempty()

Checks whether there is a DN value stored in a “[Slapi_DN](#)” on page 263 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_sdn_isempty( const Slapi_DN *sdn);
```

Parameters

This function takes the following parameter:

sdn Pointer to the “[Slapi_DN](#)” on page 263 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](#)” on page 263 structure.
- 0 if *sdn* is not empty.

Description

This function checks whether a “[Slapi_DN](#)” on page 263 structure contains a normalized or non-normalized value.

See Also

“[slapi_sdn_done\(\)](#)” on page 497

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:

- parent* Pointer to the “[Slapi_DN](#)” on page 263 structure containing the DN that claims to be the grandparent DN of the DN in *child*.
- child* Pointer to the “[Slapi_DN](#)” on page 263 structure containing the DN of the supposed “grandchild” of the DN in the structure pointed to by *parent*.

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_get_parent\(\)” on page 502](#)

[“slapi_sdn_isparent\(\)” on page 505](#)

[“slapi_sdn_issuffix\(\)” on page 506](#)

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:

- parent* Pointer to the “[Slapi_DN](#)” on page 263 structure containing the DN that claims to be the parent of the DN in *child*.
- child* Pointer to the “[Slapi_DN](#)” on page 263 structure containing the DN of the supposed child of the DN in the structure pointed to by *parent*.

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_get_parent\(\)” on page 502](#)

[“slapi_sdn_issuffix\(\)” on page 506](#)

slapi_sdn_issuffix()

Checks whether a [“Slapi_DN” on page 263](#) structure contains a suffix of another [“Slapi_DN” on page 263](#) 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” on page 263 structure to be checked.
<i>suffixsdn</i>	Pointer to the “Slapi_DN” on page 263 structure of the suffix.

Returns

This function returns one of the following values:

- 1 if the DN is *suffixsdn* is the suffix of *sdn*.
- 0 if the DN in *suffixsdn* is not a suffix of *sdn*.

See Also

[“slapi_sdn_isparent\(\)” on page 505](#)

slapi_sdn_new()

Allocates a new [“Slapi_DN” on page 263](#) structure and initializes it 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](#)” on page 263 structure.

Description

This function creates a new “[Slapi_DN](#)” on page 263 structure by allocating the necessary memory and initializing both DN and normalized DN values to NULL.

See Also

“[slapi_sdn_copy\(\)](#)” on page 497

“[slapi_sdn_done\(\)](#)” on page 497

“[slapi_sdn_free\(\)](#)” on page 498

slapi_sdn_new_dn_byref()

Creates a new “[Slapi_DN](#)” on page 263 structure and sets a DN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_dn_byref(const char *dn);
```

Parameters

This function takes the following parameter:

dn The DN value to be set in the new “[Slapi_DN](#)” on page 263 structure.

Returns

This function returns a pointer to the new “[Slapi_DN](#)” on page 263 structure with a DN value set to the content of dn.

Description

This function creates a new “[Slapi_DN](#)” on page 263 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.

Memory Concerns

The memory is allocated by the function itself.

See Also

“[slapi_sdn_new_dn_byval\(\)](#)” on page 508

“[slapi_sdn_new_dn_passin\(\)](#)” on page 509

slapi_sdn_new_dn_byval()

Creates a new “[Slapi_DN](#)” on page 263 structure and sets a DN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_dn_byval(const char *dn);
```

Parameters

This function takes the following parameter:

dn The DN value to be set in the new “[Slapi_DN](#)” on page 263 structure.

Returns

This function returns a pointer to the new “[Slapi_DN](#)” on page 263 structure with a DN value set to the content of dn.

Description

This function creates a new “[Slapi_DN](#)” on page 263 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.

Memory Concerns

The memory is allocated by the function itself.

See Also

“[slapi_sdn_new_dn_byref\(\)](#)” on page 507

“[slapi_sdn_new_dn_passin\(\)](#)” on page 509

`slapi_sdn_new_dn_passin()`

Creates a new “[Slapi_DN](#)” on page 263 structure and sets a DN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_dn_passin(const char *dn);
```

Parameters

This function takes the following parameter:

dn The DN value to be set in the new “[Slapi_DN](#)” on page 263 structure.

Returns

This function returns a pointer to the new “[Slapi_DN](#)” on page 263 structure with DN value set to the content of `dn`.

Description

This function creates a new “[Slapi_DN](#)” on page 263 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.

Memory Concerns

The memory is allocated by the function itself.

See Also

[“slapi_sdn_new_dn_byref\(\)” on page 507](#)

[“slapi_sdn_new_dn_byval\(\)” on page 508](#)

slapi_sdn_new_ndn_byref()

Creates a new “[Slapi_DN](#)” [on page 263](#) structure and sets a normalized DN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_ndn_byref(const char *ndn);
```

Parameters

This function takes the following parameter:

ndn The normalized DN value to be set in the new “[Slapi_DN](#)” [on page 263](#) structure.

Returns

This function returns a pointer to the new “[Slapi_DN](#)” [on page 263](#) structure with a normalized DN value set to the content of *ndn*.

Description

This function creates a new “[Slapi_DN](#)” [on page 263](#) 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.

Memory Concerns

The memory is allocated by the function itself.

See Also

[“slapi_sdn_new_ndn_byval\(\)” on page 511](#)

slapi_sdn_new_ndn_byval()

Creates a new “[Slapi_DN](#)” on page 263 structure and sets a normalized DN value.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_new_ndn_byval(const char *ndn);
```

Parameters

This function takes the following parameter:

ndn The normalized DN value to be set in the new “[Slapi_DN](#)” on page 263 structure.

Returns

This function returns a pointer to the new “[Slapi_DN](#)” on page 263 structure with a normalized DN value set to the content of *ndn*.

Description

This function creates a new “[Slapi_DN](#)” on page 263 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.

Memory Concerns

The memory is allocated by the function itself.

See Also

“[slapi_sdn_new_ndn_byref\(\)](#)” on page 510

slapi_sdn_scope_test()

Checks whether an entry, given its DN, is in the scope of a certain base DN.

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:

- dn* The DN of the entry subject of scope test.
- base* The base DN to which dn is going to be tested against.
- scope* The scope tested. This parameter can take one of the following levels
- 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 DN
 - LDAP_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.

Description

This function carries out a simple test to check whether the DN passed in the dn parameter is actually in scope of the base DN according to the values passed into the scope and base parameters.

See Also

[“slapi_sdn_compare\(\)” on page 496](#)

[“slapi_sdn_isparent\(\)” on page 505](#)

[“slapi_sdn_issuffix\(\)” on page 506](#)

slapi_sdn_set_dn_byref()

Sets a DN value in a “[Slapi_DN](#)” [on page 263](#) structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_dn_byref(Slapi_DN *sdn, const char *dn);
```

Parameters

This function takes the following parameters:

sdn The target “[Slapi_DN](#)” on page 263 structure.

dn The DN value to be set in *sdn*.

Returns

This function returns a pointer to the “[Slapi_DN](#)” on page 263 structure containing the new DN value.

Description

This function sets a DN value in a “[Slapi_DN](#)” on page 263 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.

See Also

“[slapi_sdn_set_dn_byval\(\)](#)” on page 513

“[slapi_sdn_set_dn_passin\(\)](#)” on page 514

`slapi_sdn_set_dn_byval()`

Sets a DN value in a “[Slapi_DN](#)” on page 263 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_dn_byval(Slapi_DN *sdn, const char *dn);
```

Parameters

This function takes the following parameters:

sdn The target “[Slapi_DN](#)” on page 263 structure.

dn The DN value to be set in *sdn*.

Returns

This function returns a pointer to the “[Slapi_DN](#)” on page 263 structure containing the new DN value.

Description

This function sets a DN value in a “[Slapi_DN](#)” on page 263 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.

See Also

“[slapi_log_info_ex\(\)](#)” on page 409

“[slapi_sdn_set_dn_passin\(\)](#)” on page 514

`slapi_sdn_set_dn_passin()`

Sets a DN value in “[Slapi_DN](#)” on page 263 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_DN *slapi_sdn_set_dn_passin(Slapi_DN *sdn, const char *dn);
```

Parameters

This function takes the following parameters:

sdn The target “[Slapi_DN](#)” on page 263 structure.

dn The DN value to be set in *sdn*.

Returns

This function returns a pointer to the “[Slapi_DN](#)” on page 263 structure containing the new DN value.

Description

This function sets a DN value in a “[Slapi_DN](#)” on page 263 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.

See Also

“[slapi_log_info_ex\(\)](#)” on page 409

“[slapi_sdn_set_dn_byval\(\)](#)” on page 513

slapi_sdn_set_ndn_byref()

Sets a normalized DN in a “[Slapi_DN](#)” on page 263 structure.

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:

- sdn* The target “[Slapi_DN](#)” on page 263 structure.
- dn* The normalized DN value to be set in *sdn*.

Returns

This function returns a pointer to the “[Slapi_DN](#)” on page 263 structure containing the new normalized DN value.

Description

This function sets a normalized DN value in a “[Slapi_DN](#)” on page 263 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.

See Also

“[slapi_sdn_set_ndn_byval\(\)](#)” on page 515

“[slapi_sdn_set_dn_passin\(\)](#)” on page 514

slapi_sdn_set_ndn_byval()

Sets a normalized DN value in “[Slapi_DN](#)” on page 263 structure.

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:

sdn The target “[Slapi_DN](#)” on page 263 structure.

dn The normalized DN value to be set in *sdn*.

Returns

This function returns a pointer to the “[Slapi_DN](#)” on page 263 structure containing the new normalized DN value.

Description

This function sets a normalized DN value in a “[Slapi_DN](#)” on page 263 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.

See Also

“[slapi_log_info_ex\(\)](#)” on page 409

“[slapi_sdn_set_dn_passin\(\)](#)” on page 514

`slapi_sdn_set_parent()`

Sets a new parent for a given entry.

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:

sdn The “[Slapi_DN](#)” on page 263 structure containing the DN of the entry.

parentdn The new parent DN.

Returns

The function returns a pointer to the “[Slapi_DN](#)” on page 263 structure that contains the DN of the entry after the new parent DN has been set.

Description

This function sets a new parent for an 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`).

See Also

“[slapi_sdn_get_parent\(\)](#)” on page 502

“[slapi_sdn_isparent\(\)](#)” on page 505

`slapi_sdn_set_rdn()`

Sets a new RDN for given entry.

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:

sdn The “[Slapi_DN](#)” on page 263 structure containing the DN of the entry.

rdn The new RDN.

Returns

This function returns a pointer to the “[Slapi_DN](#)” on page 263 structure that keeps the DN of the entry after the new RDN has been set.

Description

This function sets a new RDN for an 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.

See Also

[“slapi_sdn_get_rdn\(\)” on page 502](#)

slapi_search_internal_callback_pb()

Performs an LDAP search operation based on a parameter block to search the directory. Unlike [“slapi_search_internal_pb\(\)” on page 521](#), 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 “slapi_search_internal_set_pb()” on page 521 .
<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” on page 254 .
<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” on page 255 .
<i>prec</i>	Callback function that the server calls when finding an entry that contains LDAP v3 referrals. The function must have the prototype specified by “plugin_referral_entry_callback” on page 253 .

Returns

This function returns -1 if the parameter passed is a NULL pointer. Otherwise, it returns 0.

After your code calls this function, the server sets `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to the appropriate LDAP result code. You can therefore check `SLAPI_PLUGIN_INTOP_RESULT` in the parameter block to determine whether an error has occurred.

Description

Like [“`slapi_search_internal_pb\(\)`” on page 521](#), this function allows you to search the directory from a plug-in function.

Unlike a search operation requested by a client, no result code, search entries, or referrals are sent to a client by `slapi_search_internal_callback_pb()`. However, you can write your own callback functions that are invoked when these events occur:

- You can write a callback function that is invoked when the search operation normally sends a result code to the client. This function must have the prototype specified by [“`plugin_result_callback`” on page 254](#). You specify this function in the `prc` argument of `slapi_search_internal_callback_pb()`.
- You can write a callback function that is invoked when the search operation normally sends a search entry to the client. This function must have the prototype specified by [“`plugin_search_entry_callback`” on page 255](#). You specify this function in the `psec` argument of `slapi_search_internal_callback_pb()`.
- You can write a callback function that is invoked when the search operation normally sends LDAP v3 search result references. This function must have the prototype specified by [“`plugin_referral_entry_callback`” on page 253](#). You specify this function in the `prec` argument of `slapi_search_internal_callback_pb()`.

You can also pass arbitrary plug-in or operation-specific data to these callback functions. Specify the data that you want to pass as the `callback_data` argument of `slapi_search_internal_callback_pb()`.

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\(\)`” on page 366](#) 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\(\)`” on page 332](#) to make a copy.

You do not need to call [“`slapi_free_search_results_internal\(\)`” on page 396](#) after calling `slapi_search_internal_callback_pb()`.

slapi_search_internal_get_entry()

Performs an internal search operation to read one entry (that is, it performs a base object search).

Syntax

```
#include "slapi-plugin.h"
int slapi_search_internal_get_entry( Slapi_DN const *dn, char ** attrlist,
    Slapi_Entry **ret_entry, void *caller_identity);
```

Parameters

This function takes the following parameters:

<i>dn</i>	The DN of the entry to be read.
<i>attrlist</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 “ Slapi_Entry ” on page 263 pointer to receive the entry if it is found.
<i>caller_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.

Description

This function performs an internal search operation to read one entry (that is, it preforms 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 *attrlist* parameter.

Memory Concerns

The returned entry (**ret_entry*) should be freed by calling “[slapi_entry_free\(\)](#)” on [page 368](#).

See Also

“[slapi_entry_free\(\)](#)” on [page 368](#)

“[slapi_search_internal_pb\(\)](#)” on [page 521](#)

slapi_search_internal_pb()

Performs an LDAP search operation based on a parameter block to search the directory.

Syntax

```
#include "slapi-plugin.h"
int slapi_search_internal_pb(Slapi_PBlock *pb);
```

Parameters

This function takes the following parameter:

pb A parameter block that has been initialized using “[slapi_search_internal_set_pb\(\)](#)” [on page 521](#).

Returns

This function returns -1 if the parameter passed is a NULL pointer. Otherwise, it returns 0.

After your code calls this function, the server sets SLAPI_PLUGIN_INTOP_RESULT in the parameter block to the appropriate LDAP result code. You can therefore check SLAPI_PLUGIN_INTOP_RESULT in the parameter block to determine whether an error has occurred.

Description

This function performs an internal search based on a parameter block. The parameter block should be initialized by calling the “[slapi_search_internal_set_pb\(\)](#)” [on page 521](#) function.

Memory Concerns

“[slapi_free_search_results_internal\(\)](#)” [on page 396](#) should be called to dispose of any entires and other items that were allocated by a call to `slapi_search_internal_pb()`.

slapi_search_internal_set_pb()

Sets a parameter block for an internal search.

Syntax

```
#include slapi-plugin.h
int 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.
<i>base</i>	The base of the search.
<i>scope</i>	LDAP_SCOPE_SUBTREE, LDAP_SCOPE_ONELEVEL , or LDAP_SCOPE_BASE
<i>filter</i>	The search filter.
<i>attrs</i>	Request attribute list.
	This parameter is currently ignored.
<i>attrsonly</i>	Specifying 1 retrieves only the attribute types. Specifying 0 retrieves the attribute types and the attribute values.
<i>controls</i>	LDAP control that you wish to attach.
<i>uniqueid</i>	Unique ID must be set to NULL.
<i>plugin_identity</i>	Plug-in identifier obtained from SLAPI_PLUGIN_IDENTITY during plug-in initialization.
<i>operation_flags</i>	The only flag that is exposed in this release is SLAPI_OP_FLAG_NEVER_CHAIN . If this flag is set, then the search will not be conducted if the entry is a chained backend.

Returns

This function returns 0 if successful. Otherwise, it returns an LDAP error code.

Description

Use this function to set the parameter block to perform an internal search. This function is needed in order to use the internal search operation function, [“slapi_search_internal_pb\(\)” on page 521](#).

You would typically create a [“Slapi_PBlock” on page 267](#) structure using [“slapi_pblock_new\(\)” on page 464](#), pass that parameter block to [“slapi_search_internal_pb\(\)” on page 521](#), and then pass the same parameter block to actually do the search.

Memory Concerns

Allocate the parameter block using “[slapi_pblock_new\(\)](#)” on page 464 before calling this function.

Directory Server does not free the parameters you passed to this function.

Free the parameter block after calling “[slapi_search_internal_pb\(\)](#)” on page 521 .

See Also

“[slapi_pblock_new\(\)](#)” on page 464

“[slapi_search_internal_pb\(\)](#)” on page 521

slapi_send_ldap_referral()

Processes an entry’s LDAP v3 referrals (which are found in the entry’s `ref` attribute). For LDAP v3 clients, this function sends the LDAP referrals back to the client. For LDAP v2 clients, this function copies the referrals to an array of “[berval](#)” on page 249 structures that you can pass to “[slapi_send_ldap_result\(\)](#)” on page 524 function at a later time.

