

# [MS-ADTS]: Active Directory Technical Specification

---

## Intellectual Property Rights Notice for Protocol Documentation

- This protocol documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the protocols, and may distribute portions of it in your implementations of the protocols or your documentation as necessary to properly document the implementation. This permission also applies to any documents that are referenced in the protocol documentation.
- Microsoft does not claim any trade secret rights in this documentation.
- Microsoft has patents that may cover your implementations of the protocols. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. If you are interested in obtaining a patent license, please contact [protocol@microsoft.com](mailto:protocol@microsoft.com).
- The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them.

## Revision Summary

Date	Revision History	Revision Class	Comments
03/14/2007	1.0		Version 1.0 release
04/10/2007	1.1		Version 1.1 release
05/18/2007	1.2		Version 1.2 release
06/08/2007	1.2.1	Editorial	Revised and edited the technical content.
07/10/2007	2.0	Major	Deleted unused references; corrected type name and

<b>Date</b>	<b>Revision History</b>	<b>Revision Class</b>	<b>Comments</b>
			field information.
08/17/2007	2.0.1	Editorial	Revised and edited the technical content.
09/21/2007	3.0	Major	Made changes to bitfield diagrams; added bitflags.
10/26/2007	3.1	Minor	Updated the technical content.
01/25/2008	3.1.1	Editorial	Revised and edited the technical content.

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>18</b>
1.1	Glossary .....	18
1.1.1	Pervasive Concepts .....	18
1.1.2	Ordinary Glossary Entries .....	19
1.2	References .....	31
1.2.1	Normative References .....	31
1.2.2	Informative References .....	35
1.3	Overview (Synopsis).....	35
1.4	Relationship to Other Protocols .....	36
1.5	Prerequisites/Preconditions .....	36
1.6	Applicability Statement .....	36
1.7	Versioning and Capability Negotiation .....	37
1.8	Vendor-Extensible Fields .....	37
1.9	Standards Assignments.....	37
<b>2</b>	<b>Messages .....</b>	<b>38</b>
2.1	Transport .....	38
2.2	Message Syntax .....	38
2.2.1	LCID-Locale Mapping Table .....	38
2.2.2	DS_REPL_NEIGHBORW_BLOB .....	44
2.2.3	DS_REPL_KCC_DSA_FAILUREW_BLOB .....	48
2.2.4	DS_REPL_OPW_BLOB .....	50
2.2.5	DS_REPL_QUEUE_STATISTICSW_BLOB.....	52
2.2.6	DS_REPL_CURSOR_BLOB .....	53
2.2.7	DS_REPL_ATTR_META_DATA_BLOB .....	55
2.2.8	DS_REPL_VALUE_META_DATA_BLOB .....	56
2.2.9	Search Flags .....	58
2.2.10	System Flags .....	59
2.2.11	schemaFlagsEx Flags .....	60
2.2.12	Group Type Flags .....	60
2.2.13	Security Privilege Flags .....	60
2.2.14	Domain RID Values .....	61
2.2.15	userAccountControl Bits .....	61
<b>3</b>	<b>Details .....</b>	<b>64</b>
3.1	Common Details .....	64
3.1.1	Abstract Data Model.....	64
3.1.1.1	State Model.....	64
3.1.1.1.1	Scope .....	64
3.1.1.1.2	State Modeling Primitives and Notational Conventions .....	65
3.1.1.1.3	Basics, objectGUID, and Special Attribute Behavior .....	66
3.1.1.1.4	objectClass, RDN, DN, Constructed Attributes .....	68
3.1.1.1.5	NC, NC Replica.....	70
3.1.1.1.6	Attribute Syntaxes, Object References, Referential Integrity, and Well-Known Objects .....	73
3.1.1.1.7	Forest, Canonical Name.....	76
3.1.1.1.8	GC .....	78
3.1.1.1.9	DCs, usn Counters, and the Originating Update Stamp.....	78
3.1.1.1.10	GC Server .....	83
3.1.1.1.11	FSMO Roles .....	83
3.1.1.1.12	Cross-NC Object References .....	84
3.1.1.1.13	NC Replica Graph .....	85

3.1.1.1.14	Scheduled and Event-Driven Replication .....	86
3.1.1.1.15	Replication Latency and Tombstone Lifetime .....	87
3.1.1.2	Active Directory Schema .....	87
3.1.1.2.1	Schema NC .....	87
3.1.1.2.2	Syntaxes .....	88
3.1.1.2.2.1	Introduction .....	88
3.1.1.2.2.2	LDAP Representations .....	89
3.1.1.2.2.2.1	Object(DN-String) .....	91
3.1.1.2.2.2.2	Object(Access-Point) .....	91
3.1.1.2.2.2.3	Object(DN-Binary) .....	91
3.1.1.2.2.2.4	Object(OR-Name) .....	91
3.1.1.2.2.2.5	String(Case) .....	91
3.1.1.2.2.2.6	String(NT-Sec-Desc) .....	92
3.1.1.2.2.2.7	String(Sid) .....	92
3.1.1.2.2.2.8	String(Teletex) .....	92
3.1.1.2.2.3	Referential Integrity .....	92
3.1.1.2.2.4	Supported Comparison Operations .....	92
3.1.1.2.2.4.1	Bool Comparison Rule .....	96
3.1.1.2.2.4.2	Integer Comparison Rule .....	96
3.1.1.2.2.4.3	DN-String Comparison Rule .....	96
3.1.1.2.2.4.4	DN-Binary Comparison Rule .....	96
3.1.1.2.2.4.5	DN Comparison Rule .....	96
3.1.1.2.2.4.6	PresentationAddress Comparison Rule .....	96
3.1.1.2.2.4.7	Octet Comparison Rule .....	97
3.1.1.2.2.4.8	CaseString Comparison Rule .....	97
3.1.1.2.2.4.9	SecDesc Comparison Rule .....	97
3.1.1.2.2.4.10	OID Comparison Rule .....	97
3.1.1.2.2.4.11	Sid Comparison Rule .....	97
3.1.1.2.2.4.12	NoCaseString Comparison Rule .....	97
3.1.1.2.2.4.13	UnicodeString Comparison Rule .....	97
3.1.1.2.2.4.14	Time Comparison Rule .....	98
3.1.1.2.3	Attributes .....	98
3.1.1.2.3.1	Auto-Generated linkID .....	101
3.1.1.2.3.2	Auto-Generated mAPIID .....	102
3.1.1.2.3.3	Property Set .....	102
3.1.1.2.3.4	ldapDisplayName Generation .....	103
3.1.1.2.3.5	Flag fRODCFilteredAttribute in Attribute SearchFlags .....	103
3.1.1.2.4	Classes .....	104
3.1.1.2.4.1	Class Categories .....	104
3.1.1.2.4.2	Inheritance .....	104
3.1.1.2.4.3	objectClass .....	105
3.1.1.2.4.4	Structure Rules .....	105
3.1.1.2.4.5	Content Rules .....	105
3.1.1.2.4.6	Auxiliary Class .....	105
3.1.1.2.4.7	RDN Attribute of a Class .....	106
3.1.1.2.4.8	Class classSchema .....	106
3.1.1.2.5	Schema Modifications .....	108
3.1.1.2.5.1	Consistency and Safety Checks .....	108
3.1.1.2.5.1.1	Consistency Checks .....	108
3.1.1.2.5.1.2	Safety Checks .....	109
3.1.1.2.5.2	Defunct .....	110
3.1.1.2.5.2.1	Forest Functional Level Less Than WIN2003 .....	110
3.1.1.2.5.2.2	Forest Functional Level WIN2003 or Greater .....	111
3.1.1.2.6	ATTRTYP .....	112
3.1.1.3	LDAP Protocol .....	112



3.1.1.3.1	LDAP Conformance .....	113
3.1.1.3.1.1	Schema .....	113
3.1.1.3.1.1.1	subSchema .....	113
3.1.1.3.1.1.2	Syntaxes .....	115
3.1.1.3.1.1.3	Attributes .....	115
3.1.1.3.1.1.4	Classes .....	123
3.1.1.3.1.1.5	Auxiliary Classes .....	126
3.1.1.3.1.2	Object Naming .....	127
3.1.1.3.1.2.1	Naming Attributes .....	127
3.1.1.3.1.2.2	NC Naming .....	127
3.1.1.3.1.2.3	Multivalued and Multiple-Attribute RDNs .....	128
3.1.1.3.1.2.4	Alternative Forms of DNS .....	128
3.1.1.3.1.2.5	Alternative Form of SIDs .....	129
3.1.1.3.1.3	Search Operations .....	129
3.1.1.3.1.3.1	Search Filters .....	129
3.1.1.3.1.3.2	Range Retrieval of Attribute Values .....	130
3.1.1.3.1.3.3	Ambiguous Name Resolution .....	131
3.1.1.3.1.3.4	Searches Using the objectCategory Attribute .....	132
3.1.1.3.1.3.5	Restrictions on rootDSE Searches .....	133
3.1.1.3.1.4	Referrals in LDAP v2 and v3 .....	133
3.1.1.3.1.5	Password Modify Operations .....	133
3.1.1.3.1.5.1	unicodePwd .....	133
3.1.1.3.1.5.2	userPassword .....	135
3.1.1.3.1.6	Dynamic Objects .....	135
3.1.1.3.1.7	Modify DN Operations .....	136
3.1.1.3.1.8	Aliases .....	136
3.1.1.3.1.9	Error Message Strings .....	136
3.1.1.3.1.10	Ports .....	136
3.1.1.3.2	rootDSE Attributes .....	136
3.1.1.3.2.1	configurationNamingContext .....	140
3.1.1.3.2.2	currentTime .....	141
3.1.1.3.2.3	defaultNamingContext .....	141
3.1.1.3.2.4	dNSHostName .....	141
3.1.1.3.2.5	dsSchemaAttrCount .....	141
3.1.1.3.2.6	dsSchemaClassCount .....	141
3.1.1.3.2.7	dsSchemaPrefixCount .....	141
3.1.1.3.2.8	dsServiceName .....	141
3.1.1.3.2.9	highestCommittedUSN .....	141
3.1.1.3.2.10	isGlobalCatalogReady .....	141
3.1.1.3.2.11	isSynchronized .....	141
3.1.1.3.2.12	ldapServiceName .....	141
3.1.1.3.2.13	namingContexts .....	142
3.1.1.3.2.14	netlogon .....	142
3.1.1.3.2.15	pendingPropagations .....	142
3.1.1.3.2.16	rootDomainNamingContext .....	142
3.1.1.3.2.17	schemaNamingContext .....	142
3.1.1.3.2.18	serverName .....	142
3.1.1.3.2.19	subschemaSubentry .....	142
3.1.1.3.2.20	supportedCapabilities .....	142
3.1.1.3.2.21	supportedControl .....	142
3.1.1.3.2.22	supportedLDAPPolicies .....	143
3.1.1.3.2.23	supportedLDAPVersion .....	143
3.1.1.3.2.24	supportedSASLMechanisms .....	143
3.1.1.3.2.25	domainControllerFunctionality .....	143
3.1.1.3.2.26	domainFunctionality .....	143

3.1.1.3.2.27	forestFunctionality .....	143
3.1.1.3.2.28	msDS-ReplAllInboundNeighbors, msDS-ReplConnectionFailures, msDS-ReplLinkFailures, and msDS-ReplPendingOps .....	144
3.1.1.3.2.29	msDS-ReplAllOutboundNeighbors .....	145
3.1.1.3.2.30	msDS-ReplQueueStatistics .....	145
3.1.1.3.2.31	msDS-TopQuotaUsage.....	146
3.1.1.3.2.32	supportedConfigurableSettings .....	147
3.1.1.3.2.33	supportedExtension.....	147
3.1.1.3.2.34	validFSMOs .....	147
3.1.1.3.2.35	dsaVersionString.....	148
3.1.1.3.2.36	msDS-PortLDAP.....	148
3.1.1.3.2.37	msDS-PortSSL.....	148
3.1.1.3.2.38	msDS-PrincipalName.....	148
3.1.1.3.2.39	serviceAccountInfo .....	149
3.1.1.3.2.40	spnRegistrationResult.....	149
3.1.1.3.2.41	tokenGroups .....	149
3.1.1.3.2.42	usnAtRifm .....	149
3.1.1.3.3	rootDSE Modify Operations .....	149
3.1.1.3.3.1	becomeDomainMaster .....	151
3.1.1.3.3.2	becomeInfrastructureMaster.....	152
3.1.1.3.3.3	becomePdc .....	152
3.1.1.3.3.4	becomePdcWithCheckPoint.....	152
3.1.1.3.3.5	becomeRidMaster .....	152
3.1.1.3.3.6	becomeSchemaMaster.....	153
3.1.1.3.3.7	checkPhantoms .....	153
3.1.1.3.3.8	doGarbageCollection .....	154
3.1.1.3.3.9	dumpDatabase .....	154
3.1.1.3.3.10	fixupInheritance .....	154
3.1.1.3.3.11	invalidateRidPool .....	155
3.1.1.3.3.12	recalcHierarchy.....	156
3.1.1.3.3.13	schemaUpdateNow.....	156
3.1.1.3.3.14	removeLingeringObject.....	156
3.1.1.3.3.15	doLinkCleanup .....	157
3.1.1.3.3.16	doOnlineDefrag.....	157
3.1.1.3.3.17	replicateSingleObject.....	158
3.1.1.3.3.18	updateCachedMemberships .....	158
3.1.1.3.3.19	doGarbageCollectionPhantomsNow.....	159
3.1.1.3.3.20	invalidateGCCConnection .....	159
3.1.1.3.3.21	renewServerCertificate .....	159
3.1.1.3.3.22	rODCPurgeAccount.....	160
3.1.1.3.3.23	runSamUpgradeTasks.....	160
3.1.1.3.3.24	sqmRunOnce.....	161
3.1.1.3.4	LDAP Extensions .....	161
3.1.1.3.4.1	LDAP Extended Controls .....	162
3.1.1.3.4.1.1	LDAP_PAGED_RESULT_OID_STRING .....	166
3.1.1.3.4.1.2	LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID .....	167
3.1.1.3.4.1.3	LDAP_SERVER_DIRSYNC_OID.....	167
3.1.1.3.4.1.4	LDAP_SERVER_DOMAIN_SCOPE_OID .....	169
3.1.1.3.4.1.5	LDAP_SERVER_EXTENDED_DN_OID .....	169
3.1.1.3.4.1.6	LDAP_SERVER_GET_STATS_OID.....	170
3.1.1.3.4.1.7	LDAP_SERVER_LAZY_COMMIT_OID .....	173
3.1.1.3.4.1.8	LDAP_SERVER_PERMISSIVE_MODIFY_OID .....	173
3.1.1.3.4.1.9	LDAP_SERVER_NOTIFICATION_OID .....	174
3.1.1.3.4.1.10	LDAP_SERVER_RANGE_OPTION_OID.....	174
3.1.1.3.4.1.11	LDAP_SERVER_SD_FLAGS_OID.....	174

3.1.1.3.4.1.12	LDAP_SERVER_SEARCH_OPTIONS_OID .....	175
3.1.1.3.4.1.13	LDAP_SERVER_SORT_OID and LDAP_SERVER_RESP_SORT_OID .....	176
3.1.1.3.4.1.14	LDAP_SERVER_SHOW_DELETED_OID .....	183
3.1.1.3.4.1.15	LDAP_SERVER_TREE_DELETE_OID .....	183
3.1.1.3.4.1.16	LDAP_SERVER_VERIFY_NAME_OID .....	183
3.1.1.3.4.1.17	LDAP_CONTROL_VLVREQUEST and LDAP_CONTROL_VLVRESPONSE ..	184
3.1.1.3.4.1.18	LDAP_SERVER_ASQ_OID .....	186
3.1.1.3.4.1.19	LDAP_SERVER_QUOTA_CONTROL_OID .....	187
3.1.1.3.4.1.20	LDAP_SERVER_SHUTDOWN_NOTIFY_OID .....	187
3.1.1.3.4.1.21	LDAP_SERVER_FORCE_UPDATE_OID .....	188
3.1.1.3.4.1.22	LDAP_SERVER_RANGE_RETRIEVAL_NOERR_OID .....	188
3.1.1.3.4.1.23	LDAP_SERVER_RODC_DCPROMO_OID .....	188
3.1.1.3.4.1.24	LDAP_SERVER_INPUT_DN_OID .....	189
3.1.1.3.4.2	LDAP Extended Operations .....	189
3.1.1.3.4.2.1	LDAP_SERVER_FAST_BIND_OID .....	190
3.1.1.3.4.2.2	LDAP_SERVER_START_TLS_OID .....	191
3.1.1.3.4.2.3	LDAP_TTL_REFRESH_OID .....	191
3.1.1.3.4.2.4	LDAP_SERVER_WHO_AM_I_OID .....	191
3.1.1.3.4.3	LDAP Capabilities .....	191
3.1.1.3.4.3.1	LDAP_CAP_ACTIVE_DIRECTORY_OID .....	192
3.1.1.3.4.3.2	LDAP_CAP_ACTIVE_DIRECTORY_LDAP_INTEG_OID .....	192
3.1.1.3.4.3.3	LDAP_CAP_ACTIVE_DIRECTORY_V51_OID .....	192
3.1.1.3.4.3.4	LDAP_CAP_ACTIVE_DIRECTORY_ADAM_DIGEST .....	192
3.1.1.3.4.3.5	LDAP_CAP_ACTIVE_DIRECTORY_ADAM_OID .....	193
3.1.1.3.4.3.6	LDAP_CAP_ACTIVE_DIRECTORY_PARTIAL_SECRETS_OID .....	193
3.1.1.3.4.3.7	LDAP_CAP_ACTIVE_DIRECTORY_V61_OID .....	193
3.1.1.3.4.4	LDAP Matching Rules (extensibleMatch) .....	193
3.1.1.3.4.4.1	LDAP_MATCHING_RULE_BIT_AND .....	193
3.1.1.3.4.4.2	LDAP_MATCHING_RULE_BIT_OR .....	193
3.1.1.3.4.4.3	LDAP_MATCHING_RULE_TRANSITIVE_EVAL .....	193
3.1.1.3.4.5	LDAP SASL Mechanisms .....	194
3.1.1.3.4.5.1	GSSAPI .....	194
3.1.1.3.4.5.2	GSS-SPNEGO .....	195
3.1.1.3.4.5.3	EXTERNAL .....	195
3.1.1.3.4.5.4	DIGEST-MD5 .....	195
3.1.1.3.4.6	LDAP Policies .....	195
3.1.1.3.4.7	LDAP Configurable Settings .....	197
3.1.1.3.4.8	LDAP IP-Deny List .....	200
3.1.1.4	Reads .....	200
3.1.1.4.1	Introduction .....	201
3.1.1.4.2	Definitions .....	201
3.1.1.4.3	Access Checks .....	202
3.1.1.4.4	Extended Access Checks .....	202
3.1.1.4.5	Constructed Attributes .....	204
3.1.1.4.5.1	subSchemaSubEntry .....	204
3.1.1.4.5.2	canonicalName .....	204
3.1.1.4.5.3	allowedChildClasses .....	205
3.1.1.4.5.4	sDRightsEffective .....	205
3.1.1.4.5.5	allowedChildClassesEffective .....	205
3.1.1.4.5.6	allowedAttributes .....	206
3.1.1.4.5.7	allowedAttributesEffective .....	206
3.1.1.4.5.8	fromEntry .....	206
3.1.1.4.5.9	createTimeStamp .....	206
3.1.1.4.5.10	modifyTimeStamp .....	206
3.1.1.4.5.11	primaryGroupToken .....	207