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:

- pb* Parameter block.
- e* Pointer to the “[Slapi_Entry](#)” on page 263 structure representing the entry that you are working with.
- refs* Pointer to the NULL terminated array of “[berval](#)” on page 249 structures containing the LDAP v3 referrals (search result references) found in the entry.
- urls* Pointer to the array of “[berval](#)” on page 249 structures used to collect LDAP referrals for LDAP v2 clients.

Returns

This function returns 0 if successful, or -1 if an error occurs.

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 LDAP v3 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.)

When you call the `slapi_send_ldap_referral()` function for LDAP v3 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 LDAP v2 protocol, servers can send the LDAP result code `LDAP_PARTIAL_RESULTS` to refer the client to other LDAP server.

When you call the `slapi_send_ldap_referral()` function for LDAP v2 clients, the server collects the referrals specified in `refs` in the `urls` argument. No data is sent to the LDAP v2 client.

To get the referrals to an LDAP v2 client, you need to pass the `urls` argument (along with an `LDAP_PARTIAL_RESULTS` result code) to the [“`slapi_send_ldap_result\(\)`” on page 524](#) function. [“`slapi_send_ldap_result\(\)`” on page 524](#) 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_search_entry_fn_ptr_t`” on page 258](#) and set the `SLAPI_PLUGIN_DB_REFERRAL_FN` parameter in the parameter block to the name of your function.

See Also

[“`slapi_send_ldap_result\(\)`” on page 524](#)

[“`slapi_send_ldap_search_entry\(\)`” on page 526](#)

`slapi_send_ldap_result()`

Sends an LDAP result code back to the client.

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 LDAP v2 client or an LDAP_REFERRAL result code to an LDAP v3 client, use this argument to specify the array of “ berval ” on page 249 structures containing the referral URLs. (Pass NULL in other situations.)

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, o=Example, c=US
```

and the database contains entries for `c=US` and `o=Example, c=US`, but no entry for `ou=Product Division, o=Example, c=US`, you should set the `matched` parameter to:

```
o=Example, c=US
```

- **urls**

When sending an LDAP_PARTIAL_RESULTS result code back to an LDAP v2 client or an LDAP_REFERRAL result code back to an LDAP v3 client, use `urls` to specify the referral URLs.

For LDAP v3 referrals, you can call `slapi_send_ldap_result()` to send referrals to LDAP v3 clients and collect them for LDAP v2 clients. You can pass the array of collected referrals to the `urls` argument of `slapi_send_ldap_result()`. 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_search_entry_fn_ptr_t](#)” on page 258 and set the `SLAPI_PLUGIN_DB_RESULT_FN` parameter in the parameter block to the name of your function.

See Also

“[slapi_send_ldap_referral\(\)](#)” on page 523

“[slapi_send_ldap_search_entry\(\)](#)” on page 526

slapi_send_ldap_search_entry()

Sends an entry found by a search back to the client.

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 “ Slapi_Entry ” on page 263 structure representing the entry that you want to send back to the client.
<i>ectrls</i>	Pointer to the array of “ LDAPControl ” on page 250 structures representing the controls associated with the search request.
<i>attrs</i>	Attribute types specified in the LDAP search request
<i>attrsonly</i>	Specifies whether or not 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 0 if successful, 1 if the entry is not sent (for example, if access control did not allow it to be sent), or -1 if an error occurs.

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](#)” on page 258 and set the `SLAPI_PLUGIN_DB_ENTRY_FN` parameter in the parameter block to the name of your function.

See Also

“[slapi_send_ldap_referral\(\)](#)” on page 523

“[slapi_send_ldap_result\(\)](#)” on page 524

`slapi_set_object_extension()`

Modify an object extension.

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>	Set by the server and used to retrieve the extension
<i>object</i>	Extended object
<i>extensionhandle</i>	Set by the server and used to retrieve the extension
<i>extension</i>	New extension to attach to the object

Description

This function modifies an extension to an object.

See Also

[“slapi_register_object_extension\(\)” on page 488](#)

slapi_str2entry()

Converts an LDIF description of a directory entry (a string value) into an entry of the [“Slapi_Entry” on page 263](#) type.

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 “Slapi_Entry” on page 263 .
<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_ADDRDNVALS</i>	Adds the relative distinguished name (RDN) components (for example, <code>uid=bjensen</code>) as attributes of the entry.

Returns

Pointer to the “[Slapi_Entry](#)” on page 263 structure representing the entry, or NULL if the string cannot be converted (for example, if no DN is specified in the string).

Description

A directory entry can be described by a string in LDIF format.

Calling the `slapi_str2entry()` function converts a string description in this format to a “[Slapi_Entry](#)” on page 263 structure, which you can pass to other API functions.

Note – This function modifies the string argument `s`. If you must use the string value again, 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 finished working with the entry, you should call the “[slapi_entry_free\(\)](#)” on page 368 function.

To convert an entry to a string description, call the “[slapi_entry2str\(\)](#)” on page 348 function.

Memory Concerns

Do not use `free()` or `slapi_ch_free()` to free the entry. Always use “[slapi_entry_free\(\)](#)” on page 368 instead.

See Also

“[slapi_entry2str\(\)](#)” on page 348

“[slapi_entry_free\(\)](#)” on page 368

`slapi_str2filter()`

Converts a string description of a search filter into a filter of the “[Slapi_Filter](#)” on page 265 type.

Syntax

```
#include "slapi-plugin.h"
Slapi_Filter *slapi_str2filter( char *str );
```

Parameters

This function takes the following parameter:

str String description of a search filter.

Memory Concerns

Do not pass a static string to this function. Instead, create a duplicate using [“slapi_ch_strdup\(\)” on page 332](#). When you are finished working with this filter, you should free the [“Slapi_Filter” on page 265](#) structure by calling [“slapi_filter_free\(\)” on page 383](#).

Returns

Pointer to the [“Slapi_Filter” on page 265](#) structure representing the search filter, or NULL if the string cannot be converted (for example, if an empty string is specified or if the filter syntax is incorrect).

slapi_unlock_mutex()

Unlocks the specified mutex.

Syntax

```
#include "slapi-plugin.h"
int slapi_unlock_mutex( Slapi_Mutex *mutex );
```

Parameters

This function takes the following parameters:

mutex Pointer to an [“Slapi_Mutex” on page 266](#) structure representing the mutex that you want to unlock.

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.

Description

This function unlocks the mutex specified by the [“Slapi_Mutex” on page 266](#) structure.

See Also

[“slapi_destroy_mutex\(\)” on page 338](#)

[“slapi_lock_mutex\(\)” on page 407](#)

[“slapi_new_mutex\(\)” on page 458](#)

slapi_UTF-8CASECMP()

Compares two UTF-8 strings.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF-8CASECMP(char *s0, char *s1);
```

Parameters

This function takes the following parameters:

- s0* A NULL terminated UTF-8 string.
- s1* A NULL terminated UTF-8 string.

Returns

This function returns one of the following values:

- positive number if *s0* is after *s1*.
- 0 if the two string are identical, ignoring case.
- negative number if *s1* is after *s0*.

Description

This function has the following rules:

- If both UTF-8 strings are NULL or 0-length, 0 is returned.
- If one of the strings is NULL or 0-length, the NULL or 0-length string is smaller.
- If one or both of the strings are not UTF-8, system provided `strcasemp` is used.
- If one of the two strings contains no 8-bit characters, `strcasemp` is used.
- The strings are compared after they are converted to lowercase 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.

slapi_UTF-8NCASECMP()

Compares a specified number of UTF-8 characters.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF-8NCASECMP(char *s0, char *s1, int n);
```

Parameters

This function takes the following parameters:

- s0* A NULL terminated UTF-8 string.
- s1* A NULL terminated UTF-8 string.
- n* The number of UTF-8 characters (not bytes) from *s0* and *s1* to compare.

Returns

This function returns one of the following values:

- positive number if *s0* is after *s1*.
- 0 if the two string are identical, ignoring case.
- negative number if *s1* is after *s0*.

Description

This function has the following rules:

- If both UTF-8 strings are NULL or 0-length, 0 is returned.
- If one of the strings is NULL or 0-length, the NULL or 0-length string is smaller.
- If one or both of the strings are not UTF-8, system provided `strcascmp` is used.
- If one of the two strings contains no 8-bit characters, `strcascmp` 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.

slapi_UTF-8ISLOWER()

Verifies if a UTF-8 character is lower case.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF-8ISLOWER(char *s);
```

Parameters

This function takes the following parameter:

s Pointer to a single UTF-8 character (could be multiple bytes).

Returns

This function returns 1 if the character is a lower case letter, or 0 if it is not.

slapi_UTF-8ISUPPER()

Verifies if a single UTF-8 character is upper case.

Syntax

```
#include "slapi-plugin.h"
int slapi_UTF-8ISUPPER(char *s);
```

Parameters

This function takes the following parameter:

s Pointer to a single UTF-8 character (could be multiple bytes).

Returns

This function returns 1 if the character is an upper case letter, or 0 if it is not.

slapi_UTF-8STRTOLOWER()

Converts a UTF-8 string to lower case.

Syntax

```
#include "slapi-plugin.h"
unsigned char *slapi_UTF-8STRTOLOWER(char *s);
```

Parameters

This function takes the following parameter:

s A NULL terminated UTF-8 string to be converted to lower case.

Returns

This function returns 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. If the string is not found to be a UTF-8 string, this function returns NULL.

Description

This function converts a string of multiple UTF-8 characters, and not a single character, as in [“slapi_UTF-8TOLOWER\(\)” on page 535](#).

Memory Concerns

The output string is allocated, and needs to be released when it is no longer needed.

See Also

[“slapi_UTF-8TOLOWER\(\)” on page 535](#)

slapi_UTF-8STRTOUPPER()

Converts a string made up of UTF-8 characters and converts it to upper case.

Syntax

```
#include "slapi-plugin.h"
unsigned char *slapi_UTF-8STRTOUPPER(char *s);
```

Parameters

This function takes the following parameter:

s A NULL terminated UTF-8 string to be converted to upper case.

Returns

This function returns a NULL terminated UTF-8 string whose characters are converted to upper case. Character which are not lower case are copied as is. If the string is not considered to be a UTF-8 string, this function returns NULL.

Memory Concerns

The output string is allocated in this function, and needs to be released when it is no longer used.

slapi_UTF-8TOLOWER()

Converts a UTF-8 character to lower case.

Syntax

```
#include "slapi-plugin.h"
void slapi_UTF-8TOLOWER(char *s, char *d, int *ssz, int *dsz);
```

Parameters

This function takes the following parameters:

- s* A single UTF-8 character (could be multiple bytes).
- d* Pointer to the lower case form of *s*. The memory for this must be allocated by the caller before calling the function.
- ssz* Returns the length in bytes of the input character.
- dsz* Returns the length in bytes of the output character.

slapi_UTF-8TOUPPER()

Converts a lower case UTF-8 character to an upper case character.

Syntax

```
#include "slapi-plugin.h"
void slapi_UTF-8TOUPPER(char *s, char *d, int *ssz, int *dsz);
```

Parameters

This function takes the following parameters:

- s* Pointer to a single UTF-8 character (could be multiple bytes).
- d* Pointer to the upper case version of *s*. The memory for this must be allocated by the caller before calling the function.
- ssz* Returns the length in bytes of the input character.
- dsz* Returns the length in bytes of the output character.

slapi_value_compare()

Compares two values for a given attribute to determine if they are equals.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_compare(const Slapi_Attr *a, const Slapi_Value *v1,
    const Slapi_Value *v2);
```

Parameters

This function takes the following parameters:

- a* A pointer to an attribute used to determine how the two values will be compared.
- v1* Pointer to the “[Slapi_ValueSet](#)” on page 270 structure containing the first value to compare.
- v2* Pointer to the “[Slapi_Value](#)” on page 270 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*.

Description

This function compares two `Slapi_Value`s using the matching rule associated to the attribute *a*.

This function replaces the deprecated `slapi_attr_value_cmp()` function used in previous releases, and uses the “[Slapi_Value](#)” on page 270 attribute values instead of the “[berval](#)” on page 249 attribute values.

`slapi_value_done()`

Frees internals of a `Slapi_Value` structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_value_done(Slapi_Value *v);
```

Parameters

This function takes the following parameter:

v Pointer to the “[Slapi_Value](#)” on page 270 structure whose internals you want to free.

See Also

“[slapi_value_init\(\)](#)” on page 544, “[slapi_value_init_berval\(\)](#)” on page 545,
“[slapi_value_init_string\(\)](#)” on page 546

`slapi_value_dup()`

Duplicates a value.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_dup(const Slapi_Value *v);
```

Parameters

This function takes the following parameter:

v Pointer to the “[Slapi_Value](#)” on page 270 structure you wish to duplicate.

Returns

This function returns a pointer to a newly allocated “[Slapi_Value](#)” on page 270.

Memory Concerns

The new “[Slapi_Value](#)” on page 270 is allocated and needs to be freed by the caller, using “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_free\(\)](#)” on page 538

slapi_value_free()

Frees the specified “[Slapi_Value](#)” on page 270 structure and its members from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_value_free(Slapi_Value **value);
```

Parameters

This function takes the following parameter:

value Address of the pointer to the “[Slapi_Value](#)” on page 270 you wish to free.

Description

This function frees the “[Slapi_Value](#)” on page 270 structure and its members (if it is not NULL), and sets the pointer to NULL.

Memory Concerns

Call this function when you are finished working with the structure.

`slapi_valuearray_free()`

Frees the specified array of “[Slapi_Value](#)” on page 270 structures and their members from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_valuearray_free(Slapi_Value ***value);
```

Parameters

This function takes the following parameter:

value Array of “[Slapi_Value](#)” on page 270 structures to free.

Description

This function frees each “[Slapi_Value](#)” on page 270 structure and its members, and sets the pointer to NULL.

Memory Concerns

Call this function when you are finished working with the array of values.

`slapi_value_get_berval()`

Gets the “[berval](#)” on page 249 structure of the value.

Syntax

```
#include "slapi-plugin.h"
const struct berval * slapi_value_get_berval(
    const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

value Pointer to the “[Slapi_Value](#)” on page 270 of which you wish to get the “[berval](#)” on page 249.

Returns

Returns a pointer to the “[berval](#)” on page 249 structure contained in the “[Slapi_Value](#)” on page 270. This function returns a pointer to the actual “[berval](#)” on page 249 structure, and not a copy of it.

Memory Concerns

You should not free the “[berval](#)” on page 249 structure unless you plan to replace it by calling “[slapi_value_set_berval\(\)](#)” on page 553.

See Also

“[slapi_value_set_berval\(\)](#)” on page 553

slapi_value_get_int()

Converts the 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:

value Pointer to the “[Slapi_Value](#)” on page 270 of which you wish to get as an integer.

Returns

This function returns an integer that corresponds to the value stored in the “[Slapi_Value](#)” on page 270 structure, or 0 if there is no value.

Description

Converts the value in the “[Slapi_Value](#)” on page 270 to an integer.

See Also

[“slapi_value_get_long\(\)” on page 541](#)

[“slapi_value_get_uint\(\)” on page 543](#)

slapi_value_get_length()

Gets the actual length of the value.

Syntax

```
#include "slapi-plugin.h"
size_t slapi_value_get_length(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

value Pointer to the [“Slapi_Value” on page 270](#) of which you wish to get the length.

Returns

This function returns the length of the value contained in [“Slapi_Value” on page 270](#), or 0 if there is no value.

Description

This function returns the actual length of a value contained in the [“Slapi_Value” on page 270](#) structure.

slapi_value_get_long()

Converts the value 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:

value Pointer to the “[Slapi_Value](#)” on page 270 of which you wish to get as a long integer.

Returns

This function returns a long integer that corresponds to the value stored in the “[Slapi_Value](#)” on page 270 structure, or 0 if there is no value.

Description

This function converts the value contained in the “[Slapi_Value](#)” on page 270 structure into a long integer.

See Also

“[slapi_value_get_int\(\)](#)” on page 540

“[slapi_value_get_uint\(\)](#)” on page 543

“[slapi_value_get_ulong\(\)](#)” on page 544

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:

value Pointer to the “[Slapi_Value](#)” on page 270 of which you wish to get as a string.

Returns

This function returns a string containing the value, or NULL if there is no value.

This function returns a pointer to the actual string value in “[Slapi_Value](#)” on page 270, not a copy of it.

Memory Concerns

You should not free the string unless to plan to replace it by calling `“slapi_value_set_string()”` on page 555.

See Also

`“slapi_value_set_string()”` on page 555

slapi_value_get_uint()

Converts the value to 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:

value Pointer to the `“Slapi_Value”` on page 270 of which you wish to get as an unsigned integer.

Returns

This function returns an unsigned integer that corresponds to the value stored in the `“Slapi_Value”` on page 270 structure, or 0 if there is no value.

Description

Converts the value contained in `“Slapi_Value”` on page 270 into an unsigned integer.

See Also

`“slapi_value_get_int()”` on page 540

`“slapi_value_get_long()”` on page 541

`“slapi_value_get_ulong()”` on page 544

slapi_value_get_ulong()

Converts the value into an unsigned long.

Syntax

```
#include "slapi-plugin.h"
unsigned long slapi_value_get_ulong(const Slapi_Value *value);
```

Parameters

This function takes the following parameter:

value Pointer to the “[Slapi_Value](#)” on page 270 of which you wish to get as an unsigned long integer.

Returns

This function returns an unsigned long integer that corresponds to the value stored in the “[Slapi_Value](#)” on page 270 structure, or 0 if there is no value.

Description

Converts the value contained in the “[Slapi_Value](#)” on page 270 structure into an unsigned long integer.

See Also

“[slapi_value_get_int\(\)](#)” on page 540

“[slapi_value_get_long\(\)](#)” on page 541

“[slapi_value_get_uint\(\)](#)” on page 543

slapi_value_init()

Initializes a “[Slapi_Value](#)” on page 270 structure with no value.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_init(Slapi_Value *v);
```


Parameters

This function takes the following parameter:

v Pointer to the value to be initialized. The pointer must not be NULL.

Returns

This function returns a pointer to the initialized “[Slapi_Value](#)” on page 270 structure (itself).

Description

This function initializes the “[Slapi_Value](#)” on page 270 structure, resetting all of its fields to zero. The value passed as the parameter must be a valid “[Slapi_Value](#)” on page 270.

Memory Concerns

When finished using the “[Slapi_Value](#)” on page 270 structure, free its internal structures by using “[slapi_value_done\(\)](#)” on page 537.

slapi_value_init_berval()

Initializes a “[Slapi_Value](#)” on page 270 structure from the “[berval](#)” on page 249 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_init_berval(Slapi_Value *v,
    struct berval *bval);
```

Parameters

This function takes the following parameters:

v Pointer to the value to initialize. The pointer must not be NULL.

bval Pointer to the “[berval](#)” on page 249 structure to be used to initialize the value.

Returns

This function returns a pointer to the initialized “[Slapi_Value](#)” on page 270 structure (itself).

Description

This function initializes the “[Slapi_Value](#)” on page 270 structure with the value contained in the “[berval](#)” on page 249 structure. The content of the “[berval](#)” on page 249 structure is duplicated.

Memory Concerns

When finished using the “[Slapi_Value](#)” on page 270 structure, free its internal structures by using “[slapi_value_done\(\)](#)” on page 537.

slapi_value_init_string()

Initializes a “[Slapi_Value](#)” on page 270 structure from a string.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_init_string(Slapi_Value *v, const char *s);
```

Parameters

This function takes the following parameters:

- v* Pointer to the value to be initialized. The pointer must not be NULL.
- s* NULL terminated string used to initialize the value.

Returns

This function returns a pointer to the initialized “[Slapi_Value](#)” on page 270 structure (itself).

Description

This function initializes the “[Slapi_Value](#)” on page 270 structure with the value contained in the string. The string is duplicated.

Memory Concerns

When finished using the “[Slapi_Value](#)” on page 270 structure, free its internal structures by using “[slapi_value_done\(\)](#)” on page 537.

slapi_value_init_string_passin()

Initializes a “[Slapi_Value](#)” on page 270 structure with value contained in the string.

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 be initialized. The pointer must not be NULL.
- s* NULL terminated string used to initialize the value.

Returns

This function returns a pointer to the initialized “[Slapi_Value](#)” on page 270 structure (itself).

Description

This function initializes a “[Slapi_Value](#)” on page 270 structure with the value contained in the string. The string is not duplicated and must be freed.

Memory Concerns

The string will be freed when the “[Slapi_Value](#)” on page 270 structure is freed from memory by calling “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_free\(\)](#)” on page 538

“[slapi_value_new_string_passin\(\)](#)” on page 550

“[slapi_value_set_string_passin\(\)](#)” on page 556

slapi_value_new()

Allocates a new “[Slapi_Value](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new();
```

Parameters

This function does not take any parameters.

Returns

This function returns a pointer to the newly allocated “[Slapi_Value](#)” on page 270 structure. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Description

This function returns an empty “[Slapi_Value](#)” on page 270 structure. You can call other functions of the API to set the value.

Memory Concerns

When you are no longer using the value, free it from memory by calling “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_dup\(\)](#)” on page 537

“[slapi_value_free\(\)](#)” on page 538

“[slapi_value_new_berval\(\)](#)” on page 548

`slapi_value_new_berval()`

Allocates a new “[Slapi_Value](#)” on page 270 structure and initializes it from a “[berval](#)” on page 249 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new_berval(const struct berval *bval);
```

Parameters

This function takes the following parameter:

bval Pointer to the “[berval](#)” on page 249 structure used to initialize the newly allocated “[Slapi_Value](#)” on page 270.

Returns

This function returns a pointer to the newly allocated “[Slapi_Value](#)” on page 270. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program will terminate.

Description

This function returns a “[Slapi_Value](#)” on page 270 structure containing a value duplicated from the “[berval](#)” on page 249 structure passed as the parameter.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_dup\(\)](#)” on page 537

“[slapi_value_free\(\)](#)” on page 538

“[slapi_value_new\(\)](#)” on page 547

“[slapi_value_new_string\(\)](#)” on page 549

`slapi_value_new_string()`

Allocates a new “[Slapi_Value](#)” on page 270 structure and initializes it from a string.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new_string(const char *s);
```

Parameters

This function takes the following parameter:

`s` NULL terminated string used to initialize the newly allocated “[Slapi_Value](#)” on page 270.

Returns

This function returns a pointer to the newly allocated “[Slapi_Value](#)” on page 270. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program will terminate.

Description

This function returns a “[Slapi_Value](#)” on page 270 structure containing a value duplicated from the string passed as the parameter.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_dup\(\)](#)” on page 537

“[slapi_value_free\(\)](#)” on page 538

“[slapi_value_new\(\)](#)” on page 547

“[slapi_value_new_berval\(\)](#)” on page 548

`slapi_value_new_string_passin()`

Allocates a new “[Slapi_Value](#)” on page 270 structure and initializes it from a string.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new_string_passin ( char *s );
```

Parameters

This function takes the following parameter:

`s` NULL terminated string used to initialize the newly allocated “[Slapi_Value](#)” on page 270 structure.

Returns

This function returns a pointer to a newly allocated “[Slapi_Value](#)” on page 270 structure. If space cannot be allocated (for example, if no virtual memory exists), the `slapd` program terminates.

Description

This function returns a “[Slapi_Value](#)” on page 270 structure containing the string passed as the parameter. The string passed in must not be freed from memory.

Memory Concerns

The value should be freed by the caller, using “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_dup\(\)](#)” on page 537

“[slapi_value_free\(\)](#)” on page 538

“[slapi_value_new\(\)](#)” on page 547

`slapi_value_new_value()`

Allocates a new “[Slapi_Value](#)” on page 270 structure and initializes it from another “[Slapi_Value](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_new_value(const Slapi_Value *v);
```

Parameters

This function takes the following parameter:

- `v` Pointer to the “[Slapi_Value](#)” on page 270 structure used to initialize the newly allocated “[Slapi_Value](#)” on page 270.

Returns

This function returns a pointer to the newly allocated “[Slapi_Value](#)” on page 270. If space cannot be allocated (for example, if no more virtual memory exists), the `slapd` program will be terminated.

Description

This function returns a “[Slapi_Value](#)” on page 270 structure containing a value duplicated from the “[Slapi_Value](#)” on page 270 structure passed as the parameter. This function is identical to “[slapi_value_dup\(\)](#)” on page 537.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling the “[slapi_value_free\(\)](#)” on page 538 function/

See Also

“[slapi_value_dup\(\)](#)” on page 537

“[slapi_value_free\(\)](#)” on page 538

slapi_value_set()

Sets the value in a “[Slapi_Value](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_set(Slapi_Value *value, void *val,
    unsigned long len);
```

Parameters

This function takes the following parameters:

value Pointer to the “[Slapi_Value](#)” on page 270 in which to set the value.

val Pointer to the value.

len Length of the value.

Returns

This function returns a pointer to the “[Slapi_Value](#)” on page 270 with the value set.

Description

This function sets the value in the “[Slapi_Value](#)” on page 270 structure. The value is a duplicate of the data pointed to by *val* and of length *len*.

Memory Concerns

If the pointer to the “[Slapi_Value](#)” on page 270 structure is NULL, then nothing is done and the function returns NULL. If the “[Slapi_Value](#)” on page 270 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](#)” on page 270 structure, you should free it from memory by calling “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_free\(\)](#)” on page 538

slapi_value_set_berval()

Copies the value from a “[berval](#)” on page 249 structure into a “[Slapi_Value](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_set_berval(Slapi_Value *value,
    const struct berval *bval );
```

Parameters

This function takes the following parameters:

value Pointer to the “[Slapi_Value](#)” on page 270 structure in which to set the value.

bval Pointer to the “[berval](#)” on page 249 value to be copied.

Returns

This function returns the pointer to the “[Slapi_Value](#)” on page 270 structure passed as the parameter, or NULL if it was NULL.

Description

This function sets the value of “[Slapi_Value](#)” on page 270 structure. The value is duplicated from the “[berval](#)” on page 249 structure *bval*.

Memory Concerns

If the pointer to the “[Slapi_Value](#)” on page 270 structure is NULL, nothing is done and the function returns NULL. If the “[Slapi_Value](#)” on page 270 already contains a value, it is freed from memory before the new one is set.

When you are no longer using the “[Slapi_Value](#)” on page 270 structure, you should free it from memory by calling “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_free\(\)](#)” on page 538

slapi_value_set_int()

Sets the integer value of a “[Slapi_Value](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_set_int(Slapi_Value *value, int intVal);
```

Parameters

This function takes the following parameters:

value Pointer to the “[Slapi_Value](#)” on page 270 structure in which to set the integer value.
intVal 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](#)” on page 270 is NULL.

Description

This function sets the value of the “[Slapi_Value](#)” on page 270 structure from the integer *intVal*.

Memory Concerns

If the pointer to the “[Slapi_Value](#)” on page 270 structure is NULL, nothing is done and the function returns -1. If the “[Slapi_Value](#)” on page 270 already contains a value, it is freed from memory before the new one is set.

When you are no longer using the “[Slapi_Value](#)” on page 270 structure, you should free it from memory by calling “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_free\(\)](#)” on page 538

slapi_value_set_string()

Copies a string in the value of a “[Slapi_Value](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_set_string(Slapi_Value *value, const char *strVal);
```

Parameters

This function takes the following parameters:

value Pointer to the “[Slapi_Value](#)” on page 270 structure in which to set the value.
strVal 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](#)” on page 270 is NULL.

Description

This function sets the value of the “[Slapi_Value](#)” on page 270 structure by duplicating the string *strVal*.

Memory Concerns

If the pointer to the “[Slapi_Value](#)” on page 270 is NULL, nothing is done and the function returns -1. If the “[Slapi_Value](#)” on page 270 already contains a value, it is freed from memory before the new one is set.

When you are no longer using the “[Slapi_Value](#)” on page 270 structure, you should free it from memory by calling “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_free\(\)](#)” on page 538

slapi_value_set_string_passin()

Sets the value of a “[Slapi_Value](#)” on page 270 structure from a string.

Syntax

```
#include "slapi-plugin.h"
int slapi_value_set_string_passin(Slapi_Value *value, char *strVal);
```

Parameters

This function takes the following parameters:

value Pointer to the “[Slapi_Value](#)” on page 270 structure in which to set the value.

strVal The string containing the value to set.

Returns

This function returns 0 if the value is set, or -1 if the pointer to the “[Slapi_Value](#)” on page 270 structure is NULL.

Description

This function sets the value of “[Slapi_Value](#)” on page 270 structure with the string *strVal*. If the “[Slapi_Value](#)” on page 270 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.

Memory Concerns

Use “[slapi_value_free\(\)](#)” on page 538 when you are finished working with the structure to free it from memory.

slapi_value_set_value()

Copies the value of a “[Slapi_Value](#)” on page 270 structure into a “[Slapi_Value](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
Slapi_Value * slapi_value_set_value(Slapi_Value *value,
    const Slapi_Value *vfrom);
```

Parameters

This function takes the following parameters:

value Pointer to the “[Slapi_Value](#)” on page 270 in which to set the value.

vfrom Pointer to the “[Slapi_Value](#)” on page 270 from which to get the value.

Returns

This function returns the pointer to the “[Slapi_Value](#)” on page 270 structure passed as the parameter, or NULL if it was NULL.

Description

This function sets the value of the “[Slapi_Value](#)” on page 270 structure. This value is duplicated from the “[Slapi_Value](#)” on page 270 structure *vfrom*. *vfrom* must not be NULL.

Memory Concerns

If the pointer to the “[Slapi_Value](#)” on page 270 is NULL, nothing is done and the function returns NULL. If the “[Slapi_Value](#)” on page 270 already contains a value, it is freed from before the new one is set.

When you are no longer using the “[Slapi_Value](#)” on page 270 structure, you should free it from memory by calling “[slapi_value_free\(\)](#)” on page 538.

See Also

“[slapi_value_free\(\)](#)” on page 538

slapi_valueset_add_value_optimised()

Adds a “[Slapi_Value](#)” on page 270 in the “[Slapi_ValueSet](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_add_value_optimised(Slapi_ValueSet *vs,
    const Slapi_Value *addval, void *syntax_plugin);
```

Parameters

This function takes the following parameters:

<i>vs</i>	Pointer to the “ Slapi_ValueSet ” on page 270 structure to which to add the value.
<i>addval</i>	Pointer to the “ Slapi_Value ” on page 270 to add to the “ Slapi_ValueSet ” on page 270.
<i>syntax_plugin</i>	Pointer to the plug-in for this attribute type, obtained using “ slapi_attr_get_plugin() ” on page 308.

Description

This function adds a value in the form of a “[Slapi_Value](#)” on page 270 structure in a “[Slapi_ValueSet](#)” on page 270 structure.

Memory Concerns

The value is duplicated from the “[Slapi_Value](#)” on page 270 structure, which can be freed from memory after using it without altering the “[Slapi_ValueSet](#)” on page 270 structure.

This function does not verify if the value is already present in the “[Slapi_ValueSet](#)” on page 270 structure. You can manually check this using “[slapi_valueset_first_value_const\(\)](#)” on page 560 and “[slapi_valueset_next_value_const\(\)](#)” on page 563.

See Also

“[slapi_valueset_first_value_const\(\)](#)” on page 560

“[slapi_valueset_next_value_const\(\)](#)” on page 563

`slapi_valueset_count()`

Returns the number of values contained in a “[Slapi_ValueSet](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_valueset_count(const Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameter:

vs Pointer to the “[Slapi_ValueSet](#)” on page 270 structure of which you wish to get the count.

Returns

This function returns the number of values contained in the “[Slapi_ValueSet](#)” on page 270 structure.

slapi_valueset_done()

Frees the values contained in the “[Slapi_ValueSet](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_done(Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameter:

vs Pointer to the “[Slapi_ValueSet](#)” on page 270 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 reuse the “[Slapi_ValueSet](#)” on page 270 structure for a new set of values.

slapi_valueset_find_const()

Finds the value in a value set using the syntax of an attribute.

```
#include "slapi-plugin.h"
const Slapi_Value *slapi_valueset_find(const Slapi_Attr *a,
```

```
const Slapi_ValueSet *vs, const Slapi_Value *v);
```

Parameters

This function takes the following parameters:

- a* Pointer to the attribute. This is used to determine the syntax of the values and how to match them.
- vs* Pointer to the “[Slapi_ValueSet](#)” on page 270 structure from which you wish to get the value.
- v* Address of the pointer to the “[Slapi_Value](#)” on page 270 structure for the returned value.

Returns

This function returns a pointer to the value in the value set if the value was found. Otherwise, it returns NULL.

Description

Use this function to check for duplicate values in an attribute.

slapi_valueset_first_value_const()

Gets the first value of a “[Slapi_ValueSet](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_valueset_first_value_const(const Slapi_ValueSet *vs,
    const Slapi_Value **v);
```

Parameters

This function takes the following parameters:

- vs* Pointer to the “[Slapi_ValueSet](#)” on page 270 structure from which you wish to get the value.
- v* Address of the pointer to the “[Slapi_Value](#)” on page 270 structure for the returned value.

Returns

This function returns the index of the value in the “[Slapi_ValueSet](#)” on page 270, or -1 if there was no value.

Description

Call this function when you wish to get the first value of a “[Slapi_ValueSet](#)” on page 270, or you wish to iterate through all of the values. The returned value is the index of the value in the “[Slapi_ValueSet](#)” on page 270 structure and must be passed to call “[slapi_valueset_next_value_const\(\)](#)” on page 563 to get the next value.