3.1.1.4.5.12	entryTTL.....	207
3.1.1.4.5.13	msDS-NCReplInboundNeighbors, msDS-NCReplCursors, msDS- ReplAttributeMetaData, msDS-ReplValueMetaData .....	207
3.1.1.4.5.14	msDS-NCReplOutboundNeighbors .....	208
3.1.1.4.5.15	msDS-Approx-Immed-Subordinates .....	208
3.1.1.4.5.16	msDS-KeyVersionNumber .....	208
3.1.1.4.5.17	msDS-User-Account-Control-Computed .....	208
3.1.1.4.5.18	msDS-Auxiliary-Classes .....	210
3.1.1.4.5.19	tokenGroups, tokenGroupsNoGCAcceptable.....	210
3.1.1.4.5.20	tokenGroupsGlobalAndUniversal .....	210
3.1.1.4.5.21	possibleInferiors .....	211
3.1.1.4.5.22	msDS-QuotaEffective .....	211
3.1.1.4.5.23	msDS-QuotaUsed .....	212
3.1.1.4.5.24	msDS-TopQuotaUsage.....	212
3.1.1.4.5.25	ms-DS-UserAccountAutoLocked .....	213
3.1.1.4.5.26	msDS-UserPasswordExpired .....	213
3.1.1.4.5.27	msDS-PrincipalName .....	214
3.1.1.4.5.28	parentGUID.....	214
3.1.1.4.5.29	msDS-SiteName .....	214
3.1.1.4.5.30	msDS-isRODC .....	214
3.1.1.4.5.31	msDS-isGC .....	215
3.1.1.4.5.32	msDS-isUserCachableAtRodc.....	215
3.1.1.4.5.33	msDS-UserPasswordExpiryTimeComputed .....	216
3.1.1.4.5.34	msDS-RevealedList .....	216
3.1.1.4.5.35	msDS-RevealedListBL.....	217
3.1.1.4.5.36	msDS-ResultantPSO .....	217
3.1.1.4.6	Referrals .....	218
3.1.1.4.7	Continuations.....	220
3.1.1.4.8	Effects of Defunct Attributes and Classes.....	220
3.1.1.5	Updates .....	221
3.1.1.5.1	General.....	221
3.1.1.5.1.1	Enforce Schema Constraints.....	221
3.1.1.5.1.2	Naming Constraints.....	221
3.1.1.5.1.3	Transactional Semantics .....	222
3.1.1.5.1.4	Stamp Construction .....	222
3.1.1.5.1.5	Replication Notification .....	222
3.1.1.5.1.6	Urgent Replication .....	223
3.1.1.5.1.7	Updates Performed Only on FSMOs .....	223
3.1.1.5.1.8	Originating Updates Attempted on an RODC.....	226
3.1.1.5.1.9	Constraints and Processing Specifics Defined Elsewhere.....	226
3.1.1.5.2	Add Operation.....	226
3.1.1.5.2.1	Security Considerations .....	227
3.1.1.5.2.2	Constraints .....	227
3.1.1.5.2.3	Special Classes and Attributes .....	231
3.1.1.5.2.4	Processing Specifics .....	232
3.1.1.5.2.5	Quota Invariants.....	234
3.1.1.5.2.6	NC Invariants .....	235
3.1.1.5.2.7	crossRef Invariants .....	235
3.1.1.5.2.8	NC-Add Operation.....	235
3.1.1.5.2.8.1	Constraints.....	236
3.1.1.5.2.8.2	Security Considerations.....	236
3.1.1.5.2.8.3	Processing Specifics.....	236
3.1.1.5.3	Modify Operation .....	237
3.1.1.5.3.1	Security Considerations .....	237
3.1.1.5.3.1.1	Validated Writes.....	238

3.1.1.5.3.1.1.1	Member.....	238
3.1.1.5.3.1.1.2	dNSHostName.....	238
3.1.1.5.3.1.1.3	msDS-AdditionalDnsHostName .....	238
3.1.1.5.3.1.1.4	servicePrincipalName .....	238
3.1.1.5.3.1.2	FSMO Changes.....	239
3.1.1.5.3.2	Constraints .....	239
3.1.1.5.3.3	Processing Specifics .....	243
3.1.1.5.3.4	BehaviorVersion Updates .....	244
3.1.1.5.3.5	ObjectClass Updates.....	244
3.1.1.5.3.6	wellKnownObjects Updates .....	245
3.1.1.5.3.7	Undelete Operation .....	245
3.1.1.5.3.7.1	Undelete Security Considerations .....	246
3.1.1.5.3.7.2	Undelete Constraints .....	246
3.1.1.5.3.7.3	Undelete Processing Specifics .....	246
3.1.1.5.4	Modify DN .....	247
3.1.1.5.4.1	Intra Domain Modify DN .....	248
3.1.1.5.4.1.1	Security Considerations.....	248
3.1.1.5.4.1.2	Constraints.....	249
3.1.1.5.4.1.3	Processing Specifics.....	250
3.1.1.5.4.2	Cross Domain Move .....	250
3.1.1.5.4.2.1	Security Considerations.....	250
3.1.1.5.4.2.2	Constraints.....	251
3.1.1.5.4.2.3	Processing Specifics.....	253
3.1.1.5.5	Delete Operation .....	255
3.1.1.5.5.1	Tombstone Invariants.....	256
3.1.1.5.5.2	dynamicObject Invariants .....	257
3.1.1.5.5.3	Protected Objects .....	257
3.1.1.5.5.4	Security Considerations .....	257
3.1.1.5.5.5	Constraints .....	257
3.1.1.5.5.6	Processing Specifics .....	258
3.1.1.5.5.7	Tree-delete Operation .....	259
3.1.1.5.5.7.1	Tree-delete Security Considerations.....	259
3.1.1.5.5.7.2	Tree-delete Constraints.....	259
3.1.1.5.5.7.3	Tree-delete Processing Specifics.....	259
3.1.1.6	Background Tasks.....	260
3.1.1.6.1	AdminSDHolder.....	260
3.1.1.6.1.1	Authoritative Security Descriptor .....	261
3.1.1.6.1.2	Protected Objects .....	261
3.1.1.6.1.3	Protection Operation .....	261
3.1.1.6.1.4	Configurable State .....	262
3.1.1.6.2	Reference Update .....	262
3.1.1.7	NT4 Replication Support .....	263
3.1.1.7.1	Format of nt4ReplicationState and pdcChangeLog .....	263
3.1.1.7.1.1	nt4ReplicationState .....	263
3.1.1.7.1.2	pdcChangeLog.....	263
3.1.1.7.2	State Changes .....	264
3.1.1.7.2.1	Initialization .....	264
3.1.1.7.2.2	Directory Updates .....	264
3.1.1.7.2.3	Acquiring the PDC Role.....	268
3.1.1.7.2.4	Resetting the pdcChangeLog .....	269
3.1.1.7.3	Format of referent of pmsgOut.V1.pLog .....	269
3.1.1.8	AD/LDS Special Objects .....	270
3.1.1.8.1	AD/LDS Users .....	270
3.1.1.8.2	Bind Proxies.....	270

<b>4</b>	<b>Protocol Examples .....</b>	<b>272</b>
<b>5</b>	<b>Security .....</b>	<b>273</b>
5.1	LDAP Security .....	273
5.1.1	Authentication.....	273
5.1.1.1	Supported Authentication Methods .....	273
5.1.1.1.1	Simple Authentication .....	274
5.1.1.1.2	SASL Authentication .....	275
5.1.1.1.3	Sicily Authentication .....	276
5.1.1.2	Using SSL/TLS.....	277
5.1.1.3	Using Fast Bind.....	278
5.1.1.4	Mutual Authentication.....	279
5.1.1.5	Supported Types of Security Principals .....	279
5.1.2	Message Security .....	281
5.1.2.1	Using SASL .....	281
5.1.2.2	Using SSL/TLS.....	281
5.1.3	Authorization .....	281
5.1.3.1	Background.....	281
5.1.3.2	Access Rights .....	282
5.1.3.2.1	Control Access Rights.....	284
5.1.3.2.2	Validated Writes .....	287
5.1.3.3	Checking Access .....	287
5.1.3.3.1	Null vs. Empty DACLS .....	288
5.1.3.3.2	Checking Simple Access .....	288
5.1.3.3.3	Checking Object Specific Access .....	289
5.1.3.3.4	Checking Control Access Right-Based Access .....	291
5.1.3.3.5	Checking Validated Write-Based Access.....	291
5.1.3.3.6	Checking Object Visibility .....	292
5.1.3.4	AD/LDS Security Context Construction.....	293
<b>6</b>	<b>Index of Version-Specific Behavior .....</b>	<b>294</b>
<b>7</b>	<b>Additional Information .....</b>	<b>298</b>
7.1	Special Objects and Forest Invariants .....	298
7.1.1	Special Objects .....	298
7.1.1.1	Naming Contexts .....	298
7.1.1.1.1	Any NC Root .....	298
7.1.1.1.2	Config NC Root.....	299
7.1.1.1.3	Schema NC Root .....	300
7.1.1.1.4	Domain NC Root.....	301
7.1.1.1.5	Application NC Root .....	301
7.1.1.2	Configuration Objects .....	302
7.1.1.2.1	Cross-Ref-Container Container .....	303
7.1.1.2.1.1	Cross-Ref Objects .....	303
7.1.1.2.1.1.1	Foreign crossRef Objects .....	304
7.1.1.2.1.1.2	Configuration crossRef Object .....	304
7.1.1.2.1.1.3	Schema crossRef Object .....	304
7.1.1.2.1.1.4	Domain crossRef Object .....	305
7.1.1.2.1.1.5	Application NC crossRef Object.....	305
7.1.1.2.2	Sites Container .....	305
7.1.1.2.2.1	Site Object.....	306
7.1.1.2.2.1.1	NTDS Site Settings Object .....	306
7.1.1.2.2.1.2	Servers Container .....	307
7.1.1.2.2.1.2.1	Server Object.....	307
7.1.1.2.2.1.2.1.1	nTDSDSA Object .....	308
7.1.1.2.2.1.2.1.2	Connection Object.....	309

7.1.1.2.2.2	Subnets Container .....	311
7.1.1.2.2.2.1	Subnet Object.....	312
7.1.1.2.2.3	Inter-Site Transports Container .....	313
7.1.1.2.2.3.1	IP Transport Container .....	313
7.1.1.2.2.3.2	SMTP Transport Container .....	313
7.1.1.2.2.3.3	Site Link Object .....	314
7.1.1.2.2.3.4	Site Link Bridge Object.....	315
7.1.1.2.3	Display Specifiers Container.....	315
7.1.1.2.3.1	Display Specifier Object.....	315
7.1.1.2.4	Services .....	317
7.1.1.2.4.1	Windows NT .....	317
7.1.1.2.4.1.1	Directory Service.....	317
7.1.1.2.4.1.2	dSHeuristics .....	318
7.1.1.2.4.1.3	Query-Policies.....	320
7.1.1.2.4.1.3.1	Default Query Policy.....	321
7.1.1.2.4.1.4	SCP Publication Service Object.....	321
7.1.1.2.5	Physical Locations.....	321
7.1.1.2.6	WellKnown Security Principals.....	321
7.1.1.2.6.1	Anonymous Logon .....	322
7.1.1.2.6.2	Authenticated Users .....	322
7.1.1.2.6.3	Batch .....	322
7.1.1.2.6.4	Creator Group .....	322
7.1.1.2.6.5	Creator Owner.....	322
7.1.1.2.6.6	Dialup .....	322
7.1.1.2.6.7	Digest Authentication .....	322
7.1.1.2.6.8	Enterprise Domain Controllers .....	323
7.1.1.2.6.9	Everyone .....	323
7.1.1.2.6.10	Interactive .....	323
7.1.1.2.6.11	IUSR .....	323
7.1.1.2.6.12	Local Service.....	323
7.1.1.2.6.13	Network.....	323
7.1.1.2.6.14	Network Service .....	323
7.1.1.2.6.15	NTLM Authentication .....	324
7.1.1.2.6.16	Other Organization.....	324
7.1.1.2.6.17	Owner Rights .....	324
7.1.1.2.6.18	Proxy .....	324
7.1.1.2.6.19	Remote Interactive Logon .....	324
7.1.1.2.6.20	Restricted .....	324
7.1.1.2.6.21	SChannel Authentication.....	324
7.1.1.2.6.22	Self.....	325
7.1.1.2.6.23	Service .....	325
7.1.1.2.6.24	System.....	325
7.1.1.2.6.25	Terminal Server User .....	325
7.1.1.2.6.26	This Organization.....	325
7.1.1.2.7	Extended Rights .....	325
7.1.1.2.7.1	controlAccessRight objects .....	325
7.1.1.2.7.2	Change-Rid-Master .....	326
7.1.1.2.7.3	Do-Garbage-Collection .....	326
7.1.1.2.7.4	Recalculate-Hierarchy.....	326
7.1.1.2.7.5	Allocate-Rids .....	326
7.1.1.2.7.6	Change-PDC.....	326
7.1.1.2.7.7	Add-GUID .....	326
7.1.1.2.7.8	Change-Domain-Master .....	327
7.1.1.2.7.9	Public-Information .....	327
7.1.1.2.7.10	msmq-Receive-Dead-Letter.....	327

7.1.1.2.7.11	msmq-Peek-Dead-Letter .....	327
7.1.1.2.7.12	msmq-Receive-computer-Journal .....	327
7.1.1.2.7.13	msmq-Peek-computer-Journal .....	327
7.1.1.2.7.14	msmq-Receive .....	327
7.1.1.2.7.15	msmq-Peek .....	328
7.1.1.2.7.16	msmq-Send .....	328
7.1.1.2.7.17	msmq-Receive-journal .....	328
7.1.1.2.7.18	msmq-Open-Connector .....	328
7.1.1.2.7.19	Apply-Group-Policy .....	328
7.1.1.2.7.20	RAS-Information .....	328
7.1.1.2.7.21	DS-Install-Replica .....	328
7.1.1.2.7.22	Change-Infrastructure-Master .....	329
7.1.1.2.7.23	Update-Schema-Cache .....	329
7.1.1.2.7.24	Recalculate-Security-Inheritance .....	329
7.1.1.2.7.25	DS-Check-Stale-Phantoms .....	329
7.1.1.2.7.26	Certificate-Enrollment .....	329
7.1.1.2.7.27	Self-Membership .....	329
7.1.1.2.7.28	Validated-DNS-Host-Name .....	329
7.1.1.2.7.29	Validated-SPN .....	330
7.1.1.2.7.30	Generate-RSoP-Planning .....	330
7.1.1.2.7.31	Refresh-Group-Cache .....	330
7.1.1.2.7.32	Reload-SSL-Certificate .....	330
7.1.1.2.7.33	SAM-Enumerate-Entire-Domain .....	330
7.1.1.2.7.34	Generate-RSoP-Logging .....	330
7.1.1.2.7.35	Domain-Other-Parameters .....	330
7.1.1.2.7.36	DNS-Host-Name-Attributes .....	331
7.1.1.2.7.37	Create-Inbound-Forest-Trust .....	331
7.1.1.2.7.38	DS-Replication-Get-Changes-All .....	331
7.1.1.2.7.39	Migrate-SID-History .....	331
7.1.1.2.7.40	Reanimate-Tombstones .....	331
7.1.1.2.7.41	Allowed-To-Authenticate .....	331
7.1.1.2.7.42	DS-Execute-Intentions-Script .....	331
7.1.1.2.7.43	DS-Replication-Monitor-Topology .....	332
7.1.1.2.7.44	Update-Password-Not-Required-Bit .....	332
7.1.1.2.7.45	Unexpire-Password .....	332
7.1.1.2.7.46	Enable-Per-User-Reversibly-Encrypted-Password .....	332
7.1.1.2.7.47	DS-Query-Self-Quota .....	332
7.1.1.2.7.48	Private-Information .....	332
7.1.1.2.7.49	MS-TS-GatewayAccess .....	332
7.1.1.2.7.50	Terminal-Server-License-Server .....	333
7.1.1.2.7.51	Domain-Administer-Server .....	333
7.1.1.2.7.52	User-Change-Password .....	333
7.1.1.2.7.53	User-Force-Change-Password .....	333
7.1.1.2.7.54	Send-As .....	333
7.1.1.2.7.55	Receive-As .....	333
7.1.1.2.7.56	Send-To .....	333
7.1.1.2.7.57	Domain-Password .....	334
7.1.1.2.7.58	General-Information .....	334
7.1.1.2.7.59	User-Account-Restrictions .....	334
7.1.1.2.7.60	User-Logon .....	334
7.1.1.2.7.61	Membership .....	334
7.1.1.2.7.62	Open-Address-Book .....	334
7.1.1.2.7.63	Personal-Information .....	335
7.1.1.2.7.64	Email-Information .....	335
7.1.1.2.7.65	Web-Information .....	335