Memory Concerns

This function gives a pointer to the actual value within the “[Slapi_ValueSet](#)” on page 270. You should not free it from memory.

See Also

“[slapi_valueset_next_value_const\(\)](#)” on page 563

slapi_valueset_free()

Frees the specified “[Slapi_ValueSet](#)” on page 270 structure and its members from memory.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_free(Slapi_ValueSet *vs)
```

Parameters

This function takes the following parameter:

vs Pointer to the “[Slapi_ValueSet](#)” on page 270 to free.

Description

This function frees the “[Slapi_ValueSet](#)” on page 270 structure and its members if it is not NULL. Call this function when you are finished working with the structure.

See Also

“[slapi_valueset_done\(\)](#)” on page 559

slapi_valueset_init()

Resets a “[Slapi_ValueSet](#)” on page 270 structure to no values.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_init(Slapi_ValueSet *vs);
```

Parameters

This function takes the following parameter:

vs Pointer to the “[Slapi_ValueSet](#)” on page 270 to reset.

Description

This function initializes the values contained in the “[Slapi_ValueSet](#)” on page 270 structure (sets them to 0). This does not free the values contained in the structure. To free the values, use “[slapi_valueset_done\(\)](#)” on page 559.

Memory Concerns

When you are no longer using the “[Slapi_ValueSet](#)” on page 270 structure, you should free it from memory by using “[slapi_valueset_free\(\)](#)” on page 561.

See Also

“[slapi_valueset_done\(\)](#)” on page 559

“[slapi_valueset_free\(\)](#)” on page 561

slapi_valueset_new()

Allocates a new “[Slapi_ValueSet](#)” on page 270 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](#)” on page 270 structure. If no space could be allocated (for example, if no more virtual memory exists), the `slapd` program terminates.

Description

This function returns an empty “[Slapi_ValueSet](#)” on page 270 structure. You can call other `slapi_valueset` functions of the API to set the values in the “[Slapi_ValueSet](#)” on page 270 structure.

Memory Concerns

When you are no longer using the value, you should free it from memory by calling “[slapi_valueset_free\(\)](#)” on page 561.

See Also

“[slapi_valueset_free\(\)](#)” on page 561

`slapi_valueset_next_value_const()`

Gets the next value from a “[Slapi_ValueSet](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
int slapi_valueset_next_value_const(const Slapi_ValueSet *vs, int index,
    const Slapi_Value **v);
```

Parameters

This function takes the following parameters:

- vs* Pointer to the “[Slapi_ValueSet](#)” on page 270 structure from which you wish to get the value.
- index* Value returned by the previous call to `slapi_valueset_next_value_const()` or “[slapi_valueset_first_value_const\(\)](#)” on page 560.
- v* Address to the pointer to the “[Slapi_Value](#)” on page 270 structure for the returned value.

Returns

This function returns the index of the value in the “[Slapi_ValueSet](#)” on page 270, or -1 if there was no more value or the input index is incorrect.

Description

Call this function when you wish to get the next value of a “[Slapi_ValueSet](#)” on page 270, after having first called “[slapi_valueset_first_value_const\(\)](#)” on page 560. The returned value is the index of the value in the “[Slapi_ValueSet](#)” on page 270 structure and must be passed to `slapi_valueset_next_value_const()`.

Memory Concerns

This function gives a pointer to the actual value within the “[Slapi_ValueSet](#)” on page 270 and you should not free it from memory.

See Also

“[slapi_valueset_first_value_const\(\)](#)” on page 560

`slapi_valueset_set_from_smod()`

Copies the values of “[Slapi_Mod](#)” on page 265 structure into a “[Slapi_ValueSet](#)” on page 270 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 “ Slapi_ValueSet ” on page 270 structure into which you wish to copy the values. |
| <i>smod</i> | Pointer to the “ Slapi_Mod ” on page 265 structure from which you wish to copy the values. |

Description

This function copies all of the values contained in a “[Slapi_Mod](#)” on page 265 structure into a “[Slapi_ValueSet](#)” on page 270 structure.

Memory Concerns

This function does not verify that the “[Slapi_ValueSet](#)” on page 270 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\(\)](#)” on page 559.

See Also

“[slapi_valueset_done\(\)](#)” on page 559

slapi_valueset_set_valueset_optimised()

Initializes a “[Slapi_ValueSet](#)” on page 270 structure from another “[Slapi_ValueSet](#)” on page 270 structure.

Syntax

```
#include "slapi-plugin.h"
void slapi_valueset_set_valueset_optimised(Slapi_ValueSet *vs1,
    const Slapi_ValueSet *vs2, void *syntax_plugin);
```

Parameters

This function takes the following parameters:

<i>vs1</i>	Pointer to the “ Slapi_ValueSet ” on page 270 structure to which you wish to set the values.
<i>vs2</i>	Pointer to the “ Slapi_ValueSet ” on page 270 structure from which you wish to copy the values.
<i>syntax_plugin</i>	Pointer to the plug-in for this attribute type, obtained using <code>slapi_attr_type2plugin()</code> .

Description

This function initializes a “[Slapi_ValueSet](#)” on page 270 structure by copying the values contained in another “[Slapi_ValueSet](#)” on page 270 structure.

Memory Concerns

The function does not verify that the “[Slapi_ValueSet](#)” on [page 270](#) 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\(\)](#)” on [page 559](#).

See Also

“[slapi_valueset_done\(\)](#)” on [page 559](#)

slapi_vattr_attr_free()

Free a virtual attribute.

Syntax

```
#include "slapi-plugin.h"
void slapi_vattr_attr_free(Slapi_Attr **a, int buffer_flags);
```

Parameters

This function takes the following parameters:

<i>a</i>	Attribute to free
<i>buffer_flags</i>	Bitmask of SLAPI_VIRTUALATTRS_RETURNED_POINTERS, SLAPI_VIRTUALATTRS_RETURNED_COPIES, SLAPI_VIRTUALATTRS_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_RETURNED_TYPENAME_ONLY

Description

Use this function to free a virtual attribute when finished with it.

See Also

“[slapi_vattr_attrs_free\(\)](#)” on [page 566](#)

slapi_vattr_attrs_free()

Free a list of virtual attributes.

Syntax

```
#include "slapi-plugin.h"
void slapi_vattr_attrs_free(vattr_type_thang **types, int flags);
```

Parameters

This function takes the following parameters:

types List of attributes to free

flags Bitmask of SLAPI_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_ONLY, SLAPI_VIRTUALATTRS_REQUEST_POINTERS, SLAPI_VIRTUALATTRS_LIST_OPERATIONAL_ATTRS passed as flags to [“slapi_vattr_list_attrs\(\)” on page 569](#)

Description

Use this function to free a list of virtual attributes obtained using [“slapi_vattr_list_attrs\(\)” on page 569](#).

See Also

[“slapi_vattr_attr_free\(\)” on page 566](#)

[“slapi_vattr_list_attrs\(\)” on page 569](#)

slapi_vattr_filter_test()

Test a filter against an entry that may contain virtual attributes.

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:

pb Parameter block containing the search request

e Candidate entry

f Filter to check

verify_access 1 to verify access to the entry before checking, 0 otherwise

Description

This function checks whether the candidate entry *e* matches the filter *f*. It does not support LDAP v3 extensible match filters.

Returns

This functions returns 0 if the filter matches, or if the filter is NULL. Otherwise, it returns -1.

slapi_vattr_is_registered()

Check whether an attribute may be virtual.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_is_registered(const char *attrtype,
                             const char *scopendn);
```

Parameters

This function takes the following parameters:

attrtype Attribute type to check
scopendn Base of the scope to check

Description

This function checks whether a virtual attribute service is registered for the attribute type in the scope specified.

The fact that a virtual attribute service is registered for an attribute type does not guarantee that the service can currently provide a value.

Returns

This functions returns 1 if the attribute may be virtual in the scope specified. Otherwise, it returns 0.

slapi_vattr_is_virtual()

Check whether an attribute is virtually generated.

Syntax

```
#include "slapi-plugin.h"
int slapi_vattr_is_virtual( Slapi_Entry *e, const char *attrtype,
                          Slapi_Value *v);
```

Parameters

This function takes the following parameters:

<i>e</i>	Entry to check.
<i>attrtype</i>	Attribute type to check.
<i>v</i>	Not currently used.

Returns

This functions returns 1 if the attribute value is virtually generated. Otherwise, it returns 0.

slapi_vattr_list_attrs()

Get a list of 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>	Get attributes for this entry
<i>types</i>	List of attributes set by the server
<i>flags</i>	Bitmask of SLAPI_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_ONLY, SLAPI_VIRTUALATTRS_REQUEST_POINTERS, SLAPI_VIRTUALATTRS_LIST_OPERATIONAL_ATTRS determining what to set in the list

buffer_flags Bitmask of SLAPI_VIRTUALATTRS_RETURNED_POINTERS, SLAPI_VIRTUALATTRS_RETURNED_COPIES, SLAPI_VIRTUALATTRS_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_RETURNED_TYPENAME_ONLY determining how the virtual attributes should be handled

Description

This function sets types to point to a full list of attributes for the entry *e* depending on the *flags* parameter. Use the *buffer_flags* parameter when freeing the list.

Use “[slapi_vattr_values_type_thang_get\(\)](#)” on page 573 to access the attributes.

Returns

This functions returns 1 if the attribute may be virtual in the scope specified. Otherwise, it returns 0.

Memory Considerations

When finished with the types list, free it using “[slapi_vattr_attrs_free\(\)](#)” on page 566.

See Also

“[vattr_type_thang](#)” on page 270

“[slapi_vattr_attrs_free\(\)](#)” on page 566

“[slapi_vattr_values_type_thang_get\(\)](#)” on page 573

slapi_vattr_value_compare()

Compares attribute type and name in a given entry.

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:

e 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 0 for success, in which case `result` contains the result of the comparison.

Otherwise, this function returns the following:

- `SLAPI_VIRTUALATTRS_LOOP_DETECTED` (failed to evaluate a virtual attribute).
- `SLAPI_VIRTUAL_NOT_FOUND` (type not recognized by any virtual attribute and not a real attr in entry).
- `ENOMEM` (memory error).

Description

There is no need to call “[slapi_vattr_values_free\(\)](#)” on [page 571](#) after calling this function.

slapi_vattr_values_free()

Frees the value set and type names.

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>	Value set 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() ” on page 572 . This contains information that this function needs to determine which objects need to be freed.

Description

This function should be used to free the value set and type names returned from “[slapi_vattr_values_get_ex\(\)](#)” on page 572.

See Also

“[slapi_vattr_values_get_ex\(\)](#)” on page 572

slapi_vattr_values_get_ex()

Returns the values for an attribute type from an entry.

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_VIRTUALATTRS_ONLY▪ SLAPI_VIRTUALATTRS_REQUEST_POINTERS▪ SLAPI_VIRTUALATTRS_LIST_OPERATIONAL_ATTRS
<i>buffer_flags</i>	Bit mask to be used as input flags for “ slapi_vattr_values_free() ” on page 571.
<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 in `e`.
- `type_name_disposition` contains information on how each type was matched. Valid values are:
 - `SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_EXACTLY_OR_ALIAS`
 - `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_vattr_values_free\(\)`” on page 571](#).
- `subtype_count` contains the number of subtypes matched.

Otherwise, this function returns the following

- `SLAPI_VIRTUALATTRS_LOOP_DETECTED` (failed to evaluate a virtual attribute).
- `SLAPI_VIRTUAL_NOT_FOUND` (type not recognized by any virtual attribute and no real attribute in entry).
- `ENOMEM` (memory error).

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.

Memory Concerns

[“`slapi_vattr_values_free\(\)`” on page 571](#) 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\(\)`” on page 571](#)

`slapi_vattr_values_type_thang_get()`

Get values from a list of the real and virtual attributes for an entry.

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 the attributes belong to
<i>type_thang</i>	Real or virtual attribute type
<i>results</i>	Values for the attribute, set by the server
<i>type_name_disposition</i>	Set by the server to reflect how type name matched; one of SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_EXACTLY_OR_ALIAS, SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_SUBTYPE, SLAPI_VIRTUALATTRS_NOT_FOUND (type matched no real or virtual attribute on the entry), or SLAPI_VIRTUALATTRS_LOOP_DETECTED (could not evaluate the virtual attribute)
<i>actual_type_name</i>	Set by the server to the actual type name found
<i>flags</i>	Bitmask of SLAPI_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_ONLY, SLAPI_VIRTUALATTRS_REQUEST_POINTERS, SLAPI_VIRTUALATTRS_LIST_OPERATIONAL_ATTRS applied when obtaining the list using “ slapi_vattr_list_attrs() ” on page 569
<i>buffer_flags</i>	Set by the server to a bitmask of SLAPI_VIRTUALATTRS_RETURNED_POINTERS, SLAPI_VIRTUALATTRS_RETURNED_COPIES, SLAPI_VIRTUALATTRS_REALATTRS_ONLY, SLAPI_VIRTUALATTRS_RETURNED_TYPENAME_ONLY, useful for freeing the list

Description

This function offers optimized access to values of attributes in a list set by “[slapi_vattr_list_attrs\(\)](#)” on [page 569](#).

Returns

This function returns 0 for success, in which case:

- `results` contains the current values for type all of the subtypes in `e`.

- `type_name_disposition` contains information on how each type was matched. Valid values are
 - `SLAPI_VIRTUALATTRS_TYPE_NAME_MATCHED_EXACTLY_OR_ALIAS`
 - `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_vattr_values_free\(\)`” on page 571](#).
 - `subtype_count` contains the number of subtypes matched.
- Otherwise, this function returns the following
- `SLAPI_VIRTUALATTRS_LOOP_DETECTED` (failed to evaluate a virtual attribute).
 - `SLAPI_VIRTUAL_NOT_FOUND` (type not recognized by any virtual attribute and not a real attr in entry).
 - `ENOMEM` (memory error).

See Also

[“`vattr_type_thang`” on page 270](#)

[“`slapi_vattr_list_attrs\(\)`” on page 569](#)

`slapi_wait_condvar()`

Wait for a change in a condition variable.

Syntax

```
#include "slapi-plugin.h"
int slapi_wait_condvar(Slapi_CondVar *cvar, struct timeval *timeout);
```

Parameters

This function takes the following parameter:

<i>cvar</i>	Condition variable on which to wait
<i>timeout</i>	NULL means block until notified. Otherwise, block until the time is up, then try again to acquire the lock.

Description

This function enables thread synchronization using a wait/notify mechanism.

Returns

This function returns 1 if successful. Otherwise, it returns NULL.

Memory Considerations

Call “[slapi_notify_condvar\(\)](#)” on [page 459](#) to notify other threads of a change to the condition variable.

See Also

“[slapi_destroy_condvar\(\)](#)” on [page 338](#)

“[slapi_new_condvar\(\)](#)” on [page 457](#)

“[slapi_notify_condvar\(\)](#)” on [page 459](#)

Parameter Block Reference

This chapter describes the parameters available in the parameter block, “[Slapi_PBlock](#)” on [page 267](#), structure. This chapter lists data types associated with each parameter.

Note – To read parameter values, use “[slapi_pblock_get\(\)](#)” on [page 462](#). To write parameter values, use “[slapi_pblock_set\(\)](#)” on [page 464](#). Use of other functions may crash the server.

Refer to the [Part I](#) and the example plug-ins under *install-path/examples/* to better grasp how to use these parameters.

Parameter Categories

This chapter categorizes parameters as follows.

TABLE 18–1 Slapi_PBlock Parameter Categories

Category	Accessible to...
“Access Log” on page 578	All plug-in functions
“Add” on page 578	Pre- and post-operation add functions
“Backend Information” on page 579	All plug-in functions
“Bind” on page 579	Pre- and post-operation bind functions
“Compare” on page 580	Pre- and post-operation compare functions
“Connection Information” on page 580	All plug-in functions
“Delete” on page 581	Pre- and post-operation delete functions

TABLE 18-1 Slapi_PBlock Parameter Categories <i>(Continued)</i>	
Category	Accessible to...
“Directory Configuration Information” on page 582	All plug-in functions
“Extended Operations” on page 582	Extended operation functions
“Internal Operations” on page 583	All plug-in functions
“Modify” on page 583	Pre- and post-operation modify functions
“Operation Information” on page 583	All plug-in functions
“Plug-In Registration” on page 584	Plug-in initialization functions and specific types as mentioned in this section
“Password Verification” on page 586	Password check functions and data
“Post-Operation Entry Access” on page 586	Post-operation plug-in functions
“Rename (Modify RDN)” on page 593	Pre- and post-operation modify RDN functions
“Results” on page 593	All plug-in functions
“Search” on page 594	Pre- and post-operation search functions

Access Log

The following parameter allows you to configure access log output.

TABLE 18-2 Access Log Parameters

Parameter ID	Data Type	Description
SLAPI_OPERATION_NOTES	unsigned int	Flag specifying that unindexed searches be indicated in the access log. Setting this parameter to SLAPI_OP_NOTE_UNINDEXED causes the string Notes=U to be appended to access log entries reflecting unindexed searches.