7.1.1.2.7.66	DS-Replication-Get-Changes .....	335
7.1.1.2.7.67	DS-Replication-Synchronize .....	336
7.1.1.2.7.68	DS-Replication-Manage-Topology .....	336
7.1.1.2.7.69	Change-Schema-Master .....	336
7.1.1.2.7.70	DS-Replication-Get-Changes-In-Filtered-Set .....	336
7.1.1.2.8	Forest Updates Container .....	336
7.1.1.2.8.1	Operations Container .....	337
7.1.1.2.8.2	Windows2003Update Container .....	337
7.1.1.2.8.3	ActiveDirectoryUpdate Container .....	337
7.1.1.2.8.4	ActiveDirectoryRodcUpdate Container .....	337
7.1.1.3	Critical Domain Objects .....	337
7.1.1.3.1	Domain Controller Object .....	338
7.1.1.3.2	Read-Only Domain Controller Object .....	339
7.1.1.4	Well-Known Objects .....	340
7.1.1.4.1	Lost and Found Container .....	342
7.1.1.4.2	Deleted Objects Container .....	342
7.1.1.4.3	NTDS Quotas Container .....	343
7.1.1.4.4	Infrastructure Object .....	343
7.1.1.4.5	Domain Controllers OU .....	343
7.1.1.4.6	Users Container .....	343
7.1.1.4.7	Computers Container .....	343
7.1.1.4.8	Program Data Container .....	344
7.1.1.4.9	Foreign Security Principals Container .....	344
7.1.1.4.10	System Container .....	344
7.1.1.4.10.1	Password Settings Container .....	344
7.1.1.4.11	Builtin Container .....	345
7.1.1.4.11.1	Account Operators Group Object .....	345
7.1.1.4.11.2	Administrators Group Object .....	345
7.1.1.4.11.3	Backup Operators Group Object .....	345
7.1.1.4.11.4	Certificate Service DCOM Access Group Object .....	346
7.1.1.4.11.5	Cryptographic Operators Group Object .....	346
7.1.1.4.11.6	Distributed COM Users Group Object .....	346
7.1.1.4.11.7	Event Log Readers Group Object .....	346
7.1.1.4.11.8	Guests Group Object .....	346
7.1.1.4.11.9	IIS_IUSRS Group Object .....	346
7.1.1.4.11.10	Incoming Forest Trust Builders Group Object .....	346
7.1.1.4.11.11	Network Configuration Operators Group Object .....	346
7.1.1.4.11.12	Performance Log Users Group Object .....	347
7.1.1.4.11.13	Performance Monitor Users Group Object .....	347
7.1.1.4.11.14	Pre-Windows 2000 Compatible Access Group Object .....	347
7.1.1.4.11.15	Print Operators Group Object .....	347
7.1.1.4.11.16	Remote Desktop Users Group Object .....	347
7.1.1.4.11.17	Replicator Group Object .....	347
7.1.1.4.11.18	Server Operators Group Object .....	347
7.1.1.4.11.19	Terminal Server License Servers Group Object .....	347
7.1.1.4.11.20	Users Group Object .....	348
7.1.1.4.11.21	Windows Authorization Access Group Group Object .....	348
7.1.1.4.12	Roles Container .....	348
7.1.1.4.12.1	Administrators Group Object .....	348
7.1.1.4.12.2	Readers Group Object .....	349
7.1.1.4.12.3	Users Group Object .....	349
7.1.1.4.12.4	Instances Group Object .....	349
7.1.1.5	Other Well-Known Objects .....	349
7.1.1.5.1	AdminSDHolder Object .....	349
7.1.1.5.2	Default Domain Policy Container .....	349

7.1.1.5.3	Sam Server Object .....	350
7.1.1.5.4	Domain Updates Container .....	350
7.1.1.5.4.1	Operations Container.....	350
7.1.1.5.4.2	Windows2003Updates Container .....	350
7.1.1.5.4.3	ActiveDirectoryUpdate Container .....	350
7.1.1.6	Well-Known Domain-Relative Security Principals .....	350
7.1.1.6.1	Administrator .....	351
7.1.1.6.2	Guest.....	351
7.1.1.6.3	Key Distribution Center Service Account.....	351
7.1.1.6.4	Cert Publishers.....	351
7.1.1.6.5	Domain Administrators.....	351
7.1.1.6.6	Domain Computers.....	352
7.1.1.6.7	Domain Controllers.....	352
7.1.1.6.8	Domain Guests .....	352
7.1.1.6.9	Domain Users .....	352
7.1.1.6.10	Enterprise Administrators .....	352
7.1.1.6.11	Group Policy Creator Owners .....	353
7.1.1.6.12	RAS and IAS Servers .....	353
7.1.1.6.13	Read-Only Domain Controllers .....	353
7.1.1.6.14	Enterprise Read-Only Domain Controllers .....	353
7.1.1.6.15	Schema Admins .....	353
7.1.2	Forest Invariants .....	354
7.1.2.1	DC Existence .....	354
7.1.2.2	NC Existence .....	355
7.1.2.3	Hosting Invariants.....	355
7.1.2.3.1	DC and Application NC Replica .....	355
7.1.2.3.2	DC and Regular Domain NC Replica .....	355
7.1.2.3.3	DC and Schema/Config NC Replicas .....	356
7.1.2.3.4	DC and Partial Replica NCs Replicas .....	356
7.1.3	Security Descriptor Invariants .....	356
7.1.3.1	ACE Ordering Rules .....	358
7.1.3.2	SD flags control .....	358
7.1.3.3	Processing Specifics .....	358
7.1.3.4	Security Considerations .....	359
7.1.3.5	DACL and SACL Defaulting Rules.....	360
7.1.3.6	Owner and Group Defaulting Rules .....	361
7.1.3.7	Default Administrators Group .....	361
7.1.4	Special Attributes .....	362
7.1.4.1	ntMixedDomain.....	362
7.1.4.2	msDS-Behavior-Version: DC Functional Level .....	362
7.1.4.3	msDS-Behavior-Version: Domain NC Functional Level .....	363
7.1.4.4	msDS-Behavior-Version: Forest Functional Level.....	363
7.1.4.5	Replication Schedule Structures .....	364
7.1.4.5.1	SCHEDULE_HEADER Structure .....	364
7.1.4.5.2	SCHEDULE Structure.....	364
7.1.4.5.3	REPS_FROM.....	365
7.1.4.5.4	REPS_TO.....	365
7.1.4.5.5	MTX_ADDRESS Structure .....	365
7.1.4.5.6	REPLTIMES Structure .....	365
7.1.4.5.7	PAS_DATA Structure .....	365
7.1.4.6	msDS-AuthenticatedAtDC .....	365
7.1.5	FSMO Roles .....	365
7.1.5.1	Domain Naming Master FSMO Role.....	366
7.1.5.2	RID Master FSMO Role.....	366
7.1.5.3	PDC Emulator FSMO Role .....	367

7.1.5.4	Infrastructure FSMO Role .....	367
7.1.6	Trust Objects .....	367
7.1.6.1	Overview (Synopsis) .....	367
7.1.6.2	Relationship to Other Protocols .....	368
7.1.6.2.1	TDO Replication over DRS .....	368
7.1.6.2.2	TDO Roles in Authentication Protocols over Domain Boundaries.....	368
7.1.6.2.3	TDO Roles in Authorization over Domain Boundaries.....	368
7.1.6.2.4	Roles in Name Lookup and Routing Decisions.....	368
7.1.6.3	Prerequisites/Preconditions .....	369
7.1.6.4	Versioning and Capability Negotiation .....	369
7.1.6.5	Vendor-Extensible Fields .....	369
7.1.6.6	Transport .....	369
7.1.6.7	Essential Attributes of a Trusted Domain Object.....	369
7.1.6.7.1	flatName .....	370
7.1.6.7.2	isCriticalSystemObject .....	370
7.1.6.7.3	msDs-supportedEncryptionTypes.....	370
7.1.6.7.4	msDS-TrustForestTrustInfo.....	371
7.1.6.7.5	nTSecurityDescriptor .....	371
7.1.6.7.6	objectCategory.....	372
7.1.6.7.7	objectClass .....	372
7.1.6.7.8	securityIdentifier .....	372
7.1.6.7.9	trustAttributes .....	372
7.1.6.7.10	trustAuthIncoming.....	374
7.1.6.7.11	trustAuthOutgoing.....	374
7.1.6.7.12	trustDirection.....	375
7.1.6.7.13	trustPartner.....	375
7.1.6.7.14	trustPosixOffset.....	375
7.1.6.7.15	trustType .....	375
7.1.6.8	Details.....	376
7.1.6.8.1	trustAuthInfo attributes - TDO Keys and Trusts .....	376
7.1.6.8.1.1	LSAPR_AUTH_INFORMATION.....	377
7.1.6.8.1.2	Kerberos Usages of trustAuthInfo Attributes.....	379
7.1.6.8.1.3	Netlogon Usages of trustAuth Attributes .....	379
7.1.6.8.2	msDS-TrustForestTrustInfo Attribute .....	380
7.1.6.8.2.1	Record.....	380
7.1.6.8.2.2	Building Well Formed msDS-TrustForestTrustInfo Messages .....	384
7.1.6.8.3	Computation of posixOffset.....	386
7.1.6.8.4	Timers .....	387
7.1.6.8.4.1	Trust Secret Cycling .....	387
7.1.6.8.5	Initialization.....	387
7.1.6.9	Security Considerations for Implementers .....	388
7.1.7	DynamicObject invariants .....	388
7.2	Knowledge Consistency Checker .....	389
7.2.1	References .....	389
7.2.2	Overview.....	389
7.2.2.1	Refresh kCCFailedLinks and kCCFailedConnections .....	391
7.2.2.2	Intra-site Connection Creation .....	392
7.2.2.3	Inter-site Connection Creation .....	394
7.2.2.3.1	ISTG Selection .....	395
7.2.2.3.2	Merge of kCCFailedLinks and kCCFailedLinks from Bridgeheads .....	396
7.2.2.3.3	Site Graph Concepts .....	396
7.2.2.3.4	Connection Creation .....	398
7.2.2.3.4.1	Types .....	398
7.2.2.3.4.2	Main Entry Point .....	399
7.2.2.3.4.3	Site Graph Construction.....	400

7.2.2.3.4.4	Spanning Tree Computation .....	404
7.2.2.3.4.5	nTDSConnection Creation .....	416
7.2.2.4	Removing Unnecessary Connections .....	420
7.2.2.5	Connection Translation .....	421
7.2.2.6	Remove Unneeded kCCFailedLinks and kCCFailedConnections Tuples .....	423
7.3	Publishing and Locating a Domain Controller .....	423
7.3.1	Structures and Constants .....	424
7.3.1.1	NETLOGON_NT_VERSION Options Bits.....	424
7.3.1.2	DS_FLAG Options Bits .....	425
7.3.1.3	Operation Code.....	426
7.3.1.4	NETLOGON_SAM_LOGON_REQUEST .....	426
7.3.1.5	NETLOGON_SAM_LOGON_RESPONSE_NT40 .....	428
7.3.1.6	NETLOGON_SAM_LOGON_RESPONSE .....	429
7.3.1.7	NETLOGON_SAM_LOGON_RESPONSE_EX .....	431
7.3.2	DNS Record Registrations .....	433
7.3.2.1	SRV Records Registered by DC .....	433
7.3.2.2	Non-SRV Records Registered by a DC .....	435
7.3.3	LDAP "Ping" .....	437
7.3.3.1	Syntactic Validation of the Filter.....	438
7.3.3.2	Domain Controller Response to an LDAP "Ping" .....	438
7.3.3.3	Response to Invalid Filter .....	442
7.3.4	NetBIOS Broadcast and NBNS Background .....	442
7.3.5	Mailslot "Ping" .....	443
7.3.6	Locate a Domain Controller .....	444
7.3.7	Name Compression and Decompression .....	445
7.3.8	AD/LDS DC Publication .....	447
7.4	Domain Join .....	448
7.4.1	State of a Machine Joined to a Domain .....	449
7.4.2	State in an Active Directory Domain.....	449
7.4.3	Relationship to Protocols .....	450
7.5	Unicode String Comparison .....	450
7.5.1	String Comparison by Using Sort Keys .....	450
7.5.2	Unisort.txt Data .....	451
7.5.3	GetWindowsActiveDirectorySortKey Pseudo Code .....	453
7.5.4	GetWindowsActiveDirectorySortKey .....	456
7.5.5	TestDoubleCompression .....	461
7.5.6	GetCompression .....	462
7.5.7	CorrectUnicodeWeight .....	463
7.5.8	MakeUnicodeWeight .....	463
7.5.9	GetCharacterWeights .....	464
7.5.10	GetExpansionWeights .....	465
7.5.11	GetExpandedCharacters .....	466
7.5.12	Is3To1Compression .....	466
7.5.13	Is2To1Compression .....	467
7.5.14	SpecialCaseHandler .....	468
7.5.15	GetPositionSpecialWeight .....	473
7.5.16	MapOldHangulSortKey.....	474
7.5.17	GetJamoComposition .....	476
7.5.18	GetJamoStateData.....	477
7.5.19	FindNewJamoState .....	478
7.5.20	UpdateJamoSortInfo .....	479
7.5.21	IsJamo .....	480
7.5.22	IsJamoLeading .....	480
7.5.23	IsJamoTrailing.....	481
7.5.24	InitKoreanScriptMap .....	481

<b>8</b>	<b>Index.....</b>	<b>483</b>
----------	-------------------	------------

# 1 Introduction

This is the primary specification for Active Directory, both AD/DS and AD/LDS. The state model for this specification is prerequisite to the other specifications for Active Directory: [\[MS-DRSR\]](#) and [\[MS-SRPL\]](#).

State is included in the state model for this specification only as necessitated by the requirement that a licensee implementation of Windows Server protocols be able to receive messages and respond in the same manner as a Windows Server. Behavior is specified in terms of request message received, processing based on current state, resulting state transformation, and response message sent. Unless otherwise specified in the sections that follow, all of the behaviors are required for interoperability.

Attribute names are underlined in this document, following the convention of [\[MS-DRSR\]](#).

References from this specification to top-level subsections of [\[MS-DRSR\]](#) section 5 take the form "[\[MS-DRSR\]](#) section 5:**Name**" where **Name** is the subsection name within section 5. Section 5 of [\[MS-DRSR\]](#) is arranged alphabetically, so the table of contents serves as an index.

For clarity, bit flags are sometimes shown as bit field diagrams. In the case of bit flags for **LDAP** attributes, these diagrams take on big-endian characteristics, but do not reflect the actual byte ordering of integers over the wire because LDAP transfers an integer as the UTF-8 string of the decimal representation of that integer, as specified in [\[RFC2252\]](#).

## 1.1 Glossary

The following sections contain a glossary of terms that appear in this document.

### 1.1.1 Pervasive Concepts

Use [\[KNUTH1\]](#) section 2.3.4.2 as a reference for the terms **oriented tree**, **root**, **vertex**, **arc**, **initial vertex**, and **final vertex**.

**Replica:** A variable containing a set of **objects**.

**Attribute:** An identifier for a value or set of values. See also **Attribute** in the Glossary Entries section.

**Object:** A set of **attributes**, each with its associated values. Two attributes of an **object** have special significance:

- Identifying attribute. A designated single-valued attribute appears on every object; the value of this attribute identifies the object. For the set of objects in a **replica**, the values of the identifying attribute are distinct.
- Parent-identifying attribute. A designated single-valued attribute appears on every object; the value of this attribute identifies the object's parent. That is, this attribute contains the value of the parent's identifying attribute, or a reserved value identifying no object. For the set of objects in a replica, the values of this parent-identifying attribute define an **oriented tree** with objects as vertices and child-parent references as directed **arcs** with the child as an **arc's initial vertex** and the parent as an **arc's final vertex**.

Note that an object is a value, not a variable; a replica is a variable. The process of adding, modifying, or deleting an object in a replica replaces the entire value of the replica with a new value.

As the word replica suggests, it is often the case that two replicas contain "the same objects." In this usage, objects in two replicas are considered "the same" if they have the same value of the identifying attribute and if there is a process in place (**replication**) to converge the values of the remaining attributes. See **Replication**. When the members of a set of replicas are considered to be the same, it is common to say "an object" as a shorthand way of referring to the set of corresponding objects in the replicas.

**Parent object:** See **Object**.

**Child object, Children:** An **object** that is not the **root** of its **oriented tree**. The **children** of an **object** *o* is the set of all **objects** whose **parent** is *o*.

See section [3.1.1.1.3](#) for the particular use made of these definitions in this specification.

## 1.1.2 Ordinary Glossary Entries

The following terms are defined in [\[MS-GLOS\]](#):

- Abstract Class**
- Abstract Object Class**
- Access Check**
- Access Control Entry (ACE)**
- Access Control List (ACL)**
- Access Mask**
- AttributeStamp**
- Authentication**
- Authorization**
- Back Link Value**
- Bridgehead DC**
- Broadcast**
- Checksum**
- Component Object Model (COM)**
- Constructed Attribute**
- Container**
- Control Access Right (CAR)**
- Cyclic Redundancy Check (CRC)**
- Digest**
- Discretionary Access Control List (DACL)**
- Domain**
- Downlevel Trust**
- Endpoint**
- Expunge**
- Forward Link Value**
- FSMO Role Owner**
- Full NC Replica**
- Garbage Collection**
- Global Catalog (GC)**
- Global Catalog Server (GC Server)**
- Globally Unique Identifier (GUID)**
- Group**
- GUIDString**
- Inheritance**
- LDAP**
- LDAP Connection**
- Link Attribute**

Link Value  
LinkValueStamp  
Local DC  
Lost and Found Container  
Messaging Application Programming Interface (MAPI)  
Mixed Mode  
Name Service Provider Interface (NSPI)  
Native Mode  
Non-replicated Attribute  
NULL GUID  
Object Class Inheritance  
Object of Class x (or x Object)  
Operational Attribute  
Partial Attribute Set (PAS)  
Property Set  
Remote Procedure Call (RPC)  
Replication  
Replication Latency  
Replication Traffic  
RPC Transport  
Schema  
Schema Container  
Schema Object  
Security Principal  
Security Provider  
Simple Mail Transfer Protocol (SMTP)  
SSL/TLS Handshake  
Structural Object Class  
Tombstone  
Trust Object  
Trust Secret  
Trusted Domain Object (TDO)  
Unicode  
Universally Unique Identifier (UUID)  
Uplevel Trust  
Windows Error Code

The following terms are specific to this document:

**88 Object Class:** An **object class** defined in the 1988 X.500 **directory** specification ([\[X501\]](#)).  
An **88 object class** can be instantiated as a new **object**, like a **structural object class**, and on an existing **object**, like an **auxiliary object class**.

**Account Domain:** The **SID** namespace for which a given machine is authoritative. For a **DC**, this is its default **domain**. For a Windows machine that is joined to a **domain**, this is the **SID** namespace defined by the local Security Accounts Manager.

**Account Domain SID:** A **SID** within the **account domain**.

**Active:** A state of an [attributeSchema](#) or [classSchema](#) object that represents part of the **schema**. It is possible to instantiate an **active** attribute or an **active** class. The opposite term is **defunct**.

**Active Directory:** Either **Active Directory Domain Services (AD/DS)** or **Active Directory Lightweight Directory Services (AD/LDS)**.



**Active Directory Domain Services (AD/DS):** AD/DS is an operating system **directory** service implemented by a **domain controller (DC)**. The **directory** service provides a data store for **objects** that is distributed across multiple **DCs**. The **DCs** interoperate as peers to ensure that a local change to an **object** replicates correctly across **DCs**. **AD/DS** first became available as part of Microsoft Windows 2000 and is available as part of Windows 2000 Server products and Windows Server 2003 products; in these products it is called "Active Directory". It is also available as part of Windows Server 2008. **AD/DS** is not present in Windows NT 4.0 or in Windows XP. For more information, see [\[MS-SECO\]](#) section 2.2.2.

**Active Directory Lightweight Directory Services (AD/LDS):** AD/LDS is an operating system **directory** service implemented by a **domain controller (DC)**. The most significant difference between **AD/LDS** and **AD/DS** is that **AD/LDS** does not host **domain NCs**. A server can host multiple **AD/LDS DCs**. (In Microsoft documentation, **AD/LDS** is sometimes called "ADAM".)

**Ambiguous Name Resolution (ANR):** A search algorithm that permits an **LDAP** client to search multiple naming-relating **attributes** on **objects** via a single clause of the form "(aNR=value)" in an **LDAP** search **filter**. This permits a client to query for an **object** when the client possesses some identifying material related to the **object** but does not know which **attribute** of the **object** contains that identifying material.

**Ancestor Object:** An **object** A is an ancestor of **object** O if there is a directed path of child-parent **arcs** from O to A. In other words, A is on the path from O to the root of the tree containing O.

**Application NC:** A specific type of **NC**. An **application NC** cannot contain **security principal objects**. An AD/DS **forest** can have zero or more **application NCs**, while an AD/LDS **forest** can have one or more.

**Attribute:** (A specialization of the pervasive-concepts definition in the previous section.) An identifier for a single-valued or multi-valued data element that is associated with an **LDAP directory object**. An **object** consists of its **attributes** and their values. For example, [cn](#) (common name), [street](#) (street address), and [mail](#) (e-mail addresses) can all be **attributes** of a [user object](#). An **attribute's schema**, including the **syntax** of its values, is defined in an [attributeSchema object](#).

**Attribute Syntax:** A specification of the format and range of permissible values of an **attribute**. The **syntax** of an **attribute** is defined by several attributes on the [attributeSchema object](#). The **attribute syntaxes** supported by **Active Directory** include Boolean, Enumeration, Integer, LargeInteger, String(UTC-Time), String(Unicode), and Object(DS-DN).

[attributeID:](#) An **OID**-valued identifying **attribute** of each [attributeSchema object](#) in the **schema NC**.

**ATTRTYP:** A 32-bit quantity representing an **OID**. See [\[MS-DRSR\]](#) section 5:ATTRTYP.

**Auxiliary Class:** See **Auxiliary Object Class**.

**Auxiliary Object Class:** An **object class** that can be instantiated on, or removed from, an existing **object**.

**Back Link Attribute:** A **back link attribute** is a **constructed attribute** whose values include **object** references (for example, an **attribute** of **syntax** Object(DS-DN)). The **back link values** are derived from the values of a related **attribute**, a **forward link attribute**, on other **objects**. If *f* is the **forward link attribute**, one back link value exists on **object** *o* for each **object** *r* that contains a value of *o* for **attribute** *f*. The relationship between the **forward link attributes** and **back link attributes** is expressed using the [linkID attribute](#)

on the [attributeSchema](#) **objects** representing the two **attributes**. The forward link's [linkID](#) is an even number, the back link's [linkID](#) is the forward link's [linkID](#) plus one. For more information, see section [3.1.1.1.6](#).

**Basic Encoding Rules (BER):** A specific set of rules for encoding data structures for transfer over a network. These encoding rules are defined in [ITUX690].

**Built-in Domain:** The **SID** namespace defined by the fixed **SID** S-1-5-32. Contains **groups** that define roles on a local machine such as "Backup Operators".

**Built-in Domain SID:** The fixed **SID** S-1-5-32.

**Built-in Principal:** A **security principal** within the **built-in domain**.

**Canonical Name:** A syntactic transformation of an **Active Directory DN** into something resembling a pathname that still identifies an **object** within a **forest**. The **DN** "cn=Peter Houston, ou=NTDEV, dc=microsoft, dc=com" translates to the **canonical name** "microsoft.com/NTDEV/Peter Houston", while the **DN** "dc=microsoft, dc=com" translates to the **canonical name** "microsoft.com/".

**computer Object:** An **object of class** [computer](#). A **computer object** is a **security principal object**; the **principal** is the operating system running on the computer. The shared secret allows the operating system running on the computer to authenticate itself independently of any user running on the system. See **Security Principal**.

**Configuration Naming Context (Config NC):** A specific type of **NC**, or an instance of that type. A **forest** has a single **config NC**, which contains configuration information shared among all **DCs** in the **forest**.

**Critical Object:** A subset of the **objects** in the **default NC**, identified by the **attribute** [isCriticalSystemObject](#) having the value true. The **objects** that are marked in this way are essential for the operation of a **DC** hosting the **NC**.

**crossRef Object:** An **object of class** [crossRef](#). Each **crossRef object** is a child of the **partitions container** in the **config NC**. The [crossRef](#) specifies the properties of an **NC**, such as its DNS name, operational settings, etc.

**Cross Forest Trust:** A relationship between two **forests** that enables **security principals** from any **domain** in one **forest** to authenticate to computers joined to any **domain** in the other **forest**.

**Cycle:** See **Replication Cycle**.

**DC Functional Level:** A specification of functionality available in a **DC**. Possible values are DS\_BEHAVIOR\_WIN2000 (for Windows 2000 Server **DCs**), DS\_BEHAVIOR\_WIN2003 (for Windows Server 2003 **DCs**), and DS\_BEHAVIOR\_WIN2008 (for Windows Server 2008 **DCs**).

**DC In Site x:** A **DC** such that the **Site of the DC** is x.

**Default Naming Context (Default NC):** When **Active Directory** is operating as **AD/DS**, this is the **domain NC** whose full **replica** is hosted by a **DC**. In this case the **default NC** contains the **DC's** [computer object](#). When **Active Directory** is operating as **AD/LDS**, the **default NC** is the **NC** specified by the [msDS-DefaultNamingContext](#) **attribute** on the [nTDSDSA object](#) for the **DC**. See **nTDSDSA object**.

**Defunct:** A state of an [attributeSchema](#) or [classSchema](#) object that represents part of the **schema**. It is not possible to instantiate a **defunct attribute** or a **defunct class**. The opposite term is **active**.

**Directory:** A forest.

**Directory Object (or Object):** An LDAP object [\[RFC2251\]](#), which is a specialization of **object** as defined in the Pervasive Concepts section of this glossary. An **Active Directory object** can be identified by a **dsname** according to the matching rules defined in [MS-DRSR] section 5:DSNAME.

**Distinguished Name (DN):** An LDAP distinguished name [\[RFC2251\]](#). The DN of an **object** is the DN of its parent, preceded by the RDN of the **object**. Example: "cn=Peter Houston, ou=NTDEV, dc=microsoft, dc=com".

**Domain Controller (DC):** The service, running on a server, that implements **Active Directory**; or the server hosting this service. The service hosts the data store for **objects** and interoperates with other **DCs** to ensure that a local change to an **object** replicates correctly across all **DCs**. When **Active Directory** is operating as **AD/DS**, the **DC** contains **full NC replicas** of the **config NC**, **schema NC**, and one of the **domain NCs** in its **forest**. If the **AD/DS DC** is a **GC server**, it contains **partial NC replicas** of the remaining **domain NCs** in its **forest**. For more information, see [\[MS-SECO\]](#) section 2.2.2. When **Active Directory** is operating as **AD/LDS**, several **DCs** can run on one server.

**Domain Functional Level:** A specification of functionality available in a **domain**. Must be less than or equal to the **DC functional level** of every **DC** that hosts a **replica** of the **domain's NC**. Possible values in Windows Server 2008 are DS\_BEHAVIOR\_WIN2000, DS\_BEHAVIOR\_WIN2003\_WITH\_MIXED\_DOMAINS, DS\_BEHAVIOR\_WIN2003, and DS\_BEHAVIOR\_WIN2008.

**Domain Joined:** A relationship between a machine and some **domain NC** in which they share a secret. The shared secret allows the machine to authenticate to a **DC** for the **domain**.

**Domain Local Group:** An **Active Directory group** that allows [user objects](#), **global groups**, and **universal groups** from any **domain** as members. It also allows other **domain local groups** from within its **domain** as members. A **group object g** is a **domain local group** if and only if GROUP\_TYPE\_RESOURCE\_GROUP is present in g!groupType. A **security-enabled domain local group** is valid for inclusion within **ACLs** from its own **domain**. If a **domain** is in **mixed mode**, then a **security-enabled domain local group** in that **domain** allows only [user objects](#) as members.

**Domain Naming Context (Domain NC):** A specific type of **NC**, or an instance of that type. A **domain NC** can contain **security principal objects**; no other type of **NC** can. **Domain NCs** appear in the **GC**. A **forest** has one or more **domain NCs**. The root of a **domain NC** is an **object of class domainDNS**.

**Domain Prefix:** A **domain SID**, minus the **RID** portion.

[domainDNS](#): A specific **object class**. The root of a **domain NC** or an **AD/DS application NC** is an **object of class domainDNS**. The **DN** of such an **object** takes the form:

$$dc=n_1, dc=n_2, \dots dc=n_k$$

where each  $n_i$  satisfies the syntactic requirements of a DNS name component. For more information, see [\[RFC1034\]](#). Such a **DN** corresponds to the DNS name:

$$n_1. n_2. \dots .n_k$$

This is the DNS name of the **NC**, and allows **replicas** of the **NC** to be located using DNS.

**DSA Object:** See **nTDSDSA Object**.

**DSA GUID:** The [objectGUID](#) of a **DSA object**.

**dsname:** A tuple that contains between one and three identifiers for an **object**. The possible identifiers are the **object's GUID** (attribute [objectGUID](#)), **SID** (attribute [objectSid](#)), and **DN** (attribute [distinguishedName](#)). A **dsname** can appear in a protocol message and as an **attribute** value (for example, a value of an **attribute** with **syntax** Object(DS-DN)).

**Dynamic Object:** An **object** with a time-to-die, attribute [msDS-Entry-Time-To-Die](#). The **directory** service garbage-collects a **dynamic object** immediately after its time-to-die has passed. The **constructed attribute** [entryTTL](#) gives a **dynamic object's** current time-to-live, that is, [msDS-Entry-Time-To-Die](#) minus the current system time. For more information, see [\[RFC2589\]](#).

**Entry:** A synonym for **object**. See also **Object** in the Pervasive Concepts section.

**Extended-Rights Container:** A container holding objects that correspond to **control access rights**. The container is a child of **config NC** and has **RDN** CN=Extended-Rights.

**File Replication Service (FRS):** One of the services offered by a **DC**. The running/paused state of **FRS** on a **DC** is available through protocols documented in section [7.3](#).

**Filter:** One of the parameters in an **LDAP** search request. The **filter** specifies matching constraints for the candidate **objects**.

**Filtered Attribute Set:** The subset of attributes that are not replicated to the **filtered partial NC replica** and the **filtered GC partial NC replica**. The **filtered attribute set** is part of the state of the **forest**, and used to control the attributes that replicate to a **Read-Only Domain Controller (RODC)**. The [searchFlags](#) schema attribute is used to define this set.

**Filtered GC Partial NC Replica:** An **NC replica** that contains a **schema**-specified subset of attributes for the **objects**. The attributes consists of the attributes in the **GC partial attribute set** excluding those present in the **filtered attribute set**. A **filtered GC partial NC replica** is not writable.

**Filtered Partial NC Replica:** An **NC replica** that contains all the attributes of the **objects** excluding those attributes in the **filtered attribute set**. A **filtered partial NC replica** is not writable.

**Flexible Single Master Operation (FSMO):** A read or **update** operation on an **NC**, such that the operation must be performed on the single designated "master" **replica** of that **NC**. The master **replica** designation is "flexible" because it can be changed without losing the consistency gained from having a single master. This term, pronounced "fizmo", is never used alone; see **FSMO Role**, **FSMO Role Owner**.

**Foreign Principal Object (FPO):** A [foreignSecurityPrincipal](#) object.

**Forest:** For **AD/DS**, a set of NCs consisting of one **schema NC**, one **config NC**, one or more **domain NCs**, and zero or more **application NCs**. Because a set of NCs can be arranged into a tree structure, a **forest** is also a set containing one or several trees of NCs. For **AD/LDS**, a set of NCs consisting of one **schema NC**, one **config NC**, and zero or more **application NCs**. (In Microsoft documentation, an **AD/LDS forest** is called a "configuration set".)

**Forest Functional Level:** A specification of functionality available in a **forest**. Must be less than or equal to the **DC functional level** of every **DC** in the **forest**. Possible values in Windows Server 2008 are DS\_BEHAVIOR\_WIN2000, DS\_BEHAVIOR\_WIN2003\_WITH\_MIXED\_DOMAINS, DS\_BEHAVIOR\_WIN2003, and DS\_BEHAVIOR\_WIN2008.

**Forest Root Domain NC:** The **domain NC** within a **forest** whose child is the **forest's config NC**. The DNS name of this **domain** serves as the **forest** name.

**Forward Link Attribute:** A type of **attribute** whose values include **object references** (for example, an **attribute** of **syntax** Object(DS-DN)). The **forward link values** can be used to compute the values of a related **attribute**, a **back link attribute**, on other **objects**. A **forward link attribute** can exist with no corresponding **back link attribute**, but not vice versa.

**FSMO Role:** A set of **objects** that can be **updated** in only one **NC replica** (the **FSMO role owner's replica**) at any given time.

**FSMO Role Object:** The **object** in the **directory** that represents a specific **FSMO Role**. This **object** is an element of the **FSMO Role** and contains the [fSMORoleOwner](#) attribute.

**FSMO Role Transfer:** A request to a **DC** d. If d is the current Owner of the specified **FSMO Role**, the effect is to transfer that role to the client; if d is not the current Owner of the role, the effect is to **update** the client's role **objects** from d's **replica**, so the client can try the request again on another **DC**.

**GC Partial Attribute Set (PAS):** The subset of attributes that replicate to a **GC partial NC replica**. The **partial attribute set** is part of the state of the **forest**, and is used to control the attributes that replicate to **GC servers**. The [isMemberOfPartialAttributeSet](#) schema attribute is used to define this set.

**GC Partial NC Replica:** An **NC replica** that contains a **schema**-specified subset of attributes for the **objects** it contains. The subset of attributes consists of the attributes in the **GC partial attribute set**.

**Global Group:** An **Active Directory group** that allows [user](#) **objects** from its own **domain** and **global groups** from its own **domain** as members. **Universal groups** can contain **global groups**. A **group object** g is a **global group** if and only if GROUP\_TYPE\_ACCOUNT\_GROUP is present in g![groupType](#). A **security-enabled global group** is valid for inclusion within ACLs anywhere in the **forest**. If a **domain** is in **mixed mode**, then a **security-enabled global group** in that **domain** allows only [user](#) **objects** as members. See **Domain Local Group**, **Security-Enabled Group**.

[governsID](#): An **OID**-valued identifying **attribute** of each [classSchema](#) **object** in the **schema NC**.

[group](#) **Object:** An **object** of class [group](#) representing a **group**. A [group](#) has a **forward link attribute** [member](#); the values of this **attribute** either represent elements of the **group** (for example, **objects** of class [user](#) or [computer](#)) or represent subsets of the **group** (**objects** of class [group](#)). The **back link attribute** [memberOf](#) enables navigation from **group** members to the **groups** containing them. Some **groups** represent **groups** of **security principals** and some do not (and are, for instance, used to represent e-mail distribution lists).

**GUID-based DNS name:** A DNS name published for a **DC**. If a **DC's DSA GUID** is "52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8", and the **forest root domain NC's** DNS name is "fabrikam.com", then the **GUID-based DNS name** of the **DC** is "52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8.\_msdcs.fabrikam.com". See **domainDNS**.

**Inbound Trust:** A **trust** relationship between two **domains**, from the perspective of the **domain** that is trusted to perform **authentication**.

**Inter-site Topology Generator (ISTG):** A **DC** within a given **Site** that computes an **NC replica graph** for each **NC replica** on any **DC** in its **Site**. This **DC** creates, **updates**, and deletes corresponding [nTDSConnection](#) **objects** for edges directed from **NC replicas** in other **sites** to **NC replicas** in its **Site**.

[invocationId](#): An attribute of an [nTDSDSA](#) **object**. Its value is a unique identifier for a function that maps from **USNs** to **updates** to a **DC's NC replicas**. See **nTDSDSA Object**.

**Knowledge Consistency Checker (KCC):** An internal Windows component of the **Active Directory replication** used to create spanning trees for **DC-to-DC replication** and to translate those trees into settings of variables that implement the **replication** topology.

**LDAP Ping:** A specific **LDAP** search that returns information about the liveness of services on a **DC**.

**Lingering Object:** An **object** that still exists in an **NC replica** even though it has been deleted and garbage-collected from other **replicas**. This occurs, for instance, when a **DC** goes offline for longer than the **tombstone lifetime**.

**Mailslot:** A form of datagram communication using the SMB protocol, as specified in [\[MS-MAIL\]](#).

**Mailslot Ping:** A specific Mailslot request that returns information about the liveness of services on a **DC**.

**Most Specific Object Class:** In a sequence of **object classes** related by **inheritance**, the class that none of the other classes inherits from. The special **object class** [top](#) is less specific than any other class.

**Naming Context (NC):** An **NC** is a **dsname**, containing at least a **DN** and a **GUID**, used in forming names for a tree of **objects**. The **DN** of the **dsname** is the [distinguishedName](#) **attribute** of the tree root. The **GUID** of the **dsname** is the [objectGUID](#) **attribute** of the tree root. The **SID** of the **dsname**, if present, is the [objectSid](#) **attribute** of the tree root; for AD/DS, the **SID** is present if and only if the **NC** is a **domain NC**. **Active Directory** supports organizing several NCs into a tree structure.

**NC Replica:** A variable containing a tree of **objects** whose root **object** is identified by some **NC**.

**NC Replica Graph:** A directed graph containing **NC replicas** as nodes and [repsFrom](#) tuples as inbound edges by which **originating updates** replicate from each full **replica** of a given **NC** to all other **NC replicas** of the **NC**, directly or transitively.

**NetBIOS:** A protocol family including name resolution, datagram, and connection services. For more information, see [\[RFC1001\]](#) and [\[RFC1002\]](#).

**NetBIOS Name Service (NBNS):** The name service for **NetBIOS**. For more information, see [\[RFC1001\]](#) and [\[RFC1002\]](#).

**Netlogon:** A component of Windows that authenticates a computer and provides other services. The running/paused state of **Netlogon** on a **DC** is available through protocols documented in section [7.3](#).