Add

The following parameters allow you to access an entry to add to the directory through the parameter block.

TABLE 18–3 Add Function Parameters

Parameter ID	Data Type	Description
SLAPI_ADD_ENTRY	Slapi_Entry *	Entry to add.
SLAPI_ADD_TARGET	char *	DN of the entry to add.

Backend Information

The following parameters allow you to access information about directory backends through the parameter block.

TABLE 18–4 Backend Information Parameters

Parameter ID	Data Type	Description
SLAPI_BACKEND	Slapi_Backend *	Backend serving the current operation. May be NULL if no backend is associated with the current operation.
SLAPI_BE_LASTMOD	int	Whether the backend tracks modification time and who made the modification: <ul style="list-style-type: none"> ■ 0 if modifications are not tracked ■ 1 if modifications are tracked
SLAPI_BE_READONLY	int	Value of <code>nsslapd-readonly</code> in the server configuration file: <ul style="list-style-type: none"> ■ 0 if the backend is writable. ■ 1 if the backend is read-only.
SLAPI_BE_TYPE	char *	Value of <code>nsslapd-database</code> in the server configuration file.
SLAPI_DBSIZE	unsigned int	Size of the backend database in KB.

Bind

The following parameters allow you to access information about the bind operation through the parameter block.

TABLE 18–5 Bind Function Parameters

Parameter ID	Data Type	Description
SLAPI_BIND_CREDENTIALS	struct berval *	Bind request credentials such as a password or SASL mechanism credentials, depending on the bind method.
SLAPI_BIND_METHOD	int	Authentication method used. <ul style="list-style-type: none">■ LDAP_AUTH_NONE (anonymous)■ LDAP_AUTH_SASL (SASL)■ LDAP_AUTH_SIMPLE (password)■ LDAP_AUTH_SSL (certificate)
SLAPI_BIND_RET_SASLCREDS	struct berval *	SASL server credentials to send to the client.
SLAPI_BIND_SASLMECHANISM	char *	SASL mechanism used for bind.
SLAPI_BIND_TARGET	char *	DN used to bind.

Compare

The following parameters allow you to access an entry or attribute to use in a comparison through the parameter block.

TABLE 18–6 Compare Function Parameters

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

Connection Information

The following parameters allow you to access information about the client connection through the parameter block.

TABLE 18–7 Connection Information Parameters

Parameter ID	Data Type	Description
SLAPI_CLIENT_DNS	struct berval *	Fully qualified domain name of the client.
SLAPI_CONN_AUTHMETHOD	char *	Authentication method used. <ul style="list-style-type: none"> ■ SLAPD_AUTH_NONE (anonymous) ■ SLAPD_AUTH_SASL (extensible SASL) ■ SLAPD_AUTH_SIMPLE (password) ■ SLAPD_AUTH_SSL (certificate)
SLAPI_CONN_CLIENTNETADDR	PRNetAddr *	IP address of client.
SLAPI_CONN_DN	char *	DN of the user authenticated for the current connection.
SLAPI_CONNECTION	Slapi_Connection *	The current connection.
SLAPI_CONN_ID	int	Identifier for the current connection.
SLAPI_CONN_IS_REPLICATION_SESSION	int	Whether the connection is for replication. <ul style="list-style-type: none"> ■ 0 false. ■ 1 true.
SLAPI_CONN_IS_SSL_SESSION	int	Whether the connection is over SSL. <ul style="list-style-type: none"> ■ 0 false. ■ 1 true.
SLAPI_CONN_SERVERNETADDR	PRNetAddr *	IP address client is connected to.

Delete

The following parameters allow you to access an entry to delete through the parameter block.

TABLE 18–8 Delete Function Parameters

Parameter ID	Data Type	Description
SLAPI_DELETE_TARGET	char *	DN of the entry to delete.
SLAPI_ORIGINAL_TARGET	char *	Non-normalized DN of the entry to delete.

Directory Configuration Information

The following parameters allow you to access information about configuration of the directory instance through the parameter block.

TABLE 18–9 Directory Configuration Information Parameters

Parameter ID	Data Type	Description
SLAPI_ARGC	int	Number of command-line arguments passed to the server at startup.
SLAPI_ARGV	char **	Array of command-line arguments passed to the server at startup.
SLAPI_CONFIG_DIRECTORY	char *	File system directory containing configuration files for the instance.
SLAPI_CONFIG_FILENAME	char *	Configuration file used, such as <code>dse.ldif</code> .

Extended Operations

The following parameters allow you to access information about an extended operation through the parameter block.

TABLE 18–10 Extended Operation Parameters

Parameter ID	Data Type	Description
SLAPI_EXT_OP_REQ_OID	char *	Object identifier (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 identifier (OID) to return to the client.
SLAPI_EXT_OP_RET_VALUE	struct berval *	Value to send to the client.

Internal Operations

The following parameters allow you to access information about internal operations through the parameter block.

TABLE 18–11 Internal Operation Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_INTOP_RESULT	int	Result code of internal operation.
SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES	Slapi_Entry **	Array of entries found by internal search.
SLAPI_PLUGIN_INTOP_SEARCH_REFERRALS	char **	Array of referrals found by internal search in LDAP URL format.

Modify

The following parameters allow you to access an entry to modify through the parameter block.

TABLE 18–12 Modify Function Parameters

Parameter ID	Data Type	Description
SLAPI_MODIFY_MODS	LDAPMod **	NULL terminated array of “ LDAPMod ” on page 250 structures containing modifications to perform on the target entry.
SLAPI_MODIFY_TARGET	char *	DN of the entry to modify.
SLAPI_ORIGINAL_TARGET	char *	Non-normalized DN of the entry to modify.

Operation Information

The following parameters allow you to access information about the current operation through the parameter block.

TABLE 18–13 Operation Information Parameters

Parameter ID	Data Type	Description
SLAPI_CONTROLS_ARG	LDAPControl **	Array of controls passed before the operation is created.

TABLE 18-13 Operation Information Parameters (Continued)

Parameter ID	Data Type	Description
SLAPI_IS_INTERNAL_OPERATION	int	Whether the operation originated internally, or as the result of a client request. <ul style="list-style-type: none">■ 0 client request.■ 1 internal operation.
SLAPI_IS_REPLICATED_OPERATION	int	Whether the operation is part of replication with another server. <ul style="list-style-type: none">■ 0 false.■ 1 true.
SLAPI_OPERATION	Slapi_Operation *	Operation currently in progress.
SLAPI_OPERATION_ID	int	Identifier for the operation.
SLAPI_OPERATION_MSGID	long	Message identifier for the operation.
SLAPI_OPINITIATED_TIME	time_t	Time when the server began processing the operation.
SLAPI_REQCONTROLS	LDAPControl **	Array of controls specified in the request.
SLAPI_REQUESTOR_DN	char *	DN of the user requesting the operation.
SLAPI_REQUESTOR_ISROOT	int	Whether the bind DN of the user requesting the operation corresponds to the root DN, the value of <code>nsslapd-rootdn</code> on <code>cn=config</code> for the instance. <ul style="list-style-type: none">■ 0 false.■ 1 true.
SLAPI_TARGET_DN	char *	DN to which the operation applies.

Plug-In Registration

The following parameters are for use when registering plug-ins and their functions with the server and when accessing information about plug-in type and identity.

The following table lists information accessible to all types of plug-ins.

TABLE 18–14 Plug-In Information Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN	void *	Internal server representation of the plug-in.
SLAPI_PLUGIN_ARGC	int	Number of arguments in the configuration entry.
SLAPI_PLUGIN_ARGV	char *	Array of arguments in the configuration entry.
SLAPI_PLUGIN_IDENTITY	void *	Plug-in identifier set and then required by the server when handling internal operations. You may cast this to “ Slapi_ComponentId ” on page 260.
SLAPI_PLUGIN_PRIVATE	void *	Private data you pass to your plug-in functions. You define this data structure.
SLAPI_PLUGIN_TYPE	int	Type of plug-in function, corresponding to the value of <code>nsslapd-pluginType</code> in the configuration entry for the plug-in. (Use the plug-in type values given here between parentheses in configuration entries.) <ul style="list-style-type: none"> ■ SLAPI_PLUGIN_EXTENDEDOP (extendedop) ■ SLAPI_PLUGIN_INTERNAL_POSTOPERATION (internalpostoperation) ■ SLAPI_PLUGIN_INTERNAL_PREOPERATION (internalpreoperation) ■ SLAPI_PLUGIN_LDBM_ENTRY_FETCH_STORE (ldbmentryfetchstore) ■ SLAPI_PLUGIN_MATCHINGRULE (matchingrule) ■ SLAPI_PLUGIN_TYPE_OBJECT (object) ■ SLAPI_PLUGIN_PASSWDCHECK (passwordcheck) ■ SLAPI_PLUGIN_POSTOPERATION (postoperation) ■ SLAPI_PLUGIN_PREOPERATION (preoperation) ■ SLAPI_PLUGIN_PWD_STORAGE_SCHEME (pwdstoragescheme) ■ SLAPI_PLUGIN_REVER_PWD_STORAGE_SCHEME (reverpwdstoragescheme)
SLAPI_PLUGIN_VERSION	char *	Plug-in API version supported by the plug-in. <ul style="list-style-type: none"> ■ SLAPI_PLUGIN_CURRENT_VERSION (presently SLAPI_PLUGIN_VERSION_03) ■ SLAPI_PLUGIN_VERSION_01 (3.x and later servers) ■ SLAPI_PLUGIN_VERSION_02 (4.x and later servers) ■ SLAPI_PLUGIN_VERSION_03 (5.x servers)

Password Verification

Parameters for use with password check plug-ins follow.

TABLE 18–15 Password Verification Parameters

Parameter ID	Data Type	Description
SLAPI_PASSWDCHECK_VALS	Slapi_Value **	Array of password values to check.
SLAPI_PLUGIN_PASSWDCHECK_FN	void *	Function called to check password values against quality criteria. For details, see “What a Password Check Plug-In Must Do” on page 234 .

Post-Operation Entry Access

Parameters for use with post-operation plug-ins follow. These parameters allow plug-ins to compare the state of an entry before an operation with the state of an entry after an operation.

TABLE 18–16 Post-Operation Entry Access Parameters

Parameter ID	Data Type	Description
SLAPI_ENTRY_PRE_OP	Slapi_Entry *	Directory entry before the operation.
SLAPI_ENTRY_POST_OP	Slapi_Entry *	Directory entry after the operation.

Startup and Shutdown

Parameters for registering generic plug-in functions follow. These function types may be registered by any plug-in.

TABLE 18–17 Generic Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_CLOSE_FN	void *	Function called at server shutdown.
SLAPI_PLUGIN_START_FN	void *	Function called at server startup. May be called more than once.

Extended Operations

Parameters for registering an `extendedop` plug-in function and object identifier list follow.

TABLE 18–18 Extended Operation Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_EXT_OP_FN	void *	Function called upon request for an LDAP v3 extended operation.
SLAPI_PLUGIN_EXT_OP_OIDLIST	char **	NULL terminated list of object identifiers (OIDs) handled by the plug-in.

Internal Postoperation

Parameters for registering an `internalpostoperation` plug-in functions follow.

TABLE 18–19 Internal Postoperation Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_INTERNAL_POST_ADD_FN	void *	Function called after an internal add operation completes.
SLAPI_PLUGIN_INTERNAL_POST_DELETE_FN	void *	Function called after an internal delete operation completes.
SLAPI_PLUGIN_INTERNAL_POST_MODIFY_FN	void *	Function called after an internal modify operation completes.
SLAPI_PLUGIN_INTERNAL_POST_MODRDN_FN	void *	Function called after an internal rename (modify RDN) operation completes.

Internal Preoperation

Parameters for registering an `internalpreoperation` plug-in functions follow.

TABLE 18–20 Internal Preoperation Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_INTERNAL_PRE_ADD_FN	void *	Function called before an internal add operation is performed.

TABLE 18–20 Internal Preoperation Registration Parameters (Continued)

Parameter ID	Data Type	Description
SLAPI_PLUGIN_INTERNAL_PRE_DELETE_FN	void *	Function called before an internal delete operation is performed.
SLAPI_PLUGIN_INTERNAL_PRE_MODIFY_FN	void *	Function called before an internal modify operation is performed.
SLAPI_PLUGIN_INTERNAL_PRE_MODRDN_FN	void *	Function called before an internal rename (modify RDN) operation is performed.

Entry Storage and Retrieval

Parameters for registering `ldbentryfetchstore` plug-in functions follow.

TABLE 18–21 Entry Store/Fetch Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_ENTRY_FETCH_FUNC	void *	Function called upon retrieval of an entry from the backend.
SLAPI_PLUGIN_ENTRY_STORE_FUNC	void *	Function called before storing an entry through the backend.

Matching Rules

Parameters for registering `matchingrule` plug-in functions and arguments follow.

TABLE 18–22 Matching Rule Function and Argument Registration Parameters

Parameter ID	Data Type	Description
SLAPI_MATCHINGRULE_DESC	-	Used to signify registration of the matching rule description with <code>“slapi_matchingrule_set()”</code> on page 416.
SLAPI_MATCHINGRULE_NAME	-	Used to signify registration of the matching rule name with <code>“slapi_matchingrule_set()”</code> on page 416.

TABLE 18–22 Matching Rule Function and Argument Registration Parameters (Continued)

Parameter ID	Data Type	Description
SLAPI_MATCHINGRULE_OBSOLETE	-	Used to signify the matching rule is obsolete when registering with “slapi_matchingrule_set()” on page 416.
SLAPI_MATCHINGRULE_OID	-	Used to signify registration of the matching rule object identifier with “slapi_matchingrule_set()” on page 416.
SLAPI_MATCHINGRULE_SYNTAX	-	Used to signify registration of the matching rule syntax with “slapi_matchingrule_set()” on page 416.
SLAPI_PLUGIN_DESTROY_FN	void *	Function called to free memory allocated to filter object.
SLAPI_PLUGIN_MR_FILTER_CREATE_FN	void *	Filter factory function.
SLAPI_PLUGIN_MR_FILTER_INDEX_FN	void *	Function called to set the indexer function.
SLAPI_PLUGIN_MR_FILTER_MATCH_FN	void *	Function called to check for a match.
SLAPI_PLUGIN_MR_FILTER_RESET_FN	void *	Function called to reset the match filter.
SLAPI_PLUGIN_MR_FILTER_REUSABLE	unsigned int	Whether the filter is reusable. ■ 0 false. ■ 1 true.
SLAPI_PLUGIN_MR_INDEXER_CREATE_FN	void *	Index factory function.
SLAPI_PLUGIN_MR_INDEX_FN	void *	Function called to index a single entry.
SLAPI_PLUGIN_MR_KEYS	struct berval **	Array of index keys corresponding to the attribute values.
SLAPI_PLUGIN_MR_OID	char *	Object identifier (OID) corresponding to the extensible match rule.

TABLE 18–22 Matching Rule Function and Argument Registration Parameters (Continued)

Parameter ID	Data Type	Description
SLAPI_PLUGIN_MR_QUERY_OPERATOR	int	Type of operator used to check for matches. SLAPI_OP_EQUAL SLAPI_OP_GREATER SLAPI_OP_GREATER_OR_EQUAL SLAPI_OP_LESS SLAPI_OP_LESS_OR_EQUAL SLAPI_OP_SUBSTRING
SLAPI_PLUGIN_MR_TYPE	char *	Matching rule filter type.
SLAPI_PLUGIN_MR_USAGE	unsigned int	Whether to use the rule to index or to sort. SLAPI_PLUGIN_MR_USAGE_INDEX SLAPI_PLUGIN_MR_USAGE_SORT
SLAPI_PLUGIN_MR_VALUE	struct berval *	Attribute value to match.
SLAPI_PLUGIN_MR_VALUES	struct berval **	Array of attribute values to match.
SLAPI_PLUGIN_OBJECT	void *	Filter object for extensible match. You define this data structure to use in a matching rule plug-in.

Postoperation

Parameters for registering postoperation plug-in functions follow.

TABLE 18–23 Postoperation Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_POST_ABANDON_FN	void *	Function called after an operation is abandoned.
SLAPI_PLUGIN_POST_ADD_FN	void *	Function called after an add operation completes.

TABLE 18–23 Postoperation Function Registration Parameters *(Continued)*

Parameter ID	Data Type	Description
SLAPI_PLUGIN_POST_BIND_FN	void *	Function called after a bind operation completes.
SLAPI_PLUGIN_POST_COMPARE_FN	void *	Function called after a compare operation completes.
SLAPI_PLUGIN_POST_DELETE_FN	void *	Function called after a delete operation completes.
SLAPI_PLUGIN_POST_ENTRY_FN	void *	Function called after an entry is sent to the client.
SLAPI_PLUGIN_POST_MODIFY_FN	void *	Function called after a modify operation completes.
SLAPI_PLUGIN_POST_MODRDN_FN	void *	Function called after a rename (modify RDN) operation completes.
SLAPI_PLUGIN_POST_REFERRAL_FN	void *	Function called after a referral is sent to the client.
SLAPI_PLUGIN_POST_RESULT_FN	void *	Function called after a result is sent to the client.
SLAPI_PLUGIN_POST_SEARCH_FN	void *	Function called after a search operation completes. For persistent search operations, this function is called after the client interrupts the search.
SLAPI_PLUGIN_POST_UNBIND_FN	void *	Function called after an unbind operation completes.

Preoperation

Parameters for registering preoperation plug-in functions follow.

TABLE 18–24 Preoperation Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_PRE_ABANDON_FN	void *	Function called before an operation is abandoned.
SLAPI_PLUGIN_PRE_ADD_FN	void *	Function called before an add operation is performed.

TABLE 18–24 Preoperation Function Registration Parameters (Continued)

Parameter ID	Data Type	Description
SLAPI_PLUGIN_PRE_BIND_FN	void *	Function called before a bind operation is performed.
SLAPI_PLUGIN_PRE_COMPARE_FN	void *	Function called before a compare operation is performed.
SLAPI_PLUGIN_PRE_DELETE_FN	void *	Function called before an delete operation is performed.
SLAPI_PLUGIN_PRE_ENTRY_FN	void *	Function called before an entry is sent to the client.
SLAPI_PLUGIN_PRE_MODIFY_FN	void *	Function called before a modify operation is performed.
SLAPI_PLUGIN_PRE_MODRDN_FN	void *	Function called before an rename (modify RDN) operation is performed.
SLAPI_PLUGIN_PRE_REFERRAL_FN	void *	Function called before a referral is sent to the client.
SLAPI_PLUGIN_PRE_RESULT_FN	void *	Function called before a result is sent to the client.
SLAPI_PLUGIN_PRE_SEARCH_FN	void *	Function called before a search operation is performed.
SLAPI_PLUGIN_PRE_UNBIND_FN	void *	Function called before an unbind operation is performed.

One-Way and Reversible Password Storage

Parameters for registering pwdstorage and reverpwdstorage plug-in functions follow.

TABLE 18–25 Password Storage Function Registration Parameters

Parameter ID	Data Type	Description
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_CMP_FN	void *	Function called to encode a password for comparison with a stored, encoded password.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DB_PWD	char *	Stored, encoded user password.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DEC_FN	void *	(reverpwdstorage plug-ins only) Function called to decode an encrypted password.

TABLE 18–25 Password Storage Function Registration Parameters (Continued)

Parameter ID	Data Type	Description
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_ENC_FN	void *	Function called to encode a password for storage.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_NAME	char *	Short password storage scheme name used by the server to identify the encoding scheme.
SLAPI_PLUGIN_PWD_STORAGE_SCHEME_USER_PWD	char *	User password in clear text.

Rename (Modify RDN)

The following parameters allow you to access an entry to rename through the parameter block.

TABLE 18–26 Rename (Modify RDN) Function Parameters

Parameter ID	Data Type	Description
SLAPI_MODRDN_DELOLDRN	int	Whether to delete the old Relative Distinguished Name (RDN). <ul style="list-style-type: none"> ■ 0 false. ■ 1 true.
SLAPI_MODRDN_NEWRDN	char *	New RDN to assign to the entry.
SLAPI_MODRDN_NEWSUPERIOR	char *	DN of the new parent of the entry being renamed.
SLAPI_MODRDN_TARGET	char *	DN of the entry to rename.
SLAPI_ORIGINAL_TARGET	char *	Non-normalized DN of the entry to rename.

Results

The following parameters allow you to access results through the parameter block.

TABLE 18–27 Result Parameters

Parameter ID	Data Type	Description
SLAPI_ADD_RESCONTROL	LDAPControl *	Lets you add a control to the set of controls to send to the client. Use “ slapi_pblock_set() ” on page 464 to add a control with this parameter.
SLAPI_RES_CONTROLS	LDAPControl **	Array of controls to send to client. If you use this with “ slapi_pblock_set() ” on page 464 to change the set of controls to send to the client, you must retrieve and free the existing set of controls pointed to by SLAPI_RES_CONTROLS.
SLAPI_RESULT_CODE	int	Result code to send to client.
SLAPI_RESULT_MATCHED	char *	Portion of target DN that matched when sending LDAP_NO_SUCH_OBJECT to the client.
SLAPI_RESULT_TEXT	char *	Message to send to client.

Search

The following parameters allow you to access search parameters through the parameter block.

TABLE 18–28 Search Function Parameters

Parameter ID	Data Type	Description
SLAPI_NENTRIES	int	Number of entries returned by the search.
SLAPI_SEARCH_ATTRS	char **	Array of attribute types to return in the search results. The asterisk, *, can be used to mean all real (non-virtual) attributes.
SLAPI_SEARCH_ATTRSONLY	int	Whether to return both attribute types and attribute values. <ul style="list-style-type: none">■ 0 return both.■ 1 return only types.

TABLE 18–28 Search Function Parameters (Continued)

Parameter ID	Data Type	Description
SLAPI_SEARCH_DEREF	int	Method for handling aliases. <ul style="list-style-type: none"> ■ LDAP_DEREF_ALWAYS ■ LDAP_DEREF_FINDING ■ LDAP_DEREF_NEVER ■ LDAP_DEREF_SEARCHING
SLAPI_SEARCH_FILTER	Slapi_Filter *	Filter to be used for the search.
SLAPI_SEARCH_REFERRALS	struct berval **	Array of URLs to other LDAP servers to which the client is referred.
SLAPI_SEARCH_RESULT_ENTRY	void *	Entry returned while iterating through the result set. You may cast this to “ Slapi_Entry ” on page 263 .
SLAPI_SEARCH_SCOPE	int	Scope of the search. <ul style="list-style-type: none"> ■ LDAP_SCOPE_BASE ■ LDAP_SCOPE_ONELEVEL ■ LDAP_SCOPE_SUBTREE
SLAPI_SEARCH_SIZELIMIT	int	Maximum number of entries to return in the search results.
SLAPI_SEARCH_STRFILTER	char *	String representation of the filter to be used for the search.
SLAPI_SEARCH_TARGET	char *	DN of base entry for the search.
SLAPI_SEARCH_TIMELIMIT	int	Maximum number of seconds to allow for the search.

NameFinder Application

This chapter describes how to deploy and configure NameFinder.

NameFinder is a web application that offers users a convenient, browser-based interface. The application allows users to look up contact and organizational information in a Lightweight Directory Access Protocol (LDAP) directory.

This chapter covers the following topics:

- [“Prerequisite Software” on page 597](#)
- [“Deploying NameFinder” on page 598](#)
- [“Configuring NameFinder to Access Your Directory” on page 599](#)
- [“Customizing NameFinder” on page 602](#)

Prerequisite Software

The NameFinder application depends on the following prerequisite software items:

- Java™ 2 Platform Software Development Kit 1.4
- One of the following web application containers into which you deploy the NameFinder `nfDSRK.war` web application file:
 - Sun Java System Application Server 7
 - Sun Java System Web Server 6 2004Q1 Update 1 Service Pack 2
- An LDAP directory server, such as Directory Server, containing data needed by the NameFinder application

Subsequent versions of the prerequisite software can be used as well. You must install and configure prerequisite software before deploying the NameFinder application. Refer to the *Sun Java Enterprise System 5 Installation Guide for UNIX* or a more recent edition for instructions on installing the prerequisite software.

Deploying NameFinder

You can deploy the NameFinder application on Application Server or on Web Server.

▼ To Deploy on Application Server

Before You Begin Before deploying the NameFinder application, make sure you have installed and configured the software listed in [“Prerequisite Software” on page 597](#).

1 Log in to the Application Server browser-based administration interface.

For example, if you configured the Application Server administration interface to use the default port, go to `http://hostname:4848/`.

2 Select a server instance on which to deploy the NameFinder application.

For example, if you configured Application Server to use the default settings, select `server1`.

3 Upload `nfdSRK.war` as a web application, accepting the default settings.

You can find the `nfdSRK.war` web application archive in the `class` directory where you installed Directory Server Resource Kit, *install-path/dsrk6/class/nfdSRK.war*.

4 Apply changes to the server instance.

5 View the NameFinder application for the first time.

For example, if you configured Application Server to use the default settings, go to `http://hostname:81/nfdSRK/`.

You can now proceed to [“Configuring NameFinder to Access Your Directory” on page 599](#).

▼ To Deploy on Web Server

Before You Begin Before deploying the NameFinder application, make sure you have installed and configured the software listed in [“Prerequisite Software” on page 597](#).

1 Log in to the Web Server browser-based administration interface.

For example, if you configured the Web Server administration interface to use the default port, go to `http://hostname:8888/`.

2 Click Manage next to the Web Server Hostname drop-down menu.

3 Select the Virtual Server Class tab, and then click Manage next to the Virtual Server Class drop-down menu.

4 Click Manage next to the Virtual Server drop-down menu.

This page is where you manage the server instance on which to deploy the NameFinder application.

5 Select the Web Application tab, and then complete the web form for application deployment.

You can find the `nfDSRK.war` web application archive in the `class` directory where you installed the Directory Server Resource Kit, *install-path/dsrk6/class/nfDSRK.war*.

Notice when completing the web form that the installation directory is *install-path/dsrk6/class/*.

Furthermore, you can change the name of the application URI. The default is `/nfDSRK`. Another possible application URI is `/NameFinder`.

6 Apply the changes.**7 Return to the Server Manager page to turn the server off, and then turn the server on again.****8 View the NameFinder application for the first time.**

For example, if you configured Web Server to use the default settings and entered `/nfDSRK` as the application URI, go to `http://hostname/nfDSRK/`.

You can now proceed to [“Configuring NameFinder to Access Your Directory” on page 599](#).

Configuring NameFinder to Access Your Directory

You can configure NameFinder to access your directory on Application Server or on Web Server.

▼ To Configure Access When Using Application Server

After deploying the NameFinder application, Application Server creates a `WEB-INF/` container directory that holds NameFinder files. The location of this directory depends on where you installed the Application Server instance.

You must specify in the `WEB-INF/classes/NameFinder.properties` file how to access the directory that holds the data to retrieve.

1 If necessary, determine the path where you deployed the NameFinder application with the Application Server browser-based interface.

The `WEB-INF/classes/NameFinder.properties` file is located in that directory.

2 Become a user, such as superuser, with access to edit the file.

3 Adjust properties in the `WEB-INF/classes/NameFinder.properties` file to allow the application to access the directory, and then save your changes.

The `NameFinder.properties` file is a Java properties file. Everything in the file is case sensitive. Adjust at least the following lines:

```
NameFinder.ldapBase=baseDN
NameFinder.ldapServers=serverList
NameFinder.ldapVersion=3
NameFinder.ldapPort=ldapPort
NameFinder.ldapUser=bindDN
NameFinder.ldapPasswd=bindPassword
```

- *baseDN* is the base DN for people's entries in your organization, such as `ou=people,dc=example,dc=com`.
- *serverList* is a | separated list of directory servers, such as `directory|backup-directory|ext-directory.example.com`.
- *ldapPort* is the port number on which the servers listen for LDAP requests, by default 389.
- *bindDN* is the DN used to authenticate.

Do not enclose the bind DN in quotes.

- *bindPassword* is the password used to authenticate.

For hints regarding what you can adjust, read the comments in the `WEB-INF/classes/sample.properties` file.

4 In the Application Server browser-based interface, apply changes on the server instance by pressing the Apply Changes button.

After applying changes, you can begin using the NameFinder application to look up contact and organizational information.

5 Verify that the NameFinder application works by searching for a known user, such as yourself, using the browser-based interface.

After you are satisfied that the NameFinder application works, you can choose to customize the application for your organization.

▼ To Configure Access When Using Web Server

After deploying the NameFinder application, Web Server creates a `WEB-INF/` container directory that holds NameFinder files. The location of this directory depends on where you installed the Web Server instance.

You must specify in the `WEB-INF/classes/NameFinder.properties` file how to access the directory that holds the data to retrieve.

- 1 **If necessary, determine the path where you deployed the NameFinder application with the Web Server browser-based interface.**

The `WEB-INF/classes/NameFinder.properties` file is located in that directory.

- 2 **Become a user, such as superuser, with access to edit the file.**

- 3 **Adjust properties in the `WEB-INF/classes/NameFinder.properties` file to allow the application to access the directory, and then save your changes.**

The `NameFinder.properties` file is a Java properties file. Everything in the file is case sensitive. Adjust at least the following lines:

```
NameFinder.ldapBase=baseDN
NameFinder.ldapServers=serverList
NameFinder.ldapVersion=3
NameFinder.ldapPort=ldapPort
NameFinder.ldapUser=bindDN
NameFinder.ldapPasswd=bindPassword
```

- *baseDN* is the base DN for people's entries in your organization, such as `ou=people,dc=example,dc=com`.
- *serverList* is a | separated list of directory servers, such as `directory|backup-directory|ext-directory.example.com`.
- *ldapPort* is the port number on which the servers listen for LDAP requests, by default 389.
- *bindDN* is the DN used to authenticate.
Do not enclose the bind DN in quotes.
- *bindPassword* is the password used to authenticate.

For hints regarding what you can adjust, read the comments in the `WEB-INF/classes/sample.properties` file.

You must restart Web Server for the changes to take effect.

- 4 **Return to the Server Manager page in the Web Server browser-based interface to turn the server off, then on again.**

At this point, you can begin using the NameFinder application to look up contact and organizational information.

- 5 **Verify that the NameFinder application works by searching for a known user, such as yourself, using the browser-based interface.**

After you are satisfied that the NameFinder application works, you can choose to customize the application for your organization.

Customizing NameFinder

This section covers what you can customize in the NameFinder web application, using only the Java properties files provided. Detailed explanations of individual properties can be found in the `WEB-INF/classes/sample.properties` file in the directory where you deployed the application.

Note – NameFinder was designed as a Sun internal web application. The default configuration therefore relies on Sun's LDAP schema and directory information tree (DIT). The schema and DIT probably differ from the schema and DIT in use at your organization.

In addition to customizations within the application, you can also customize searches by using options in the search field. Furthermore, you can customize what attributes to display within the browser-based interface. Refer to the NameFinder online help for details.

Connection Properties

As described in [“Configuring NameFinder to Access Your Directory” on page 599](#), you customize the `WEB-INF/classes/NameFinder.properties` file to allow the application to access your directory.

By convention, connection parameters are included in the first few lines of the Java properties file. You can configure to which host-port combination NameFinder connects. You can also configure whether to use LDAP v2 or v3, and whether to bind as a particular user.

NameFinder connection properties only allow you to configure simple authentication connections to the directory, however. You cannot use connection properties to configure NameFinder to connect using SSL or a SASL mechanism.

Search Attribute Properties

NameFinder lets you configure attributes that define search options, attributes to search, and labels for the values returned. The Java properties definitions for such attributes take the following form:

`NameFinder.attr#=optChar|colChar|attr|label|colLabel`

A decimal number

Do not leave any numbers in the sequence that remain commented out.
NameFinder depends on having the numbers in ascending order without gaps.

optChar An option character for use in searches

For example, P is by default the phone number option. Thus, search for the entry with phone number 1 234 567 8910 by typing **-P "1 234 567 8910"** in the NameFinder search field.

Do not use F as an option character. This character is reserved to allow you to enter LDAP search filters, such as **-F "(telephoneNumber=1 234 567 8910)"**, directly.

This parameter is called `arg1` in `WEB-INF/classes/sample.properties`.

colChar A character for use in defining table columns as a parameter to NameFinder

For example, you can use the default configuration. You use the default by passing `fields=nfeP` as one of the options in the URL to NameFinder for a search that returns multiple entries. NameFinder displays results in a four-column table that has column labels Lastname, Firstname, eMail, and Phone #.

This argument is called `arg2` in `WEB-INF/classes/sample.properties`.

attr The LDAP attribute to search when using *optChar*

This argument is called `arg3` in `WEB-INF/classes/sample.properties`.

label The label to display for the corresponding *attr* value when showing results for a single LDAP entry

This argument is called `arg4` in `WEB-INF/classes/sample.properties`.

colLabel The column label to display in the table header for the corresponding *attr* value when showing results for multiple LDAP entries

This argument is called `arg5` in `WEB-INF/classes/sample.properties`.

You can leave variables blank in search attribute properties definitions.

Other Properties

In addition to connection and search attribute properties, NameFinder allows you to define several other properties in the `WEB-INF/classes/NameFinder.properties` file. These other properties govern the following:

- The default table layout for displaying results when searches return multiple entries
- Lists of LDAP attributes to search when looking up phone numbers and email addresses
- Which LDAP attributes correspond to NameFinder attribute *fields*, which allow NameFinder to be abstract from particular LDAP schema
- Table layouts for displaying team views and tables of a manager's direct reports

Refer to `WEB-INF/classes/sample.properties` for details.