[nTDSDSA Object](#): An **object of class** [nTDSDSA](#), representing a **DC** in the **config NC**.



**Object Class:** A set of restrictions on the construction and **update** of **objects**. An **object class** must be specified when creating an **object**. An **object class** specifies a set of must-have attributes (every **object** of the class must have at least one value of each) and may-have attributes (every **object** of the class may have a value of each). An **object class** also specifies a set of possible superiors (the **parent object** of an **object** of the class must have one of these classes). An **object class** is defined by a [classSchema](#) **object**.

**Object Class Name:** The [LDAPDisplayName](#) of the [classSchema](#) **object** of an **object class**. This document consistently uses **object class names** to denote **object classes**; for example, [user](#) and [group](#) both name **object classes**. The correspondence between **LDAP** display names and numeric **OIDs** in the **Active Directory schema** is defined in the appendices of this document: [\[MS-ADSC\]](#), [\[MS-ADA1\]](#), [\[MS-ADA2\]](#), and [\[MS-ADA3\]](#).

**Object ID:** See **Object Identifier (OID)**.

**Object Identifier (OID):** A sequence of numbers in a format defined by [\[RFC1778\]](#). See **attributeID** and **governsID**.

**Object Reference:** An **attribute** value that references an **object**; reading a reference gives the **DN** or full **dsname** of the **object**.

[objectClass](#): The **attribute** on an **object** that holds the identity of each **object class** of the **object**.

[objectGUID](#): The **attribute** on an **object** whose value is a **GUID** that uniquely identifies the **object**. The value of [objectGUID](#) is assigned when an **object** is created and is immutable thereafter. The integrity of both **object references** between NCs and of **replication** depends on the integrity of the [objectGUID](#) **attribute**.

[objectSid](#): The **attribute** on an **object** whose value is a **SID** that identifies the **object** as a **security principal object**. The value of [objectSid](#) is assigned when a **security principal object** is created and is immutable thereafter unless the **object** moves to another **domain**. The integrity of **authentication** depends on the integrity of the [objectSid](#) **attribute**.

**Oriented Tree:** A directed acyclic graph such that for every **vertex** v, except one (the root), there is a unique **arc** whose **initial vertex** is v. There is no **arc** whose **initial vertex** is the root. For more information, see [KNUTH1] section 2.3.4.2.

**Originating Update:** An **update** performed to an **NC replica** directly by a client, as opposed to an **update** applied by **replication** from another **NC replica**. An **originating update** to an **attribute** or **link value** generates a new **stamp** for the **attribute** or **link value**.

**Outbound Trust:** A **trust** relationship between two **domains**, from the perspective of the **domain** that trusts another **domain** to perform **authentication**.

**Partial NC Replica:** An **NC replica** that contains a **schema**-specified subset of attributes for the **objects** it contains. A partial **replica** is not writable - it does not accept **originating updates**. See **Writable NC Replica**.

**Partitions Container:** A **child object** of the **config NC** root. The **RDN** of the **partitions container** is "cn=Partitions" and its **class** is [crossRefContainer](#). See **crossRef Object**.

**Prefix Table:** A data structure used to translate between an **OID** and an **ATTRTYP**.

**Primary Domain Controller (PDC):** A **DC** designated to track changes made to the accounts of all computers on a **domain**. It is the only computer to receive these changes directly and is

specialized so as to ensure consistency and to eliminate the potential for conflicting entries in the **Active Directory** database. A **domain** has only one **PDC**.

**Primary Group:** The **group object** identified by the **primaryGroupID attribute** of a **user object**. The **primary group's objectSid** equals the user's **objectSid**, with its **RID** portion replaced by the **primaryGroupID** value. The user is considered a member of its **primary group**.

**Principal:** A unique entity identifiable by a **SID** that is typically the requestor of access to securable **objects** or resources. It often corresponds to a human user, but can also be a computer or service. It is sometimes referred to as a **Security Principal**.

**Privilege:** The right of a user to perform system-related operations, such as debugging the system. A user's **security context** specifies what **privileges** are held by that user.

**RDN Attribute:** The **attribute** used in an **RDN**. In the **RDN "cn=Peter Houston"** the **RDN attribute** is **cn**. In **Active Directory**, the **RDN attribute** of an **object** is determined by the most specific **structural object class** of the **object**. See **Most Specific Object Class**.

**Read Permission: Authorization** to read an **attribute** of an **object**.

**Read-Only Domain Controller (RODC):** A DC that does not accept originating updates. Additionally, an RODC does not perform outbound replication.

**Read-Only Full NC Replica:** An NC replica that contains all attributes of the objects it contains, and does not accept **originating updates**.

**Relative Distinguished Name (RDN):** The name of an **object** relative to its parent. This is the leftmost attribute-value pair in the **DN** of an **object**. For example, in the **DN "cn=Peter Houston, ou=NTDEV, dc=microsoft, dc=com"**, the **RDN** is "cn=Peter Houston". For more information, see [\[RFC2251\]](#).

**Relative Identifier (RID):** The last item in the series of sub-authority values in a **SID** (see [\[MS-DTYP\]](#) section 2.3). Differences in the **RID** are what distinguish the different SIDs generated within a **domain**.

**Replicated Attribute:** An **attribute** whose values are replicated to other **NC replicas**. An **attribute** is replicated if its **attributeSchema object o** does not have a value for the **systemFlags attribute** or if the FLAG\_ATTR\_NOT\_REPLICATED bit (bit 0) of **o's systemFlags** is zero.

**Replicated Update:** An **update** performed to an **NC replica** by the **replication** system, to propagate the effect of an originating write at another **NC replica**. The **stamp** assigned during the originating write to an **attribute** or a **link value** is preserved by **replication**.

**Replication Cycle:** A series of one or more **replication** responses associated with the same **invocation ID**, concluding with the return of a new up to date vector.

**Root Domain:** The unique **domain NC** of an **Active Directory forest** that is the parent of the **forest's config NC**. The **config NC's RDN** is "cn=Configuration" relative to the root **object** of the **root domain**.

**Root DSE (rootDSE):** A nameless **entry** containing the configuration status of the **LDAP** server. Typically access to at least a portion of the **root DSE** is available to unauthenticated clients, allowing them to determine the **authentication** methods supported by the server.



**Schema NC:** A specific type of **NC**, or an instance of that type. A **forest** has a single **schema NC**, which is replicated to each **DC** in the **forest**. Each **attribute** and class in the **forest's schema** is represented as a corresponding **object** in the **forest's schema NC**.

**Secure Sockets Layer (SSL):** A means of providing privacy and data protection between a client and a server. It may also be used to provide **authentication** between the two systems. For more information, see [\[SSL3\]](#).

**Security Context:** A data structure containing **authorization** information for a particular **security principal** in the form of a collection of SIDs. One **SID** identifies the **principal** specifically, whereas others may represent other capabilities. A server uses the **authorization** information in a **security context** to check access to requested resources.

**Security Descriptor (SD):** A data structure containing the security information associated with a securable **object**. A **Security Descriptor** identifies an **object's** owner by **SID**. If access control is configured for the **object**, its **Security Descriptor** contains a **discretionary access control list (DACL)** with SIDs for the **security principals** who are allowed or denied access. The **Security Descriptor** format is specified in [\[MS-DTYP\]](#) section 2.4.6; a string representation of **security descriptors**, called SDDL, is specified in [\[MS-DTYP\]](#) section 2.5.1.

**Security-Enabled Group:** A [group object](#) with GROUP\_TYPE\_SECURITY\_ENABLED present in its [groupType](#) attribute. Only security-enabled groups are added to a **security context**. See **group Object**.

**Security Identifier (SID):** An account identifier (in Microsoft Windows, this is used to identify an account). Conceptually, the **SID** is composed of an account authority portion (typically a **domain**) and a smaller integer representing an identity relative to the account authority, termed the **RID**. The **SID** format is specified in [\[MS-DTYP\]](#) section 2.4.2; a string representation of SIDs is specified in [\[MS-DTYP\]](#) section 2.4.2 and [\[MS-SECO\]](#) section 2.1.2.

**Security Principal Object:** An **object** that corresponds to a **security principal**. A **security principal object** contains an identifier, used by the system and applications to name the **principal**, and a secret shared only by the **principal**. In **Active Directory**, a **security principal object** is identified by the [objectSid](#) attribute. In **Active Directory**, the [domainDNS](#), [user](#), [computer](#), and [group](#) **object classes** are examples of **security principal object** classes (though not every [group object](#) is a **security principal object**). In **AD/LDS**, any **object** containing the [msDS-BindableObject](#) **auxiliary class** is a **security principal**. See **domainDNS**, **objectSid**, **computer Object**, **group Object**, **user Object**.

**server Object:** A class of **object** in the **config NC**. A **server object** can have an [nTDSDSA object](#) as a child. See **nTDSDSA Object**.

**Service Principal Name (SPN):** A name a client uses to identify a service for mutual **authentication**. For more information, see [\[RFC1964\]](#) section 2.1.1.

**Simple Authentication and Security Layer (SASL):** An **authentication** mechanism that is used by **LDAP**, and is defined in [\[RFC2222\]](#).

**Site:** A collection of one or more well-connected (reliable and fast) TCP/IP subnets. By defining **sites** (represented by [site objects](#)) an administrator can optimize both **Active Directory** access and **Active Directory replication** with respect to the physical network. When users log in, **Active Directory** clients find **DCs** that are in the same **Site** as the user, or near the same **Site** if there is no **DC** in the **Site**. See **Knowledge Consistency Checker (KCC)**.

**site Object:** An **object of class** [site](#), representing a **site**.

**Site of DC:** The [site](#) object that is an ancestor of the DC's [nTDSDSA](#) object. See **nTDSDSA Object**.

**Site Settings Object:** For a given **site** with [site](#) object *s*, its **site settings object** *o* is the child of *s* such that *o* is of class [nTDSsiteSettings](#) and the **RDN** of *o* is CN=NTDS Site Settings. See **site Object**.

**SRV Record:** A type of information record in DNS that maps the name of a service to the DNS name of a server that offers that service. **DCs** advertise their capabilities by publishing **SRV records** in DNS.

**Stamp:** Information describing an **originating update** by a **DC**. The **stamp** is not the new data value; the **stamp** is information about the **update** that created the new data value. A **stamp** is often called metadata, because it is additional information that "talks about" the conventional data values.

**Syntax:** See **Attribute Syntax**.

**Tombstone Lifetime:** The amount of time that a **tombstone** remains in storage before being permanently deleted.

**Top Level Name (TLN):** The DNS name of the **forest root domain NC**.

**Transport Layer Security (TLS):** The successor to **SSL**. As with **SSL**, it provides privacy, data protection, and optionally **authentication** between a client and server. See [\[RFC2246\]](#).

**Trust:** A relationship between two **domains**. If **domain A** trusts **domain B**, **domain A** accepts **domain B's authentication** and **authorization** statements for **principals** represented by **security principal objects** in **domain B**.

**Universal Group:** An **Active Directory group** that allows [user](#) objects, **global groups**, and **universal groups** from anywhere in the **forest** as members. A **group object** *g* is a **universal group** if and only if GROUP\_TYPE\_UNIVERSAL\_GROUP is present in *g*'s [groupType](#). A **security-enabled universal group** is valid for inclusion within ACLs anywhere in the **forest**. If a **domain** is in **mixed mode**, then a **universal group** cannot be created in that **domain**. See **Domain Local Group**, **Security-enabled Group**.

**Update:** An add, modify, or delete of one or more **objects** or **attribute** values. See **originating update**, **Replicated Update**.

**Update Sequence Number (USN):** A monotonically increasing sequence number used in assigning a **stamp** to an **originating update**. See **Invocation ID**.

[userAccountControl](#): An attribute of a **security principal** object, containing a set of security options. For instance, this attribute designates the role of a computer in a **domain**.

[user](#) Object: An **object of class user**. A [user](#) object is a **security principal object**; the **principal** is a person or service entity running on the computer. The shared secret allows the person or service entity to authenticate itself.

**UTF-8:** An 8-bit, variable-width encoding of **Unicode** characters.

**UTF-16:** A 16-bit, variable-width encoding of **Unicode** characters.

**Well-Known Object:** An **object** within an **NC** that can be located using a fixed **GUID**.

**Windows Security Descriptor:** See **Security Descriptor**.

**Writable NC Replica:** An **NC replica** that accepts **originating updates**. A **writable NC replica** is always **full**, but a **full NC replica** is not always **writable**. See **Read-Only Full NC Replica**.

**MAY, SHOULD, MUST, SHOULD NOT, MUST NOT:** These terms (in all caps) are used as described in [\[RFC2119\]](#). All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

## 1.2 References

### 1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact [dochelp@microsoft.com](mailto:dochelp@microsoft.com). We will assist you in finding the relevant information. Please check the archive site, <http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624>, as an additional source.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, <http://www.opengroup.org/public/pubs/catalog/c706.htm>

[IEEE1003.1] The Open Group, "IEEE Std 1003.1, 2004 Edition", 2004, [http://www.unix.org/version3/ieee\\_std.html](http://www.unix.org/version3/ieee_std.html)

[ISO-8601] International Organization for Standardization, "Data Elements and Interchange Formats - Information Interchange - Representation of Dates and Times", ISO 8601:2004, December 2004, <http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40874&ICS1=1&ICS2=140&ICS3=30>

**Note** There is a charge to download the specification.

[ISO-9899] International Organization for Standardization, "Programming Languages - C", ISO/IEC 9899:TC2, May 2005, <http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf>

[ITU-X501-1988] "The Directory: Models", CCITT Recommendation X.501 ISO/IEC JTC 1/SC21, International Standard 9594-2, 1988.

If you have any trouble finding [ITU-X501-1988], please check [here](#).

[ITUX680] ITU-T, "Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", Recommendation X.680, July 2002, <http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf>

[ITUX690] ITU-T, "ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", Recommendation X.690, July 2002, <http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf>

[MS-ADA1] Microsoft Corporation, "[Active Directory Schema Attributes A-L](#)", July 2006.

[MS-ADA2] Microsoft Corporation, "[Active Directory Schema Attributes M](#)", July 2006.

[MS-ADA3] Microsoft Corporation, "[Active Directory Schema Attributes N-Z](#)", July 2006.

[MS-ADLS] Microsoft Corporation, "[Active Directory Lightweight Directory Services Schema](#)", June 2007.

[MS-ADSC] Microsoft Corporation, "[Active Directory Schema Classes](#)", July 2006.

[MS-ASRT] Microsoft Corporation, "Active Directory Sorting Weight Table", June 2006. Available upon request.

[MS-DRSR] Microsoft Corporation, "[Directory Replication Service \(DRS\) Remote Protocol Specification](#)", July 2006.

[MS-DTYP] Microsoft Corporation, "[Windows Data Types](#)", March 2007.

[MS-ERREF] Microsoft Corporation, "[Windows Error Codes](#)", January 2007.

[MS-GLOS] Microsoft Corporation, "[Windows Protocols Master Glossary](#)", March 2007.

[MS-KILE] Microsoft Corporation, "[Kerberos Protocol Extensions](#)", July 2006.

[MS-LSAD] Microsoft Corporation, "[Local Security Authority \(Domain Policy\) Remote Protocol Specification](#)", July 2006.

[MS-MAIL] Microsoft Corporation, "[Remote Mailslot Protocol Specification](#)", July 2006.

[MS-NLMP] Microsoft Corporation, "[NT LAN Manager \(NTLM\) Authentication Protocol Specification](#)", July 2006.

[MS-NRPC] Microsoft Corporation, "[Netlogon Remote Protocol Specification](#)", July 2006.

[MS-PAC] Microsoft Corporation, "[Privilege Attribute Certificate Data Structure](#)", July 2006.

[MS-RPCE] Microsoft Corporation, "[Remote Procedure Call Protocol Extensions](#)", July 2006.

[MS-SAMR] Microsoft Corporation, "[Security Account Manager \(SAM\) Remote Protocol Specification \(Client-to-Server\)](#)", July 2006.

[MS-SECO] Microsoft Corporation, "[Windows Security Overview](#)", December 2006.

[MS-SFU] Microsoft Corporation, "[Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol Specification](#)", July 2006.

[MS-SPNG] Microsoft Corporation, "[Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism \(SPNEGO\) Protocol Extensions](#)", July 2006.

[MS-SRPL] Microsoft Corporation, "[Directory Replication Service \(DRS\) Protocol Extensions for SMTP](#)", July 2006.