Index

A

- abandon operation, 114
- add operation, 114, 128-133, 163-174
- arguments, passing to plug-ins, 94
- attributes
 - finding, 104
 - modifying the value of, 106
- authentication
 - bypassing, 143-161
 - credentials, 156
 - SASL mechanisms, 143-144, 146, 155-161
 - simple, 143-144, 146-155

B

- bind operation, 114, 116, 144
- build rules, 88

C

- callbacks, 174
- central log directories, 40
- certificate database, default path, 40
- clients
 - sending entries to, 114-116, 140-141
 - sending referrals to, 114-116, 140-141
 - sending result codes to, 114-116, 140-141
- compare operation, 114, 126-128
- configuration entries, 68
- configuration settings, 90-97

converting

- entries to LDIF strings, 104
- LDIF strings to entries, 103-104

D

- default locations, 39-42
- delete operation, 114, 138-139, 163-174
- deprecated features, 57-58, 61-68
- Directory Server plug-in API data structures
 - berval, 249
 - computed_attr_context, 249
 - LDAPControl, 250
 - LDAPMod, 250-252
 - mrFilterMatchFn, 252-253
 - plugin_referral_entry_callback, 253-254
 - plugin_result_callback, 254-255
 - plugin_search_entry_callback, 255-256
 - roles_get_scope_fn_type, 256
 - send_ldap_referral_fn_ptr_t, 256-257
 - send_ldap_result_fn_ptr_t, 257-258
 - send_ldap_search_entry_fn_ptr_t, 258-259
 - Slapi_Attr, 259
 - Slapi_Backend, 259
 - Slapi_ComponentId, 260-261
 - slapi_compute_callback_t, 261
 - slapi_compute_output_t, 262
 - Slapi_CondVar, 262-263
 - Slapi_Connection, 263
 - Slapi_DN, 263
 - Slapi_Entry, 263

Directory Server plug-in API data structures

(Continued)

- slapi_extension_constructor_fnptr, 264
- slapi_extension_destructor_fnptr, 264-265
- Slapi_Filter, 265
- Slapi_MatchingRuleEntry, 265
- Slapi_Mod, 265-266
- Slapi_Mods, 266
- Slapi_Mutex, 266-267
- Slapi_Operation, 267
- Slapi_PBlock, 267-268
- slapi_plugin_init_fnptr, 269
- Slapi_PluginDesc, 268-269
- Slapi_RDN, 269-270
- Slapi_Value, 270
- Slapi_ValueSet, 270
- vattn_type_thang, 270-271

Directory Server plug-in API functions, 273-430, 431-576

- alphabetically, 291, 431-576
- slapi_access_allowed(), 292-294
- slapi_acl_check_mods(), 294-296
- slapi_acl_verify_aci_syntax(), 296
- slapi_add_entry_internal_set_pb(), 297-298
- slapi_add_internal_pb(), 298
- slapi_add_internal_set_pb(), 299-300
- slapi_attr_add_value(), 300
- slapi_attr_basetype(), 301
- slapi_attr_dup(), 301-302
- slapi_attr_first_value_const(), 302-303
- slapi_attr_flag_is_set(), 303-304
- slapi_attr_free(), 304
- slapi_attr_get_bervals_copy(), 305
- slapi_attr_get_flags(), 305-306
- slapi_attr_get_numvalues(), 306-307
- slapi_attr_get_oid_copy(), 307-308
- slapi_attr_get_plugin(), 308
- slapi_attr_get_type(), 308-309
- slapi_attr_get_valueset(), 309-310
- slapi_attr_init(), 310-311
- slapi_attr_new(), 311
- slapi_attr_next_value_const(), 311-312
- slapi_attr_syntax_normalize(), 312-313
- slapi_attr_type_cmp(), 313-314

Directory Server plug-in API functions *(Continued)*

- slapi_attr_types_equivalent(), 314
- slapi_attr_value_cmp(), 315
- slapi_attr_value_find(), 315-316
- slapi_be_exist(), 316-317
- slapi_be_get_name(), 317
- slapi_be_get_readonly(), 317-318
- slapi_be_getsuffix(), 318-319
- slapi_be_gettype(), 319
- slapi_be_is_flag_set(), 319-320
- slapi_be_issuffix(), 320
- slapi_be_logchanges(), 321
- slapi_be_private(), 321-322
- slapi_be_select(), 322
- slapi_be_select_by_instance_name(), 322-323
- slapi_build_control(), 324-325
- slapi_build_control_from_berval(), 325-326
- slapi_ch_array_free(), 326-327
- slapi_ch_bvdup(), 327
- slapi_ch_bvecdup(), 328
- slapi_ch_calloc(), 328-329
- slapi_ch_free(), 329-330
- slapi_ch_free_string(), 330-331
- slapi_ch_malloc(), 331
- slapi_ch_realloc(), 332
- slapi_ch_strdup(), 332-333
- slapi_compute_add_evaluator(), 333-334
- slapi_compute_add_search_rewriter_ex(), 334-335
- slapi_control_present(), 335-336
- slapi_delete_internal_pb(), 336
- slapi_delete_internal_set_pb(), 337-338
- slapi_destroy_condvar(), 338
- slapi_destroy_mutex(), 338-339
- slapi_dn_beparent(), 339
- slapi_dn_ignore_case(), 339-340
- slapi_dn_isbesuffix(), 340
- slapi_dn_isbesuffix_norm(), 341
- slapi_dn_isparent(), 341-342
- slapi_dn_isroot(), 342
- slapi_dn_issuffix(), 342-343
- slapi_dn_normalize(), 343-344
- slapi_dn_normalize_case(), 344
- slapi_dn_normalize_to_end(), 344-345
- slapi_dn_parent(), 345-346

Directory Server plug-in API functions (*Continued*)

- slapi_dn_plus_rdn(), 346
- slapi_dup_control(), 346-347
- slapi_entry_add_rdn_values(), 351
- slapi_entry_add_string(), 352
- slapi_entry_add_value(), 352-353
- slapi_entry_add_values_sv(), 353-354
- slapi_entry_add_valueset(), 354-355
- slapi_entry_alloc(), 355-356
- slapi_entry_attr_delete(), 356
- slapi_entry_attr_find(), 356-357
- slapi_entry_attr_get_charpstr(), 357-358
- slapi_entry_attr_get_int(), 358
- slapi_entry_attr_get_long(), 358-359
- slapi_entry_attr_get_uint(), 359
- slapi_entry_attr_get_ulong(), 359-360
- slapi_entry_attr_hasvalue(), 360
- slapi_entry_attr_merge_sv(), 360-361
- slapi_entry_attr_replace_sv(), 361-362
- slapi_entry_attr_set_charpstr(), 362-363
- slapi_entry_attr_set_int(), 363
- slapi_entry_attr_set_long(), 363-364
- slapi_entry_attr_set_uint(), 364
- slapi_entry_attr_set_ulong(), 364-365
- slapi_entry_delete_string(), 365
- slapi_entry_delete_values_sv(), 365-366
- slapi_entry_dup(), 366-367
- slapi_entry_first_attr(), 367-368
- slapi_entry_free(), 368
- slapi_entry_get_dn(), 369
- slapi_entry_get_dn_const(), 369-370
- slapi_entry_get_ndn(), 370
- slapi_entry_get_sdn(), 370-371
- slapi_entry_get_sdn_const(), 371-372
- slapi_entry_get_uniqueid(), 372
- slapi_entry_has_children(), 372-373
- slapi_entry_init(), 373-374
- slapi_entry_isroot(), 374-375
- slapi_entry_merge_values_sv(), 375
- slapi_entry_next_attr(), 376
- slapi_entry_rdn_values_present(), 376-377
- slapi_entry_schema_check(), 377
- slapi_entry_schema_check_ext(), 378
- slapi_entry_set_dn(), 378-379

Directory Server plug-in API functions (*Continued*)

- slapi_entry_set_sdn(), 379-380
- slapi_entry_size(), 380-381
- slapi_entry_syntax_check(), 381
- slapi_entry_vattr_find(), 381-382
- slapi_entry2mods(), 347-348
- slapi_entry2str(), 348-349
- slapi_entry2str_with_options(), 349-351
- slapi_filter_compare(), 382-383
- slapi_filter_free(), 383
- slapi_filter_get_attribute_type(), 384-385
- slapi_filter_get_ava(), 385-386
- slapi_filter_get_choice(), 386-387
- slapi_filter_get_subfilt(), 387-388
- slapi_filter_get_type(), 388-389
- slapi_filter_join(), 389-390
- slapi_filter_list_first(), 390-391
- slapi_filter_list_next(), 391-392
- slapi_filter_test(), 392-393
- slapi_filter_test_ext(), 393-394
- slapi_filter_test_simple(), 394-395
- slapi_find_matching_paren(), 395
- slapi_free_search_results_internal(), 396
- slapi_get_first_backend(), 397
- slapi_get_next_backend(), 398-399
- slapi_get_object_extension(), 398
- slapi_get_suffix_list(), 399-400
- slapi_get_supported_controls_copy(), 400-401
- slapi_get_supported_extended_ops_copy(), 401-402
- slapi_get_supported_saslmechanisms_copy(), 402
- slapi_has8thBit(), 402-403
- slapi_is_root_suffix(), 403-404
- slapi_is_rootdse(), 403
- slapi_ldap_init(), 404-405
- slapi_ldap_unbind(), 405-406
- slapi_ldapmods_syntax_check(), 406
- slapi_lock_mutex(), 407, 461-462
- slapi_log_info_ex(), 409-411
- slapi_log_warning_ex(), 411-413
- slapi_matchingrule_free(), 413-414
- slapi_matchingrule_get(), 414-415
- slapi_matchingrule_new(), 415
- slapi_matchingrule_register(), 415-416
- slapi_matchingrule_set(), 416-417

Directory Server plug-in API functions (*Continued*)

- slapi_mod_add_value(), 417
- slapi_mod_done(), 418
- slapi_mod_dump(), 418-419
- slapi_mod_free(), 419
- slapi_mod_get_first_value(), 419-420
- slapi_mod_get_ldapmod_byref(), 420-421
- slapi_mod_get_ldapmod_passout(), 421
- slapi_mod_get_next_value(), 421-422
- slapi_mod_get_num_values(), 422
- slapi_mod_get_operation(), 423
- slapi_mod_get_type(), 423-424
- slapi_mod_init(), 424
- slapi_mod_init_byref(), 425
- slapi_mod_init_byval(), 425-426
- slapi_mod_init_passin(), 426-427
- slapi_mod_isvalid(), 427
- slapi_mod_new(), 427-428
- slapi_mod_remove_value(), 428
- slapi_mod_set_operation(), 429
- slapi_mod_set_type(), 429
- slapi_moddn_get_newdn(), 430
- slapi_modify_internal_pb(), 431-432
- slapi_modify_internal_set_pb(), 432-433
- slapi_modrdn_internal_pb(), 433-434
- slapi_mods_add(), 435-436
- slapi_mods_add_ldapmod(), 436
- slapi_mods_add_mod_values(), 437
- slapi_mods_add_modbvps(), 438
- slapi_mods_add_smod(), 439
- slapi_mods_add_string(), 439-440
- slapi_mods_done(), 440-441
- slapi_mods_dump(), 441
- slapi_mods_free(), 442
- slapi_mods_get_first_mod(), 442
- slapi_mods_get_first_smod(), 443
- slapi_mods_get_ldapmods_byref(), 443-444
- slapi_mods_get_ldapmods_passout(), 444-445
- slapi_mods_get_next_mod(), 445-446
- slapi_mods_get_next_smod(), 446
- slapi_mods_get_num_mods(), 447
- slapi_mods_init(), 447-448
- slapi_mods_init_byref(), 448
- slapi_mods_init_passin(), 448-449

Directory Server plug-in API functions (*Continued*)

- slapi_mods_insert_after(), 449-450
- slapi_mods_insert_at(), 450-451
- slapi_mods_insert_before(), 451-452
- slapi_mods_insert_smod_at(), 452
- slapi_mods_iterator_backbone(), 453-454
- slapi_mods_new(), 454
- slapi_mods_remove(), 455
- slapi_mods_remove_at(), 455-456
- slapi_mods2entry(), 434-435
- slapi_mr_filter_index(), 456
- slapi_mr_indexer_create(), 456-457
- slapi_new_condvar(), 457-458
- slapi_new_mutex(), 458-459
- slapi_notify_condvar(), 459
- slapi_op_abandoned(), 460
- slapi_op_get_type(), 460-461
- slapi_pblock_destroy(), 461-462
- slapi_pblock_get(), 462-464
- slapi_pblock_new(), 464
- slapi_pblock_set(), 464-466
- slapi_pw_find_sv(), 466-467
- slapi_pw_find_valueset(), 467-468
- slapi_rdn_add(), 468-469
- slapi_rdn_compare(), 469
- slapi_rdn_contains(), 469-470
- slapi_rdn_contains_attr(), 470-471
- slapi_rdn_done(), 471
- slapi_rdn_free(), 472
- slapi_rdn_get_first(), 472-473
- slapi_rdn_get_index(), 473-474
- slapi_rdn_get_index_attr(), 474-475
- slapi_rdn_get_next(), 475-476
- slapi_rdn_get_num_components(), 476
- slapi_rdn_get_rdn(), 476-477
- slapi_rdn_init(), 477
- slapi_rdn_init_dn(), 477-478
- slapi_rdn_init_rdn(), 478
- slapi_rdn_init_sdn(), 479
- slapi_rdn_isempty(), 479-480
- slapi_rdn_new(), 480
- slapi_rdn_new_dn(), 481
- slapi_rdn_new_rdn(), 481-482
- slapi_rdn_new_sdn(), 482-483

Directory Server plug-in API functions (*Continued*)

slapi_rdn_remove(), 483-484
 slapi_rdn_remove_attr(), 484
 slapi_rdn_remove_index(), 485
 slapi_rdn_set_dn(), 485-486
 slapi_rdn_set_rdn(), 486
 slapi_rdn_set_sdn(), 487
 slapi_rdn_syntax_check(), 487-488
 slapi_register_object_extension(), 488-489
 slapi_register_plugin(), 489-490
 slapi_register_role_get_scope(), 490
 slapi_register_supported_control(), 491-492
 slapi_register_supported_saslmechanism(), 492-493
 slapi_role_check(), 494
 slapi_role_get_scope(), 495
 slapi_sdn_add_rdn(), 495-496
 slapi_sdn_compare(), 496
 slapi_sdn_copy(), 497
 slapi_sdn_done(), 497-498
 slapi_sdn_dup(), 498
 slapi_sdn_free(), 498-499
 slapi_sdn_get_backend_parent(), 499-500
 slapi_sdn_get_dn(), 500
 slapi_sdn_get_ndn(), 500-501
 slapi_sdn_get_ndn_len(), 501-502
 slapi_sdn_get_parent(), 502
 slapi_sdn_get_rdn(), 502-503
 slapi_sdn_get_suffix(), 503
 slapi_sdn_isempty(), 504
 slapi_sdn_isgrandparent(), 504-505
 slapi_sdn_isparent(), 505-506
 slapi_sdn_issuffix(), 506
 slapi_sdn_new(), 506-507
 slapi_sdn_new_dn_byref(), 507-508
 slapi_sdn_new_dn_byval(), 508-509
 slapi_sdn_new_dn_passin(), 509-510
 slapi_sdn_new_ndn_byref(), 510
 slapi_sdn_new_ndn_byval(), 511
 slapi_sdn_scope_test(), 511-512
 slapi_sdn_set_dn_byref(), 512-513
 slapi_sdn_set_dn_byval(), 513-514
 slapi_sdn_set_dn_passin(), 514
 slapi_sdn_set_ndn_byref(), 515
 slapi_sdn_set_ndn_byval(), 515-516

Directory Server plug-in API functions (*Continued*)

slapi_sdn_set_parent(), 516-517
 slapi_sdn_set_rdn(), 517-518
 slapi_search_internal_callback_pb(), 518-519
 slapi_search_internal_get_entry(), 520
 slapi_search_internal_pb(), 521
 slapi_search_internal_set_pb(), 521-523
 slapi_send_ldap_referral(), 523-524
 slapi_send_ldap_result(), 524-526
 slapi_send_ldap_search_entry(), 526-527
 slapi_set_object_extension(), 527-528
 slapi_str2entry(), 528-529
 slapi_str2filter(), 529-530
 slapi_unlock_mutex(), 534
 slapi_UTF-8CASECMP(), 531-532
 slapi_UTF-8ISLOWER(), 533
 slapi_UTF-8ISUPPER(), 533-534
 slapi_UTF-8NCASECMP(), 532-533
 slapi_UTF-8STRTOLOWER(), 534
 slapi_UTF-8STRTOUPPER(), 534-535
 slapi_UTF-8TOLOWER(), 535
 slapi_UTF-8TOUPPER(), 536
 slapi_value_compare(), 536-537
 slapi_value_done(), 537
 slapi_value_dup(), 537-538
 slapi_value_free(), 538-539
 slapi_value_get_berval(), 539-540
 slapi_value_get_int(), 540-541
 slapi_value_get_length(), 541
 slapi_value_get_long(), 541-542
 slapi_value_get_string(), 542-543
 slapi_value_get_uint(), 543
 slapi_value_get_ulong(), 544
 slapi_value_init(), 544-545
 slapi_value_init_berval(), 545-546
 slapi_value_init_string(), 546
 slapi_value_init_string_passin(), 547
 slapi_value_new(), 547-548
 slapi_value_new_berval(), 548-549
 slapi_value_new_string(), 549-550
 slapi_value_new_string_passin(), 550-551
 slapi_value_new_value(), 551-552
 slapi_value_set(), 552-553
 slapi_value_set_berval(), 553-554