[MS-W32T] Microsoft Corporation, "[W32Time Remote Protocol Specification](#)", July 2006.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981, <http://www.ietf.org/rfc/rfc791.txt>

[RFC1001] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", RFC 1001, March 1987, <http://www.ietf.org/rfc/rfc1001.txt>

[RFC1002] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications", RFC 1002, March 1987, <http://www.ietf.org/rfc/rfc1002.txt>

[RFC1034] Mockapetris, P., "Domain Names–Concepts and Facilities", RFC 1034, November 1987, <http://www.ietf.org/rfc/rfc1034.txt>

[RFC1088] McLaughlin III, L., "A Standard for the Transmission of IP Datagrams over NetBIOS Networks", RFC 1088, February 1989, <http://www.ietf.org/rfc/rfc1088.txt>

[RFC1166] Kirkpatrick, S., Stahl, M., Recker, M., "Internet Numbers", RFC 1166, July 1990, <http://www.ietf.org/rfc/rfc1166.txt>

[RFC1274] Barker, P. and Kille, S., "The COSINE and Internet X.500 Schema", RFC 1274, November 1991, <http://www.ietf.org/rfc/rfc1274.txt>

[RFC1278] Hardcastle-Kille, S. E., "A string encoding of Presentation Address", RFC 1278, November 1991, <http://www.ietf.org/rfc/rfc1278.txt>

[RFC1777] Yeong, W., Howes, T., and Kille, S., "Lightweight Directory Access Protocol", RFC 1777, March 1995, <http://www.ietf.org/rfc/rfc1777.txt>

[RFC1778] Howes, T., Kille, S., Yeong, W., and Robbins, C., "The String Representation of Standard Attribute Syntaxes", RFC 1778, March 1995, <http://www.ietf.org/rfc/rfc1778.txt>

[RFC1798] Young, A., "Connection-less Lightweight X.500 Directory Access Protocol" RFC 1798, June 1995, <http://www.ietf.org/rfc/rfc1798.txt>

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996, <http://www.ietf.org/rfc/rfc1964.txt>

[RFC2052] Gulbrandsen, A. and P. Vixie, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2052, October 1996, <http://www.ietf.org/rfc/rfc2052.txt>

[RFC2078] Linn, J., "Generic Security Service Application Program Interface, Version 2", RFC 2078, January 1997, <http://www.ietf.org/rfc/rfc2078.txt>

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>

[RFC2222] Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997, <http://www.ietf.org/rfc/rfc2222.txt>

[RFC2246] Dierks, T. and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, <http://www.ietf.org/rfc/rfc2246.txt>

[RFC2247] Kille, S., Wahl, M., Grimstad, A., Huber, R., and Sataluri, S., "Using Domains in LDAP/X.500 Distinguished Names", RFC 2247, January 1998, <http://www.ietf.org/rfc/rfc2247.txt>

[RFC2251] Wahl, M., Howes, T., and Kille, S., "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997, <http://www.ietf.org/rfc/rfc2251.txt>

[RFC2252] Wahl, M., Coulbeck, A., Howes, T., and Kille, S., "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997, <http://www.ietf.org/rfc/rfc2252.txt>

[RFC2253] Wahl, M., Kille, S., and Howe, T., "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997, <http://www.ietf.org/rfc/rfc2253.txt>

[RFC2254] Howes, T., "The String Representation of LDAP Search Filters", RFC 2254, December 1997, <http://www.ietf.org/rfc/rfc2254.txt>

[RFC2255] Howes, T. and Smith, M., "The LDAP URL Format", RFC 2255, December 1997, <http://www.ietf.org/rfc/rfc2255.txt>

[RFC2256] Wahl, M., "A Summary of the X.500(96) User Schema for use with LDAPv3", RFC 2256, December 1997, <http://www.ietf.org/rfc/rfc2256.txt>

[RFC2307] Howard, L., "An Approach for Using LDAP as a Network Information Service", RFC 2307, March 1998, <http://www.ietf.org/rfc/rfc2307.txt>

[RFC2373] Hinden, R. and Deering, S., "IP Version 6 Addressing Architecture", RFC 2373, July 1998, <http://www.ietf.org/rfc/rfc2373.txt>

[RFC2589] Yaacovi, Y., Wahl, M., and Genovese, T., "Lightweight Directory Access Protocol (v3): Extensions for Dynamic Directory Services", RFC 2589, May 1999, <http://www.ietf.org/rfc/rfc2589.txt>

[RFC2696] Weider, C., Herron, A., Anantha, A., and Howes, T., "LDAP Control Extension for Simple Paged Results Manipulation", RFC 2696, September 1999, <http://www.ietf.org/rfc/rfc2696.txt>

[RFC2782] Gulbrandsen, A., Vixie, P., and Esibov, L., "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000, <http://www.ietf.org/rfc/rfc2782.txt>

[RFC2798] Smith, M., "Definition of the inetOrgPerson LDAP Object Class", RFC 2798, April 2000, <http://www.ietf.org/rfc/rfc2798.txt>

[RFC2829] Wahl, M., Alvestrand, H., Hodges, J., and Morgan, R., "Authentication Methods for LDAP", RFC 2829, May 2000, <http://www.ietf.org/rfc/rfc2829.txt>

[RFC2830] Hodges, J., Morgan, R., and Wahl, M., "Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security", RFC 2830, May 2000, <http://www.ietf.org/rfc/rfc2830.txt>

[RFC2831] Leach, P. and Newman, C., "Using Digest Authentication as a SASL Mechanism", RFC 2831, May 2000, <http://www.ietf.org/rfc/rfc2831.txt>

[RFC2849] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical Specification", RFC 2849, June 2000, <http://www.ietf.org/rfc/rfc2849.txt>

[RFC2891] Howes, T., Wahl, M., and Anantha, A., "LDAP Control Extension for Server Side Sorting of Search Results", RFC 2891, August 2000, <http://www.ietf.org/rfc/rfc2891.txt>

[RFC3377] Hodges, J. and Morgan, R., "Lightweight Directory Access Protocol (v3): Technical Specification", RFC 3377, September 2002, <http://www.ietf.org/rfc/rfc3377.txt>

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005, <http://www.ietf.org/rfc/rfc3961.txt>

[RFC4120] Neuman, C., Yu, T., Hartman, S., and Raeburn, K., "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005, <http://www.ietf.org/rfc/rfc4120.txt>

[RFC4122] Leach, P., Mealling, M., and Salz, R., "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, July 2005, <http://www.ietf.org/rfc/rfc4122.txt>

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and Ingersoll, W., "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005, <http://www.ietf.org/rfc/rfc4178.txt>

[RFC4370] Weltman, R., "Lightweight Directory Access Protocol (LDAP) Proxied Authorization Control", RFC 4370, February 2006, <http://www.ietf.org/rfc/rfc4370.txt>

[RFC4532] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP) "Who Am I?" Operation", RFC 4532, June 2006, <http://www.ietf.org/rfc/rfc4532.txt>

[RFC4757] Jaganathan, K., Zhu, L., and Brezak, J., "The RC4-HMAC Kerberos Encryption Types Used by Microsoft Windows", RFC 4757, December 2006, <http://www.ietf.org/rfc/rfc4757.txt>



[SSL3] Netscape, "SSL 3.0 Specification", <http://wp.netscape.com/eng/ssl3/>

If you have any trouble finding [SSL3], please check [here](#).

[X501] ITU-T, "Information Technology - Open Systems Interconnection - The Directory: The Models", Recommendation X.501, November 1993, <http://www.itu.int/rec/T-REC-X.501-199311-S/en>

[XMLSCHEMA2/2] Biron, P.V., Ed. and Malhotra, A., Ed., "XML Schema Part 2: Datatypes Second Edition", W3C Recommendation, October 2004, <http://www.w3.org/TR/xmlschema-2>

### 1.2.2 Informative References

[ADDLG] Microsoft Corporation, "Security Briefs: Credentials and Delegation", September 2005, <http://msdn.microsoft.com/msdnmag/issues/05/09/SecurityBriefs/default.aspx>

[GRAY] Gray, G. and A. Reuter, "Transaction Processing: Concepts and Techniques", San Mateo, CA: Morgan Kaufmann Publishers, 1993, ISBN: 1558601902.

[KNUTH1] Knuth, D., "The Art of Computer Programming: Volume 1/Fundamental Algorithms (Second Edition)", Reading, MA: Addison-Wesley, 1973, ASIN: B000NV8YOA.

[LISP15] McCarthy, J., Abrahams, P., Edwards, D., Hart, T., and Levin, M., "LISP 1.5 Programmers Manual", Cambridge, MA: The M.I.T. Press, 1965, ISBN-10: 0262130114.

[MS-SYS] Microsoft Corporation, "[Windows System Overview](#)", March 2007.

[MSDN-POSIX-OFFSET] Microsoft Corporation, "Mapping Posix Identifiers", <http://msdn2.microsoft.com/en-us/library/ms721870.aspx>

[MSDN-SDSTRING] Microsoft Corporation, "String(NT-Sec-Desc)", <http://msdn2.microsoft.com/en-us/library/ms684438.aspx>

[MSDN-TDI] Microsoft Corporation, "TRUSTED\_DOMAIN\_INFORMATION\_EX", <http://msdn2.microsoft.com/en-us/library/ms722477.aspx>

[VLVDRAFT] Boreham, D., Sermersheim, J., and Kashi, A., "LDAP Extensions for Scrolling View Browsing of Search Results", draft-ietf-ldapext-ldapv3-vlv-09, November 2002, <http://www3.ietf.org/proceedings/03mar/I-D/draft-ietf-ldapext-ldapv3-vlv-09.txt>

### 1.3 Overview (Synopsis)

This is the primary specification for Active Directory. The state model for this specification is prerequisite to the other specifications for Active Directory: [\[MS-DRSR\]](#) and [\[MS-SRPL\]](#).

Active Directory is either deployed as AD/DS or as AD/LDS. This document describes both forms. When the specification does not refer specifically to AD/DS or AD/LDS, it applies to both.

The remainder of this section describes the structure of this document.

The basic state model is specified in section [3.1.1.1](#). The basic state model is prerequisite to the remainder of the document. Section [3.1.1.1](#) also includes descriptive content to introduce key concepts and refer to places in the document where the full specification is given.

The **schema** completes the state model and is specified in section [3.1.1.2](#). The schema is prerequisite to the remainder of the document.

Active Directory is a server for the **LDAP** protocol. Section [3.1.1.3](#) specifies the extensions and variations of the **LDAP** protocol supported by Active Directory.

The **LDAP** protocol is an access protocol that determines very little about the behavior of the data being accessed. Section [3.1.1.4](#) specifies read (LDAP Search) behaviors, and section [3.1.1.5](#) specifies **update** (LDAP Add, Modify, Modify **DN**, Delete) behaviors. Section [3.1.1.6](#) specifies background tasks required due to write operations, to the extent that those tasks are exposed by protocols.

One of the update behaviors is the maintenance of the change log for use by Windows NT 4.0 BDC **replication** [MS-NRPC] section 3.5. The maintenance of this change log is specified in section [3.1.1.7](#).

The security services that Active Directory offers clients of the **LDAP** protocol are specified in section [5.1](#).

Active Directory contains a number of **objects**, visible through **LDAP**, that have special significance to the system. Section [7.1](#) specifies these objects.

A server running Active Directory is part of a distributed system that performs replication. **Knowledge Consistency Checker (KCC)** is a component that is used to create spanning trees for DC-to-DC replication and is specified in section [7.2](#).

A server running Active Directory is responsible for publishing the services that it offers, in order to eliminate the administrative burden of configuring clients to use particular servers running Active Directory. A server running Active Directory also implements the server side of the **LDAP Ping** and **Mailslot Ping** protocols to aid clients in selecting among all the servers offering the same service. Section [7.3](#) specifies how a server running Active Directory publishes its services, and how a client needing some service can use this publication plus the **LDAP ping** or **mailslot ping** to locate a suitable server.

Computers in a network with Active Directory can be put into a state called "**domain joined**"; when in this state, the computer can authenticate itself. Section [7.4](#) specifies both the state in Active Directory and the state on a computer required for the "domain joined" state.

Each type of data stored in Active Directory has an associated function that compares two values to determine if they are equal and, if not, which is greater. Section [3.1.1.2](#) specifies all but one of these functions; the function for comparing two **Unicode** strings is sufficiently complex to require its own section, section [7.5](#).

## 1.4 Relationship to Other Protocols

This is the primary specification for Active Directory. The state model for this specification is prerequisite to the other specifications for Active Directory: [\[MS-DRSR\]](#) and [\[MS-SRPL\]](#).

## 1.5 Prerequisites/Preconditions

Active Directory requires an IP network and a DNS infrastructure.

## 1.6 Applicability Statement

Active Directory is not suitable for storing very large **attribute** values, because for instance there is no provision for check-pointing a large data transfer to allow restart after a failure. The bandwidth and latency of typical networks makes Active Directory unsuitable for storing volatile data in **replicated attributes**. Active Directory is especially suitable for storing security account data, including passwords, and email address book data.



## 1.7 Versioning and Capability Negotiation

Capability negotiation is performed using the **root DSE** as described in section [3.1.1.3.2](#).

## 1.8 Vendor-Extensible Fields

The **LDAP** protocol is not extensible by Active Directory applications. Applications extend the **directory** by adding objects, including **schema objects** to control the application objects.

## 1.9 Standards Assignments

Active Directory's extensions and variations of the **LDAP** protocol have no standards assignments. AD/DS uses private allocations for its **LDAP global catalog** port (3268) and **LDAP** global catalog port with **SSL/TLS** (3269).

## 2 Messages

The following sections specify how the **LDAP** protocol is transported and denote common information such as bit flag values.

### 2.1 Transport

The **LDAP** protocol transport is specified in section [3.1.1.3](#) and in [\[RFC2251\]](#) section 5.

### 2.2 Message Syntax

This section specifies types and data structures used in the remainder of this document. These type specifications reference the following:

- **DWORD** and **FILETIME** types: [\[MS-DTYP\]](#) sections 2.2 and 2.3.
- [repsFrom](#), [repsTo](#), [replUpToDateVector](#) abstract attributes of an **NC replica**: [MS-DRSR] sections [5:repsFrom](#), [5:repsTo](#), and [5:replUpToDateVector](#).
- **kCCFailedConnections**, **kCCFailedLinks**, **RPCClientContexts**, **RPCOutgoingContexts**, **ldapConnections**, and **replicationQueue** variables of a **DC**: [MS-DRSR] sections [5:kCCFailedConnections](#), [5:kCCFailedLinks](#), [5:RPCClientContexts](#), [5:RPCOutgoingContexts](#), [5:LDAPConnections](#), and [5:ReplicationQueue](#).
- **stamp** variable of an attribute: [MS-DRSR] section [5:AttributeStamp](#).
- **stamp** variable of a **link value**: [MS-DRSR] section [5:LinkValueStamp](#).
- **DS\_REPL\_ATTR\_META\_DATA\_2**, **DS\_REPL\_CURSOR\_3W**, **DS\_REPL\_KCC\_DSA\_FAILUREW**, **DS\_REPL\_NEIGHBORW**, **DS\_REPL\_OPW**, **DS\_REPL\_VALUE\_META\_DATA\_2** types: [MS-DRSR] section [4.1.13.1](#).
- **IDL\_DRSGetReplInfo** method: [MS-DRSR] section [4.1.13](#).

#### 2.2.1 LCID-Locale Mapping Table

The following table maps Windows locales (for example, French - France, Irish - Ireland) to numeric identifiers called **LCIDs**. These numeric identifiers are used as input to the **Unicode** string comparison function specified in section [7.5](#). They are also used to name display specifier **containers**, specified in section [7.1](#) ("Display specifiers container").

LCID	Language	Location
0436	Afrikaans	South Africa
041c	Albanian	Albania
0401	Arabic	Saudi Arabia
0801	Arabic	Iraq
0c01	Arabic	Egypt
1001	Arabic	Libya
1401	Arabic	Algeria

<b>LCID</b>	<b>Language</b>	<b>Location</b>
1801	Arabic	Morocco
1c01	Arabic	Tunisia
2001	Arabic	Oman
2401	Arabic	Yemen
2801	Arabic	Syria
2c01	Arabic	Jordan
3001	Arabic	Lebanon
3401	Arabic	Kuwait
3801	Arabic	U.A.E.
3c01	Arabic	Bahrain
4001	Arabic	Qatar
042b	Armenian	Armenia
082c	Azeri (Cyrillic)	Azerbaijan
042c	Azeri (Latin)	Azerbaijan
042d	Basque	Basque
0423	Belarusian	Belarus
201a	Bosnian (Cyrillic)	Bosnia and Herzegovina
141a	Bosnian (Latin)	Bosnia and Herzegovina
0402	Bulgarian	Bulgaria
0403	Catalan	Catalan
0004	Chinese	Simplified
0404	Chinese	Taiwan
0804	Chinese	PRC
0c04	Chinese	Hong Kong SAR
1004	Chinese	Singapore
1404	Chinese	Macao SAR
7c04	Chinese	Traditional
041a	Croatian	Croatia
101a	Croatian (Latin)	Bosnia and Herzegovina

<b>LCID</b>	<b>Language</b>	<b>Location</b>
0405	Czech	Czech Republic
0406	Danish	Denmark
0465	Divehi	Maldives
0813	Dutch	Belgium
0413	Dutch	Netherlands
1009	English	Canada
2009	English	Jamaica
2409	English	Caribbean
2809	English	Belize
2c09	English	Trinidad
0809	English	United Kingdom
1809	English	Ireland
1c09	English	South Africa
3009	English	Zimbabwe
0c09	English	Australia
1409	English	New Zealand
3409	English	Philippines
0409	English	United States
0425	Estonian	Estonia
0438	Faroese	Faroe Islands
0464	Filipino	Philippines
040b	Finnish	Finland
0c0c	French	Canada
040c	French	France
180c	French	Monaco
100c	French	Switzerland
080c	French	Belgium
140c	French	Luxembourg
0462	Frisian	Netherlands

<b>LCID</b>	<b>Language</b>	<b>Location</b>
0456	Galician	Galician
0437	Georgian	Georgia
0407	German	Germany
0807	German	Switzerland
0c07	German	Austria
1407	German	Liechtenstein
1007	German	Luxembourg
0408	Greek	Greece
0447	Gujarati	India
040d	Hebrew	Israel
0439	Hindi	India
040e	Hungarian	Hungary
040f	Icelandic	Iceland
0421	Indonesian	Indonesia
085d	Inuktitut (Latin)	Canada
083c	Irish	Ireland
0434	isiXhosa	South Africa
0435	isiZulu	South Africa
0410	Italian	Italy
0810	Italian	Switzerland
0411	Japanese	Japan
044b	Kannada	India
043f	Kazakh	Kazakhstan
0441	Kiswahili	Kenya
0457	Konkani	India
0412	Korean	Korea
0440	Kyrgyz	Kirghizstan
0426	Latvian	Latvia
0427	Lithuanian	Lithuania

<b>LCID</b>	<b>Language</b>	<b>Location</b>
046e	Luxembourgish	Luxembourg
042f	Macedonian (FYROM)	Macedonia, Former Yugoslav Republic of
043e	Malay	Malaysia
083e	Malay	Brunei Darussalam
043a	Maltese	Malta
0481	Maori	New Zealand
047a	Mapudungun	Chile
044e	Marathi	India
047c	Mohawk	Mohawk
0450	Mongolian (Cyrillic)	Mongolia
0461	Nepali	Nepal
0414	Norwegian (Bokmål)	Norway
0814	Norwegian (Nynorsk)	Norway
0463	Pashto	Afghanistan
0429	Persian	Iran
0415	Polish	Poland
0416	Portuguese	Brazil
0816	Portuguese	Portugal
0446	Punjabi (Gurmukhi)	India
046b	Quechua	Bolivia
086b	Quechua	Ecuador
0c6b	Quechua	Peru
0418	Romanian	Romania
0417	Romansh	Switzerland
0419	Russian	Russia
243b	Sami, Inari	Finland
143b	Sami, Lule	Sweden
103b	Sami, Lule	Norway
043b	Sami, Northern	Norway

<b>LCID</b>	<b>Language</b>	<b>Location</b>
083b	Sami, Northern	Sweden
0c3b	Sami, Northern	Finland
203b	Sami, Skolt	Finland
183b	Sami, Southern	Norway
1c3b	Sami, Southern	Sweden
044f	Sanskrit	India
0c1a	Serbian (Cyrillic)	Serbia
0c1a	Serbian (Cyrillic)	Montenegro
1c1a	Serbian (Cyrillic)	Bosnia and Herzegovina
081a	Serbian (Latin)	Serbia
081a	Serbian (Latin)	Montenegro
181a	Serbian (Latin)	Bosnia and Herzegovina
046c	Sesotho sa Leboa	South Africa
0432	Setswana	South Africa
041b	Slovak	Slovakia
0424	Slovenian	Slovenia
080a	Spanish	Mexico
100a	Spanish	Guatemala
140a	Spanish	Costa Rica
180a	Spanish	Panama
1c0a	Spanish	Dominican Republic
200a	Spanish	Venezuela
240a	Spanish	Colombia
280a	Spanish	Peru
2c0a	Spanish	Argentina
300a	Spanish	Ecuador
340a	Spanish	Chile
3c0a	Spanish	Paraguay
400a	Spanish	Bolivia

LCID	Language	Location
440a	Spanish	El Salvador
480a	Spanish	Honduras
4c0a	Spanish	Nicaragua
500a	Spanish	Commonwealth of Puerto Rico
380a	Spanish	Uruguay
0c0a	Spanish (International Sort)	Spain
040a	Spanish (Traditional Sort)	Spain
041d	Swedish	Sweden
081d	Swedish	Finland
045a	Syriac	Syria
0449	Tamil	India
0444	Tatar	Russia
044a	Telugu	India
041e	Thai	Thailand
041f	Turkish	Turkey
0422	Ukrainian	Ukraine
0420	Urdu	Pakistan
0843	Uzbek (Cyrillic)	Uzbekistan
0443	Uzbek (Latin)	Uzbekistan
042a	Vietnamese	Vietnam
0452	Welsh	United Kingdom

### 2.2.2 DS\_REPL\_NEIGHBORW\_BLOB

The DS\_REPL\_NEIGHBORW\_BLOB structure is a representation of a tuple from the [repsFrom](#) or [repsTo](#) abstract attribute of an NC replica. This structure, retrieved using an **LDAP** search method, is an alternative representation of DS\_REPL\_NEIGHBORW, retrieved using the IDL\_DRSGetReplInfo **RPC** method.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
oszNamingContext																															



oszSourceDsaDN
oszSourceDsaAddress
oszAsyncIntersiteTransportDN
dwReplicaFlags
dwReserved
uuidNamingContextObjGuid
...
...
...
uuidSourceDsaObjGuid
...
...
...
uuidSourceDsaInvocationID
...
...
...
uuidAsyncIntersiteTransportObjGuid
...
...
...

usnLastObjChangeSynced
...
usnAttributeFilter
...
ftimeLastSyncSuccess
...
ftimeLastSyncAttempt
...
dwLastSyncResult
cNumConsecutiveSyncFailures
data (variable)
...

**oszNamingContext:** A 32-bit offset, in bytes, from the address of this structure to a null-terminated **Unicode** string that contains the **NC** to which this replication state data pertains.

**oszSourceDsaDN:** A 32-bit offset, in bytes, from the address of this structure to a null-terminated **Unicode** string that contains the DN of the [nTDSDSA](#) **object** of the source server to which this replication state data pertains. Each source server has different associated neighbor data.

**oszSourceDsaAddress:** A 32-bit offset, in bytes, from the address of this structure to a null-terminated **Unicode** string that contains the transport-specific network address of the source server. That is, a directory name service name for RPC/IP replication, or an **SMTP** address for an SMTP replication.

**oszAsyncIntersiteTransportDN:** A 32-bit offset, in bytes, from the address of this structure to a null-terminated **Unicode** string that contains the DN of the [interSiteTransport](#) object that corresponds to the transport over which replication is performed. This member contains NULL for RPC/IP replication.

**dwReplicaFlags:** A 32-bit bitfield containing a set of flags that specify attributes and options for the replication data. This can be zero or a combination of one or more of the following flags presented in little-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	D	S	W	X	X	X	X	X	X	X	X	X	X	T	X	X	X	N	X	X	X	F	F	X	X	N	C	X	X	X	X
I	S	O												W				S				S	S			C	C				
T	S	S												S								N	P			N					

**X:** Unused. Must be zero and ignored.

**W (DS\_REPL\_NBR\_WRITEABLE, 0x00000010):** The NC replica is writable.

**SOS (DS\_REPL\_NBR\_SYNC\_ON\_STARTUP, 0x00000020):** Replication of this NC from this source is attempted when the destination server is booted.

**DSS (DS\_REPL\_NBR\_DO\_SCHEDULED\_SYNCS, 0x00000040):** Perform replication on a schedule.

**AIT (DS\_REPL\_NBR\_USE\_ASYNC\_INTERSITE\_TRANSPORT, 0x00000080):** Perform replication indirectly through the Inter-**Site** Messaging Service. This flag is set only when replicating over SMTP. This flag is not set when replicating over inter-site RPC/IP.

**TWS (DS\_REPL\_NBR\_TWO\_WAY\_SYNC, 0x00000200):** When inbound replication is complete, the destination server requests the source server to synchronize in the reverse direction.

**FSP (DS\_REPL\_NBR\_FULL\_SYNC\_IN\_PROGRESS, 0x00010000):** The destination server is performing a full synchronization from the source server.

**FSN (DS\_REPL\_NBR\_FULL\_SYNC\_NEXT\_PACKET, 0x00020000):** The last packet from the source indicated a modification of an object that the destination server has not yet created. The next packet to be requested instructs the source server to put all attributes of the modified object into the packet.

**NS (DS\_REPL\_NBR\_NEVER\_SYNCED, 0x00200000):** A synchronization has never been successfully completed from this source.

**CC (DS\_REPL\_NBR\_COMPRESS\_CHANGES, 0x10000000):** Changes received from this source are to be compressed.

**NCN (DS\_REPL\_NBR\_NO\_CHANGE\_NOTIFICATIONS, 0x20000000):** Applies to [repsFrom](#) only. The DC storing this [repsFrom](#) is not configured to receive change notifications from this source.

**dwReserved:** Reserved for future use.

**uuidNamingContextObjGuid:** A **GUID** structure, as defined in [\[MS-DTYP\]](#) section **2.3.2**, specifying the [objectGUID](#) of the NC that corresponds to `oszNamingContext`.

**uuidSourceDsaObjGuid:** A GUID structure, as defined in [\[MS-DTYP\]](#) section **2.3.2**, specifying the [objectGUID](#) of the [nTDSDSA](#) object that corresponds to `oszSourceDsaDN`.

**uuidSourceDsaInvocationID:** A GUID structure, as defined in [\[MS-DTYP\]](#) section **2.3.2**, specifying the **invocation ID** used by the source server as of the last replication attempt.

**uuidAsyncIntersiteTransportObjGuid:** A GUID structure, as defined in [\[MS-DTYP\]](#) section **2.3.2**, specifying the [objectGUID](#) of the inter-site transport object that corresponds to `oszAsyncIntersiteTransportDN`.

**usnLastObjChangeSynced:** A **USN** value, as defined in section [3.1.1.1.9](#), containing the **update sequence number** of the last object update received.

**usnAttributeFilter:** A **USN** value, as defined in section [3.1.1.1.9](#), containing the `usnLastObjChangeSynced` value at the end of the last complete, successful **replication cycle**, or 0 if none.

**ftimeLastSyncSuccess:** A FILETIME structure that contains the date and time the last successful replication cycle was completed from this source. All members of this structure are zero if the replication cycle has never been completed.

**ftimeLastSyncAttempt:** A FILETIME structure that contains the date and time of the last replication attempt from this source. All members of this structure are zero if the replication has never been attempted.

**dwLastSyncResult:** A 32-bit unsigned integer specifying a **Windows error code** associated with the last replication attempt from this source. Contains `ERROR_SUCCESS` if the last attempt was successful.

**cNumConsecutiveSyncFailures:** A 32-bit integer specifying the number of failed replication attempts that have been made from this source since the last successful replication attempt or since the source was added as a neighbor, if no previous attempt succeeded.

**data:** This field contains all the null-terminated strings that are pointed to by the offset fields in the structure (`oszNamingContext`, `oszSourceDsaDN`, `oszSourceDsaAddress`, `oszAsyncIntersiteTransportDN`). The strings are packed into this field and the offsets can be used to determine the start of each string.

All multi-byte fields have little-endian byte ordering.

### 2.2.3 DS\_REPL\_KCC\_DSA\_FAILUREW\_BLOB

The `DS_REPL_KCC_DSA_FAILUREW_BLOB` structure is a representation of a tuple from the `kCCFailedConnections` or `kCCFailedLinks` variables of a DC. This structure, retrieved using an **LDAP** search method, is an alternative representation of `DS_REPL_KCC_DSA_FAILUREW`, retrieved using the `IDL_DRSGetReplInfo` **RPC** method.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
oszDsaDN																															
uuidDsaObjGuid																															
...																															
...																															
...																															
ftimeFirstFailure																															
...																															
cNumFailures																															
dwLastResult																															
data (variable)																															
...																															

**oszDsaDN:** A 32-bit offset, in bytes, from the address of this structure to a null-terminated string that contains the DN of the [nTDSDSA](#) object of the source server.

**uuidDsaObjGuid:** A GUID structure, defined in [\[MS-DTYP\]](#) section **2.3.2**, specifying the [objectGUID](#) of the object represented by the oszDsaDN member.

**ftimeFirstFailure:** A FILETIME structure, the content of which depends on the requested binary replication data.

Attribute requested	Meaning
<a href="#">msDS-ReplConnectionFailures</a>	Contains the date and time that the first failure occurred when attempting to establish a replica link to the source server.
<a href="#">msDS-ReplLinkFailures</a>	Contains the date and time that the first failure occurred when replicating from the source server.

**cNumFailures:** A 32-bit unsigned integer specifying the number of consecutive failures since the last successful replication.

**dwLastResult:** A 32-bit unsigned integer specifying the error code associated with the most recent failure, or ERROR\_SUCCESS if the specific error is *unavailable*.

**data:** The data field contains the null-terminated string that contains the DN of the [nTDSDSA](#) object of the source server.

All multi-byte fields have little-endian byte ordering.

**2.2.4 DS\_REPL\_OPW\_BLOB**

The DS\_REPL\_OPW\_BLOB structure is a representation of a tuple from the *replicationQueue* variable of a DC. This structure, retrieved using an **LDAP** search method, is an alternative representation of DS\_REPL\_OPW, retrieved using the IDL\_DRSGetReplInfo **RPC** method.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ftimeEnqueued																															
...																															
ulSerialNumber																															
ulPriority																															
opType																															
ulOptions																															
oszNamingContext																															
oszDsaDN																															
oszDsaAddress																															
uuidNamingContextObjGuid																															
...																															
...																															
...																															
uuidDsaObjGuid																															
...																															

...
...
data (variable)
...

**ftimeEnqueued:** A FILETIME structure that contains the date and time that this operation was added to the queue.

**ulSerialNumber:** An unsigned integer specifying the identifier of the operation. The counter used to assign this identifier is volatile; it is reset during startup of a DC. Therefore these identifiers are only unique between restarts of a DC.

**ulPriority:** An unsigned integer specifying the priority value of this operation. Tasks with a higher priority value are executed first. The priority is calculated by the server based on the type of operation and its parameters.

**opType:** Contains one of the following values that indicate the type of operation that this structure represents.

Operation	Value
DS_REPL_OP_TYPE_SYNC	0
DS_REPL_OP_TYPE_ADD	1
DS_REPL_OP_TYPE_DELETE	2
DS_REPL_OP_TYPE_MODIFY	3
DS_REPL_OP_TYPE_UPDATE_REFS	4

**ulOptions:** Zero or more bits from the DRS options defined in [MS-DRSR] section [5:DRS\\_OPTIONS](#), the interpretation of which depends on the **OpType**.

**oszNamingContext:** Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated string that contains the DN of the NC associated with this operation. For example, the NC to be synchronized for DS\_REPL\_OP\_TYPE\_SYNC.

**oszDsaDN:** Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated string that contains the DN of the [nTDSDSA](#) object of the remote server corresponding to this operation. For example, the server from which to ask for changes for DS\_REPL\_OP\_TYPE\_SYNC. This can be NULL.

**oszDsaAddress:** Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated string that contains the transport-specific network address of the remote server associated with this operation. For example, the DNS or SMTP address of the server from which to ask for changes for DS\_REPL\_OP\_TYPE\_SYNC. This can be NULL.

**uuidNamingContextObjGuid:** A GUID structure, as defined in [\[MS-DTYP\]](#) section **2.3.2**, specifying the [objectGUID](#) of the NC identified by oszNamingContext.

**uuidDsaObjGuid:** A GUID structure, as defined in [\[MS-DTYP\]](#) section **2.3.2**, specifying the [objectGUID](#) of the directory system agent object identified by `oszDsaDN`.

**data:** This field contains all the null-terminated strings that are pointed to by the offset fields in the structure (`oszNamingContext`, `oszDsaDN`, `oszDsaAddress`). The strings are packed into this field and the offsets can be used to determine the start of each string.

All multi-byte fields have little-endian byte ordering.

**2.2.5 DS\_REPL\_QUEUE\_STATISTICSW\_BLOB**

The `DS_REPL_QUEUE_STATISTICSW_BLOB` structure is the statistics related to the *replicationQueue* variable of a DC, returned by reading the `msDS-RepQueueStatistics` rootDSE attribute (section [3.1.1.3.2.30](#)).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ftimeCurrentOpStarted																															
...																															
cNumPendingOps																															
ftimeOldestSync																															
...																															
ftimeOldestAdd																															
...																															
ftimeOldestMod																															
...																															
ftimeOldestDel																															
...																															
ftimeOldestUpdRefs																															
...																															

**ftimeCurrentOpStarted:** A FILETIME structure that contains the date and time that the currently running operation started.



**cNumPendingOps:** A unsigned integer specifying the number of currently pending operations.

**ftimeOldestSync:** A FILETIME structure that contains the date and time of the oldest synchronization operation.

**ftimeOldestAdd:** A FILETIME structure that contains the date and time of the oldest add operation.

**ftimeOldestMod:** A FILETIME structure that contains the date and time of the oldest modification operation.

**ftimeOldestDel:** A FILETIME structure that contains the date and time of the oldest delete operation.

**ftimeOldestUpdRefs:** A FILETIME structure that contains the date and time of the oldest reference update operation.

All multi-byte fields have little-endian byte ordering.

## 2.2.6 DS\_REPL\_CURSOR\_BLOB

The DS\_REPL\_CURSOR\_BLOB packet representation of a cursor tuple from the *upToDateVector* variable of an NC replica. This structure, retrieved using an **LDAP** search method, is an alternative representation of DS\_REPL\_CURSOR\_3W, retrieved using the IDL\_DRSGetReplInfo **RPC** method.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
uuidSourceDsaInvocationID																															
...																															
...																															
...																															
usnAttributeFilter																															
...																															
fTimeLastSyncSuccess																															
...																															
oszSourceDsaDN																															
data (variable)																															
...																															

**uuidSourceDsaInvocationID:** A GUID structure, defined in [\[MS-DTYP\]](#) section **2.3.2**, specifying the Invocation ID of the originating server to which the usnAttributeFilter corresponds.

**usnAttributeFilter:** A USN value, as defined in section [3.1.1.1.9](#), containing the maximum **update sequence number** to which the destination server can indicate that it has recorded all changes originated by the given server at update sequence numbers less than, or equal to, this update sequence number. This is used to **filter** changes at replication source servers that the destination server has already applied.

**fTimeLastSyncSuccess:** A FILETIME structure that contains the date and time of the last successful synchronization operation.

**oszSourceDsaDN:** Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated **Unicode** string that contains the **distinguished name** of the directory service agent that corresponds to the source server to which this replication state data applies.

**data:** This field contains the null-terminated string pointed to by the offset field in the structure (oszSourceDsaDN). The offset can be used to determine the start of the string.

All multi-byte fields have little-endian byte ordering.

### 2.2.7 DS\_REPL\_ATTR\_META\_DATA\_BLOB

The DS\_REPL\_ATTR\_META\_DATA\_BLOB packet is a representation of a *stamp* variable (of type **AttributeStamp**) of an attribute. This structure, retrieved using an **LDAP** search method, is an alternative representation of DS\_REPL\_ATTR\_META\_DATA\_2, retrieved using the IDL\_DRSGetReplInfo **RPC** method.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
oszAttributeName																															
dwVersion																															
ftimeLastOriginatingChange																															
...																															
uuidLastOriginatingDsaInvocationID																															
...																															
...																															
...																															
usnOriginatingChange																															
...																															
usnLocalChange																															
...																															
oszLastOriginatingDsaDN																															
data (variable)																															
...																															

**oszAttributeName:** Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated **Unicode** string that contains the **LDAP** display name of the attribute corresponding to this metadata.

**dwVersion:** Contains the *dwVersion* of this attribute's **AttributeStamp**, as specified in section [3.1.1.1.9](#).

**ftimeLastOriginatingChange:** Contains the *timeChanged* of this attribute's **AttributeStamp**, as specified in section [3.1.1.1.9](#).

**uuidLastOriginatingDsaInvocationID:** Contains the *uuidOriginating* of this attribute's **AttributeStamp**, as specified in section [3.1.1.1.9](#).

**usnOriginatingChange:** Contains the *usnOriginating* of this attribute's **AttributeStamp**, as specified in section [3.1.1.1.9](#).

**usnLocalChange:** A USN value, defined in section [3.1.1.1.9](#), specifying the USN on the destination server (the server from which the metadata information is retrieved) at which the last change to this attribute was applied. This value typically is different on all servers.

**oszLastOriginatingDsaDN:** Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated **Unicode** string that contains the DN of the [nTDSDSA](#) object of the server that originated the last replication.

**data:** This field contains all the null-terminated strings that are pointed to by the offset fields in the structure (*oszAttributeName*, *oszLastOriginatingDsaDN*). The strings are packed into this field and the offsets can be used to determine the start of each string.

All multi-byte fields have little-endian byte ordering.

## 2.2.8 DS\_REPL\_VALUE\_META\_DATA\_BLOB

The DS\_REPL\_VALUE\_META\_DATA\_BLOB packet is a representation of a *stamp* variable (of type **LinkValueStamp**) of a link value. This structure, retrieved using an **LDAP** search method, is an alternative representation of DS\_REPL\_VALUE\_META\_DATA\_2, retrieved using the IDL\_DRSGetReplInfo **RPC** method.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
oszAttributeName																															
oszObjectDn																															
cbData																															
pbData																															
ftimeDeleted																															
...																															
ftimeCreated																															

...
dwVersion
fTimeLastOriginatingChange
...
uuidLastOriginatingDsaInvocationID
...
...
...
usnOriginatingChange
...
usnLocalChange
...
oszLastOriginatingDsaDN
data (variable)
...

**oszAttributeName:** Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated **Unicode** string that contains the **LDAP** display name of the attribute corresponding to this metadata.

**oszObjectDn:** Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated **Unicode** string that contains the DN of the object that this attribute belongs to.

**cbData:** Contains the number of bytes in the pbData array.

**pbData:** Contains a 32-bit offset, in bytes, from the address of this structure to a buffer that contains the attribute replication metadata. The cbData member contains the length, in bytes, of this buffer.