Directory Server plug-in API functions (*Continued*)

- `slapi_value_set_int()`, 554-555
- `slapi_value_set_string()`, 555-556
- `slapi_value_set_string_passin()`, 556
- `slapi_value_set_value()`, 557
- `slapi_valuearray_free()`, 539
- `slapi_valueset_add_value_optimised()`, 557-558
- `slapi_valueset_count()`, 558-559
- `slapi_valueset_done()`, 559
- `slapi_valueset_find_const()`, 559-560
- `slapi_valueset_first_value_const()`, 560-561
- `slapi_valueset_free()`, 561
- `slapi_valueset_init()`, 562
- `slapi_valueset_new()`, 562-563
- `slapi_valueset_next_value_const()`, 563-564
- `slapi_valueset_set_from_smod()`, 564-565
- `slapi_valueset_set_valueset_optimised()`, 565-566
- `slapi_vattr_attr_free()`, 566
- `slapi_vattr_attrs_free()`, 566-567
- `slapi_vattr_filter_test()`, 567-568
- `slapi_vattr_is_registered()`, 568
- `slapi_vattr_is_virtual()`, 569
- `slapi_vattr_list_attrs()`, 569-570
- `slapi_vattr_value_compare()`, 570-571
- `slapi_vattr_values_free()`, 571-572
- `slapi_vattr_values_get_ex()`, 572-573
- `slapi_vattr_values_type_thang_get()`, 573-575
- `slapi_wait_condvar()`, 575-576
- summary by functional area, 273-291

Directory Server plug-in API parameters

- extendedop*, 585
- internalpostoperation*, 585
- internalpreoperation*, 585
- LDAP_DEREF_ALWAYS*, 595
- LDAP_DEREF_FINDING*, 595
- LDAP_DEREF_NEVER*, 595
- LDAP_DEREF_SEARCHING*, 595
- LDAP_SCOPE_BASE*, 595
- LDAP_SCOPE_ONELEVEL*, 595
- LDAP_SCOPE_SUBTREE*, 595
- ldbmentryfetchstore*, 585, 588
- matchingrule*, 585
- object*, 585
- passwordcheck*, 585

Directory Server plug-in API parameters (*Continued*)

- postoperation*, 585
- preoperation*, 585
- pwdstoragescheme*, 585
- reverpwdstoragescheme*, 585
- SLAPI_ADD_ENTRY*, 579
- SLAPI_ADD_RESCONTROL*, 594
- SLAPI_ADD_TARGET*, 579
- SLAPI_ARGC*, 582
- SLAPI_ARGV*, 582
- SLAPI_BACKEND*, 579
- SLAPI_BE_LASTMOD*, 579
- SLAPI_BE_READONLY*, 579
- SLAPI_BE_TYPE*, 579
- SLAPI_BIND_CREDENTIALS*, 580
- SLAPI_BIND_METHOD*, 580
- SLAPI_BIND_RET_SASLCREDS*, 580
- SLAPI_BIND_SASLMECHANISM*, 580
- SLAPI_BIND_TARGET*, 580
- SLAPI_CLIENT_DNS*, 581
- SLAPI_COMPARE_TARGET*, 580
- SLAPI_COMPARE_TYPE*, 580
- SLAPI_COMPARE_VALUE*, 580
- SLAPI_CONFIG_DIRECTORY*, 582
- SLAPI_CONFIG_FILENAME*, 582
- SLAPI_CONN_AUTHMETHOD*, 581
- SLAPI_CONN_CLIENTNETADDR*, 581
- SLAPI_CONN_DN*, 581
- SLAPI_CONN_ID*, 581
- SLAPI_CONN_IS_REPLICATION_SESSION*, 581
- SLAPI_CONN_IS_SSL_SESSION*, 581
- SLAPI_CONN_SERVERNETADDR*, 581
- SLAPI_CONNECTION*, 581
- SLAPI_CONTROLS_ARG*, 583
- SLAPI_DBSIZE*, 579
- SLAPI_DELETE_TARGET*, 582
- SLAPI_ENTRY_POST_OP*, 586
- SLAPI_ENTRY_PRE_OP*, 586
- SLAPI_EXT_OP_REQ_OID*, 582
- SLAPI_EXT_OP_REQ_VALUE*, 582
- SLAPI_EXT_OP_RET_OID*, 582
- SLAPI_EXT_OP_RET_VALUE*, 582
- SLAPI_IS_INTERNAL_OPERATION*, 584
- SLAPI_IS_REPLICATED_OPERATION*, 584

Directory Server plug-in API parameters (*Continued*)

SLAPI_MATCHINGRULE_DESC, 588
 SLAPI_MATCHINGRULE_NAME, 588
 SLAPI_MATCHINGRULE_OBSOLETE, 589
 SLAPI_MATCHINGRULE_OID, 589
 SLAPI_MATCHINGRULE_SYNTAX, 589
 SLAPI_MODIFY_MODS, 583
 SLAPI_MODIFY_TARGET, 583
 SLAPI_MODRDN_DELOLDRDN, 593
 SLAPI_MODRDN_NEWRDN, 593
 SLAPI_MODRDN_NEWSUPERIOR, 593
 SLAPI_MODRDN_TARGET, 593
 SLAPI_NENTRIES, 594
 SLAPI_OPERATION, 584
 SLAPI_OPERATION_ID, 584
 SLAPI_OPERATION_MSGID, 584
 SLAPI_OPERATION_NOTES, 578
 SLAPI_OPINITIATED_TIME, 584
 SLAPI_ORIGINAL_TARGET, 582, 583, 593
 SLAPI_PASSWDCHECK_VALS, 586
 SLAPI_PLUGIN, 585
 SLAPI_PLUGIN_ARGC, 585
 SLAPI_PLUGIN_ARGV, 585
 SLAPI_PLUGIN_CLOSE_FN, 586
 SLAPI_PLUGIN_CURRENT_VERSION, 585
 SLAPI_PLUGIN_DESTROY_FN, 589
 SLAPI_PLUGIN_ENTRY_FETCH_FUNC, 588
 SLAPI_PLUGIN_ENTRY_STORE_FUNC, 588
 SLAPI_PLUGIN_EXT_OP_FN, 587
 SLAPI_PLUGIN_EXT_OP_OIDLIST, 587
 SLAPI_PLUGIN_EXTENDEDOP, 585
 SLAPI_PLUGIN_IDENTITY, 585
 SLAPI_PLUGIN_INTERNAL_POST_ADD_FN, 587
 SLAPI_PLUGIN_INTERNAL_POST_DELETE_FN, 587
 SLAPI_PLUGIN_INTERNAL_POST_MODIFY_FN, 587
 SLAPI_PLUGIN_INTERNAL_POST_MODRDN_FN, 587
 SLAPI_PLUGIN_INTERNAL_POSTOPERATION, 585
 SLAPI_PLUGIN_INTERNAL_PRE_ADD_FN, 587
 SLAPI_PLUGIN_INTERNAL_PRE_DELETE_FN, 588
 SLAPI_PLUGIN_INTERNAL_PRE_MODIFY_FN, 588
 SLAPI_PLUGIN_INTERNAL_PRE_MODRDN_FN, 588
 SLAPI_PLUGIN_INTERNAL_PREOPERATION, 585
 SLAPI_PLUGIN_INTOP_RESULT, 583
 SLAPI_PLUGIN_INTOP_SEARCH_ENTRIES, 583

Directory Server plug-in API parameters (*Continued*)

SLAPI_PLUGIN_INTOP_SEARCH_REFERRALS, 583
 SLAPI_PLUGIN_LDBM_ENTRY_FETCH_STORE, 585
 SLAPI_PLUGIN_MATCHINGRULE, 585
 SLAPI_PLUGIN_MR_FILTER_CREATE_FN, 589
 SLAPI_PLUGIN_MR_FILTER_INDEX_FN, 589
 SLAPI_PLUGIN_MR_FILTER_MATCH_FN, 589
 SLAPI_PLUGIN_MR_FILTER_RESET_FN, 589
 SLAPI_PLUGIN_MR_FILTER_REUSABLE, 589
 SLAPI_PLUGIN_MR_INDEX_FN, 589
 SLAPI_PLUGIN_MR_INDEXER_CREATE_FN, 589
 SLAPI_PLUGIN_MR_KEYS, 589
 SLAPI_PLUGIN_MR_OID, 589
 SLAPI_PLUGIN_MR_QUERY_OPERATOR, 590
 SLAPI_PLUGIN_MR_TYPE, 590
 SLAPI_PLUGIN_MR_USAGE, 590
 SLAPI_PLUGIN_MR_VALUE, 590
 SLAPI_PLUGIN_MR_VALUES, 590
 SLAPI_PLUGIN_OBJECT, 590
 SLAPI_PLUGIN_PASSWDCHECK, 585
 SLAPI_PLUGIN_PASSWDCHECK_FN, 586
 SLAPI_PLUGIN_POST_ABANDON_FN, 590
 SLAPI_PLUGIN_POST_ADD_FN, 590
 SLAPI_PLUGIN_POST_BIND_FN, 591
 SLAPI_PLUGIN_POST_COMPARE_FN, 591
 SLAPI_PLUGIN_POST_DELETE_FN, 591
 SLAPI_PLUGIN_POST_ENTRY_FN, 591
 SLAPI_PLUGIN_POST_MODIFY_FN, 591
 SLAPI_PLUGIN_POST_MODRDN_FN, 591
 SLAPI_PLUGIN_POST_REFERRAL_FN, 591
 SLAPI_PLUGIN_POST_RESULT_FN, 591
 SLAPI_PLUGIN_POST_SEARCH_FN, 591
 SLAPI_PLUGIN_POST_UNBIND_FN, 591
 SLAPI_PLUGIN_POSTOPERATION, 585
 SLAPI_PLUGIN_PRE_ABANDON_FN, 591
 SLAPI_PLUGIN_PRE_ADD_FN, 591
 SLAPI_PLUGIN_PRE_BIND_FN, 592
 SLAPI_PLUGIN_PRE_COMPARE_FN, 592
 SLAPI_PLUGIN_PRE_DELETE_FN, 592
 SLAPI_PLUGIN_PRE_ENTRY_FN, 592
 SLAPI_PLUGIN_PRE_MODIFY_FN, 592
 SLAPI_PLUGIN_PRE_MODRDN_FN, 592
 SLAPI_PLUGIN_PRE_REFERRAL_FN, 592
 SLAPI_PLUGIN_PRE_RESULT_FN, 592

Directory Server plug-in API parameters (*Continued*)

`SLAPI_PLUGIN_PRE_SEARCH_FN`, 592
`SLAPI_PLUGIN_PRE_UNBIND_FN`, 592
`SLAPI_PLUGIN_PREOPERATION`, 585
`SLAPI_PLUGIN_PRIVATE`, 585
`SLAPI_PLUGIN_PWD_STORAGE_SCHEME`, 585
`SLAPI_PLUGIN_PWD_STORAGE_SCHEME_CMP_FN`, 592
`SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DB_PWD`, 592
`SLAPI_PLUGIN_PWD_STORAGE_SCHEME_DEC_FN`, 592
`SLAPI_PLUGIN_PWD_STORAGE_SCHEME_ENC_FN`, 592
`SLAPI_PLUGIN_PWD_STORAGE_SCHEME_NAME`, 592
`SLAPI_PLUGIN_PWD_STORAGE_SCHEME_USER_PWD`, 592
`SLAPI_PLUGIN_REVER_PWD_STORAGE_SCHEME`, 585
`SLAPI_PLUGIN_START_FN`, 586
`SLAPI_PLUGIN_TYPE`, 585
`SLAPI_PLUGIN_TYPE_OBJECT`, 585
`SLAPI_PLUGIN_VERSION`, 585
`SLAPI_PLUGIN_VERSION_01`, 585
`SLAPI_PLUGIN_VERSION_02`, 585
`SLAPI_PLUGIN_VERSION_03`, 585
`SLAPI_REQCONTROLS`, 584
`SLAPI_REQUESTOR_DN`, 584
`SLAPI_REQUESTOR_ISROOT`, 584
`SLAPI_RES_CONTROLS`, 594
`SLAPI_RESULT_CODE`, 594
`SLAPI_RESULT_MATCHED`, 594
`SLAPI_RESULT_TEXT`, 594
`SLAPI_SEARCH_ATTRS`, 594
`SLAPI_SEARCH_ATTRSONLY`, 594
`SLAPI_SEARCH_DEREF`, 595
`SLAPI_SEARCH_FILTER`, 595
`SLAPI_SEARCH_REFERRALS`, 595
`SLAPI_SEARCH_RESULT_ENTRY`, 595
`SLAPI_SEARCH_SCOPE`, 595
`SLAPI_SEARCH_SIZELIMIT`, 595
`SLAPI_SEARCH_STRFILTER`, 595
`SLAPI_SEARCH_TARGET`, 595
`SLAPI_SEARCH_TIMELIMIT`, 595
`SLAPI_TARGET_DN`, 584

distinguished names (DN), 71, 109-112

- changing, 111
- finding parent using, 109-110
- finding suffix using, 109-110
- normalizing, 112

distinguished names (DNs), 59

E

- encryption
 - entries, 176-177, 181-182
 - passwords, 223-232
- entries, 59
- creating, 72, 101
- modifying, 72
- reading, 72
- sending to client application, 114-116, 140-141
 - verifying schema compliance, 108
- entry fetch, 175-177
- entry store, 175-177
- error codes, 64
- errors log, 65, 100
- extended operations, 64, 183-191

F

filters

- extensible match, 194, 199-211
- matching entries to, 200-203
- search, 122-126

functions

See Directory Server plug-in API functions

H

hello world, 81-85

I

- indexing, 211-218
- install-path*, 39
- instance-path*, 39
- internal operations, 64, 73, 163-174
 - add, 166-168
 - delete, 174
 - modify, 168-170

internal operations (*Continued*)

- rename or subtree move, 170-171
- search, 171-174

isw-hostname directory, 40

J

Java Naming and Directory Interface, 38

L

LDAP data interchange format (LDIF), 102

local log directory, 40

logging, 97

M

matching rules, 193-222

memory management, 73

Message Queue, 38

modifications, 59

modify operation, 114, 133-135, 163-174

modify RDN operation, 114, 136-138, 163-174

N

NameFinder application, 597

- attribute properties, 602

- configuring, 599-600, 600-601, 602-604

- connection properties, 599-600, 600-601, 602

- customizing, 602-604

- description, 597

- installation, 598

- layout properties, 603

- NameFinder.properties file, 602-604

- prerequisite software, 597

- sample.properties file, 602-604

new features, 59-60, 68-79

O

object extensions, 75

object identifiers (OID), 183-184, 194, 219-222

P

parameters, passing to plug-ins, 94

passwords

- checking, 227

- checking quality of, 233-240

- encrypting and storing, 223-232

plug-ins

- API version, 68, 86-87

- arguments, 94

- call ordering, 92-94

- compiling and linking, 88

- definition, 47-48

- dependencies, 92

- header file, 68, 88

- how they work, 48-53

- identity, 168

- registering, 62, 86, 114-116, 227-229

- return codes, 86

- support, 47-48

- types, 49, 62, 68, 91

- when to use, 47-48

postoperation, 113-116

preoperation, 113-116

R

referrals

- sending to client application, 114-116, 140-141

relative distinguished names (RDN), 77, 136-138

rename operation, 114, 136-138, 163-174

result codes

- sending to client application, 114-116, 140-141

return codes, 86

S

schema, 59

schema (*Continued*)

- verifying compliance of entries with, 108
- search candidates, 199
- search filters, 64
 - breaking down, 122-126
 - handling extensible match, 199-211
 - requesting matching rule, 194
 - Slapi_Filter, 72
- search operation, 114, 120-126, 163-174
- server root directory, 40
- simple authentication and security layer (SASL), 143-144, 155-161
- SLAMD Distributed Load Generation Engine, 38
- sorting, 219
- suffixes, 59, 109-110
- support, 47-48
- syntax, 59

U

- unbind operation, 114
- upgrading, 57-60, 61-79

V

- virtual attributes, 60, 104