**fTimeDeleted:** Contains the *timeDeleted* of this link value's LinkValueStamp, as specified in section [3.1.1.1.9](#).

**ftimeCreated:** Contains the *timeCreated* of this link value's LinkValueStamp, as specified in section [3.1.1.1.9](#).

**dwVersion:** Contains the *dwVersion* of this link value's LinkValueStamp, as specified in section [3.1.1.1.9](#).

**ftimeLastOriginatingChange:** Contains the *timeChanged* of this link value's LinkValueStamp, as specified in section [3.1.1.1.9](#).

**uuidLastOriginatingDsaInvocationID:** Contains the *uuidOriginating* of this link value's LinkValueStamp, as specified in section [3.1.1.1.9](#).

**usnOriginatingChange:** Contains the *usnOriginating* of this link value's LinkValueStamp, as specified in section [3.1.1.1.9](#).

**usnLocalChange:** Specifies the USN on the destination server, (that is, the server from which the metadata information is being retrieved), at which the last change to this attribute was applied. This value is typically different on all servers.

**oszLastOriginatingDsaDN:** Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated **Unicode** string that contains the DN of the [nTDSDSA](#) object of the server that originated the last replication.

**data:** This field contains all the null-terminated strings that are pointed to by the offset fields in the structure (oszAttributeName, oszObjectDn, oszLastOriginatingDsaDN) and the buffer pointed to by pbData. The strings and buffers are packed into this field (aligned at 32-bit boundaries) and the offsets can be used to determine the start of each string.

All multi-byte fields have little-endian byte ordering.

2.2.9 Search Flags

The following table defines the valid search flags used on attributes, as specified in section [3.1.1.2.3](#). The flags are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	0 <sup>1</sup>	1	2	3	4	5	6	7	8	9	0 <sup>2</sup>	1	2	3	4	5	6	7	8	9	0 <sup>3</sup>	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	R	N	C	S	T	C	P	A	P	I
																						O	V	F	T	P	P	R	R	I	X

**X:** Unused. Must be zero and ignored.

**IX (fATTINDEX):** Specifies a hint to the DC to create an index for the attribute.

**PI (fPDNTATTINDEX):** Specifies a hint to the DC to create an index for the **container** and the attribute.

**AR(fANR):** Specifies that the attribute is a member of the **Ambiguous Name Resolution (ANR)** set.

**PR (fPRESERVEONDELETE):** Specifies that the attribute must be preserved on logical deletion.

**CP (fCOPY):** Interpreted by LDAP clients, not by server. Specifies that the attribute must be copied when copying the object.

**TP (fTUPLEINDEX):** Specifies a hint for the DC to create a tuple index for the attribute. This will affect the performance of searches where the wildcard appears at the front of the search string.

**ST (fSUBTREEATTINDEX):** Specifies a hint for the DC to create subtree index for VLV search.

**CF (fCONFIDENTIAL):** Specifies that the attribute is confidential. Special **access check** is needed.

**NV (fNEVERVALUEAUDIT):** Specifies that auditing of changes to individual values contained in this attribute should never be performed. Auditing is outside of the state model.

**RO (fRODCAttribute):** Specifies that the attribute is a member of the **filtered attribute set**.

## 2.2.10 System Flags

The following table defines the valid system flags used on **directory objects**. The flags are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
D	A	A	A	D	D	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	R	B	O	C	P	N
D	R	M	L	R	M	E																				D	S	P	S	S	R

**X:** Unused. Must be zero and ignored.

**NR (FLAG\_ATTR\_NOT\_REPLICATED or FLAG\_CR\_NTDS\_NC):** When used on an [attributeSchema](#) object, it specifies that this attribute is not replicated. If it is used on a [crossRef](#) object, it specifies that the NC that the [crossRef](#) is for is an Active Directory NC.

**PS (FLAG\_ATTR\_REQ\_PARTIAL\_SET\_MEMBER or FLAG\_CR\_NTDS\_DOMAIN):** When used on an [attributeSchema](#) object, it specifies that the attribute is a member of **PAS**. If used on a [crossRef](#) object, it specifies that the NC is a **domain NC**.

**CS (FLAG\_ATTR\_IS\_CONSTRUCTED or FLAG\_CR\_NTDS\_NOT\_GC\_REPLICATED):** When used on an [attributeSchema](#) object, this flag specifies that the attribute is a **constructed attribute**. If used on a [crossRef](#) object, it specifies that the NC is not to be replicated to GCs.

**OP (FLAG\_ATTR\_IS\_OPERATIONAL):** Only used on [attributeSchema](#) object. It specifies that the attribute is an **operational attribute**.

**BS (FLAG\_SCHEMA\_BASE\_OBJECT):** Only used on [attributeSchema](#) and [classSchema](#) objects. It specifies that this attribute or class is part of the base schema. Modifications to base schema objects are specially restricted.

**RD (FLAG\_ATTR\_IS\_RDN):** Only used on [attributeSchema](#) object. It specifies that this attribute can be used as an **RDN attribute**.

**DE (FLAG\_DISALLOW\_MOVE\_ON\_DELETE):** Specifies that the object does not move to the Deleted Objects **container** when the object is deleted.

**DM (FLAG\_DOMAIN\_DISALLOW\_MOVE):** Specifies that if the object is in a **domain NC**, the object cannot be moved.

**DR (FLAG\_DOMAIN\_DISALLOW\_RENAME):** Specifies that if the object is in a domain NC, the object cannot be renamed.

**AL (FLAG\_CONFIG\_ALLOW\_LIMITED\_MOVE):** Specifies that if the object is in the config NC, the object can be moved, with restrictions.

**AM (FLAG\_CONFIG\_ALLOW\_MOVE):** Specifies that if the object is in the config NC, the object can be moved.

**AR (FLAG\_CONFIG\_ALLOW\_RENAME):** Specifies that if the object is in the config NC, the object can be renamed.

**DD (FLAG\_DISALLOW\_DELETE):** Specifies that the object cannot be deleted.

### 2.2.11 schemaFlagsEx Flags

The following table defines the valid [schemaFlagsEx](#) flags that are used on attributes, as specified in section [3.1.1.2.3](#). The flags are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	C
																															R

**X:** Unused. MUST be zero and ignored.

**CR (FLAG\_ATTR\_IS\_CRITICAL):** Specifies that the attribute is not a member of the filtered attribute set even if the fRODCFilteredAttribute flag is set. For more information, see sections [3.1.1.2.3](#) and [3.1.1.2.3.5](#).

### 2.2.12 Group Type Flags

Constants for defining **group** type.

Symbolic Name	Value
GROUP_TYPE_BUILTIN_LOCAL_GROUP	0x00000001
GROUP_TYPE_ACCOUNT_GROUP	0x00000002
GROUP_TYPE_RESOURCE_GROUP	0x00000004
GROUP_TYPE_UNIVERSAL_GROUP	0x00000008
GROUP_TYPE_APP_QUERY_GROUP	0x00000020
GROUP_TYPE_SECURITY_ENABLED	0x80000000

### 2.2.13 Security Privilege Flags

Constants for defining security **privilege**.

Symbolic Name	Value
SE_TAKE_OWNERSHIP_PRIVILEGE	0x00000009



Symbolic Name	Value
SE_SECURITY_PRIVILEGE	0x00000008
SE_RESTORE_PRIVILEGE	0x00000012
SE_DEBUG_PRIVILEGE	0x00000014
SE_ENABLE_DELEGATION_PRIVILEGE	0x0000001B

## 2.2.14 Domain RID Values

Constants for defining **domain RIDs**.

Symbolic Name	Value
DOMAIN_USER_RID_ADMIN	0x000001F4
DOMAIN_USER_RID_KRBTGT	0x000001F6
DOMAIN_GROUP_RID_ADMINS	0x00000200
DOMAIN_GROUP_RID_CONTROLLERS	0x00000204
DOMAIN_GROUP_RID_SCHEMA_ADMINS	0x00000206
DOMAIN_GROUP_RID_READONLY_CONTROLLERS	0x00000209
DOMAIN_ALIAS_RID_ADMINS	0x00000220
DOMAIN_ALIAS_RID_ACCOUNT_OPS	0x00000224
DOMAIN_ALIAS_RID_SYSTEM_OPS	0x00000225
DOMAIN_ALIAS_RID_PRINT_OPS	0x00000226
DOMAIN_ALIAS_RID_BACKUP_OPS	0x00000227
DOMAIN_ALIAS_RID_REPLICATOR	0x00000228

## 2.2.15 userAccountControl Bits

Bit flags describing various qualities of a security account. The flags are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	P	N	T	P	D	D	N	T	S	X	D	X	X	S	W	I	X	N	X	E	C	N	L	H	X	D	X
					S	A	A	E	R	K	D	D	R		P			T	T	D				T	C	R					

**X:** Unused. Must be zero and ignored.

**D (ADS\_UF\_ACCOUNT\_DISABLE):** Specifies that the account is not enabled for authentication.

**HR (ADS\_UF\_HOMEDIR\_REQUIRED):** Specifies that the [homeDirectory](#) attribute is required.

**L (ADS\_UF\_LOCKOUT):** Specifies that the account is temporarily locked out.

**NR (ADS\_UF\_PASSWD\_NOTREQD):** Specifies that the password-length policy does not apply to this user.

**CC (ADS\_UF\_PASSWD\_CANT\_CHANGE):** Specifies that the user cannot change their password.

**ET (ADS\_UF\_ENCRYPTED\_TEXT\_PASSWORD\_ALLOWED):** Specifies that the clear-text password is to be persisted.

**N (ADS\_UF\_NORMAL\_ACCOUNT):** Specifies that the account is the default account type that represents a typical user.

**ID (ADS\_UF\_INTERDOMAIN\_TRUST\_ACCOUNT):** Specifies that the account is for a domain-to-domain [trust](#).

**WT (ADS\_UF\_WORKSTATION\_ACCOUNT):** Specifies that the account is a computer account for a computer that is a member of this domain.

**ST (ADS\_UF\_SERVER\_TRUST\_ACCOUNT):** Specifies that the account is a computer account for a DC.

**DP (ADS\_UF\_DONT\_EXPIRE\_PASSWD):** Specifies that the password does not expire for the account.

**SR (ADS\_UF\_SMARTCARD\_REQUIRED):** Specifies that the account needs a smart card to log in.

**TD (ADS\_UF\_TRUSTED\_FOR\_DELEGATION):** Used by the Kerberos protocol. This bit indicates that the "OK as Delegate" ticket flag, as described in [\[RFC4120\]](#) section 2.8, MUST be set.

**ND (ADS\_UF\_NOT\_DELEGATED):** Used by the Kerberos protocol. This bit indicates that the TGTs of this account and the service tickets obtained by this account are not marked as forwardable or proxiable when the forwardable or proxiable ticket flags are requested. For more information, see [\[RFC4120\]](#).

**DK (ADS\_UF\_USE\_DES\_KEY\_ONLY):** Used by the Kerberos protocol. This bit indicates that only des-cbc-md5 or des-cbc-crc keys, as defined in [\[RFC3961\]](#), are used in the Kerberos protocols for this account.

**DR (ADS\_UF\_DONT\_REQUIRE\_PREAUTH):** Used by the Kerberos protocol. This bit indicates that the account is not required to present valid pre-authentication data, as described in [\[RFC4120\]](#) section 7.5.2.

**PE (ADS\_UF\_PASSWORD\_EXPIRED):** Specifies that the password age on the user has exceeded the maximum password age policy.

**TA (ADS\_UF\_TRUSTED\_TO\_AUTHENTICATE\_FOR\_DELEGATION):** Used by the Kerberos protocol. When set, this bit indicates that the account (when running as a service) obtains an S4U2self service ticket (as specified in [\[MS-SFU\]](#)) with the forwardable flag set. If this bit is cleared, the forwardable flag is not set in the S4U2self service ticket.

**NA (ADS\_UF\_NO\_AUTH\_DATA\_REQUIRED):** Used by the Kerberos protocol. This bit indicates that when the KDC is issuing a service ticket for this account, the PAC MUST NOT be included. For more information, see [\[RFC4120\]](#).

**PS (ADS\_UF\_PARTIAL\_SECRETS\_ACCOUNT):** Specifies that the account is a computer account for an RODC. If this bit is set, the ADS\_UF\_WORKSTATION\_ACCOUNT must also be set. This flag is only interpreted by a DC whose DC functional level is DS\_BEHAVIOR\_WIN2008 or greater.

## 3 Details

The following sections specify details of the abstract data model and directory operations for Active Directory.

### 3.1 Common Details

#### 3.1.1 Abstract Data Model

Sections [3.1.1.1](#) and [3.1.1.2](#) describe a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

##### 3.1.1.1 State Model

###### 3.1.1.1.1 Scope

The specification of all Active Directory protocols is based on a definition, shared by all Active Directory protocols, of the state of a server running Active Directory that is implied by the protocols. Call this the *state model* of Active Directory.

Active Directory's state model divides into two categories:

1. Certain state that is represented as objects and attributes within Active Directory is *promoted directly* into the state model. State within Active Directory becomes part of the state model if it satisfies one of the following conditions.

1. If it is replicated.
2. If it is non-replicated, and is specifically accessed remotely by a component of Windows.

The representation of non-replicated state that is only accessed locally by a server implementing Active Directory is private to the implementation of Windows, and therefore is *not* promoted directly into the state model. It might still need to be modeled as described in category 2 below.

Excluded from the "specifically accessed remotely by a component of Windows" scope are all generic access by browsing tools such as *ldp.exe* that can access any attribute of any object in the directory. If *ldp.exe* or a similar tool covered by a Windows license can display or even modify a **non-replicated attribute** of an object using only the attribute's **syntax** as defined by the schema, that does not make the attribute part of the state model. If *ldp.exe* or a similar tool covered by a Windows license accesses a non-replicated attribute and decodes or encodes its value using information outside the attribute's syntax as defined by the schema, that non-replicated attribute is included in the state model under category 1 (b) above.

2. Other state, however represented within Active Directory, is *abstracted* in the state model. Such state is included only as necessitated by the requirement that a licensee implementation of Windows Server protocols be able to receive messages and respond in the same manner as a Windows Server.

For instance, certain values sent by the Active Directory replication protocol [\[MS-DRSR\]](#) are accompanied by metadata. If the replicated values are stored by the receiving system, it must also store the metadata associated with the values. Otherwise, the receiving system will make incorrect responses to subsequent replication requests. These incorrect responses will, in general,

prevent replication from converging. So this metadata must be included within the state model. The specific way this metadata is stored by Active Directory, and the algorithms that optimize access to this metadata, are excluded from the state model.

The various indexes used by Active Directory's implementation to improve the performance of directory search are another example of state within Active Directory. These indexes have no effect, other than performance, on the protocol responses Active Directory makes. Therefore, these indexes are not included in the state model.

In this specification, the first category of state is modeled in a variant of **LDAP** information structures: **naming contexts**, objects, attributes, and values. These structures are defined precisely in the following sections. The set of replicated attributes is defined in Appendices [\[MS-ADA1\]](#), [\[MS-ADA2\]](#), [\[MS-ADA3\]](#). The set of non-replicated attributes covered under condition 1 (b) above consists of the [repsFrom](#) and [repsTo](#) attributes documented in [MS-DRSR] sections [5:repsFrom](#) and [5:repsTo](#).

**Note** Only the schema elements and instances of objects that are fundamental to Active Directory are described in this specification. If a protocol defines its own schema objects or otherwise creates its own objects in the directory, those objects are described in that protocol's specification. A summary of schema elements defined by such other protocols is included in [MS-ADA1], [MS-ADA2], [MS-ADA3], [\[MS-ADSC\]](#), and [\[MS-ADLS\]](#) as a convenience for the reader, but the documentation for the protocols using those schema elements should be consulted for a complete description.

In this specification, the second category of state is modeled using standard mathematical concepts. The concepts used and their associated notational conventions are described in the next section.

The **LDAP** protocol mandates very little about the behavior of a directory. Active Directory has many specific behaviors that are observable through **LDAP**. The remainder of this section describes the most pervasive of these behaviors. The remainder of the specification completes the picture.

### 3.1.1.1.2 State Modeling Primitives and Notational Conventions

Attribute names are underlined in this document, following the convention of [\[MS-DRSR\]](#). If a variable  $o$  refers to an object, and  $\underline{a}$  is an attribute name, then  $o!\underline{a}$  denotes the value or values of attribute  $\underline{a}$  on object  $o$ . If attribute  $\underline{a}$  is not present on  $o$ , the value of  $o!\underline{a}$  is null.

The specification uses the **LDAP** display names of attributes and **object classes** when referring to specific attributes and object classes. So if  $o$  refers to an object,

$o!\underline{\text{name}}$

denotes the [name](#) attribute of object  $o$ .

Some attributes in this specification are abstract in the sense of [MS-DRSR] section [3.3.3](#). Abstract attribute names are also underlined, for example, [repsFrom](#). RootDSE attribute names are also underlined, for example, [dumpDatabase](#), even though rootDSE attributes are not declared as attributes in the schema.

This specification models state in category 2 from the previous section using the standard mathematical concepts of set, sequence, directed graph, and tuple.

The notation  $[\text{first} \dots \text{last}]$  stands for the *subrange* first, first+1, ..., last. The type *byte* is the subrange  $[0.. 255]$ .

A *sequence* is an indexed collection of variables, called the *elements* of the sequence. The elements all have the same type. The *index type* of a sequence is a zero-based subrange.  $S[i]$  denotes the

element of the sequence *S* corresponding to the value *i* of the index type. The number of elements in a sequence *S* is denoted *S*.length. Therefore the index type of a sequence *S* is [0 .. *S*.length-1].

A fixed-length sequence can be constructed using the notation:

*[first element, second element, ... , last element]*

A *tuple* is a set of name-value pairs: [name<sub>1</sub>: value<sub>1</sub>, name<sub>2</sub>: value<sub>2</sub>, ... , name<sub>n</sub>: value<sub>n</sub>] where name<sub>k</sub> is an identifier and value<sub>k</sub> is the value bound to that identifier. Tuple types are defined as in this example:

- type **DSName** = [dn: **DN**, guid: **GUID**, sid: **SID**]

This defines **dsname** as a type of tuple with a DN-valued field *dn*, a **GUID**-valued field *guid*, and an **SID**-valued field *sid*.

### 3.1.1.1.3 Basics, objectGUID, and Special Attribute Behavior

The **LDAP** data model is defined by [\[RFC3377\]](#). Because the **LDAP** RFCs and their underlying ITU specifications have been interpreted in a variety of ways, this section defines a more specific model that correctly represents the behavior of Active Directory objects and attributes, and describes the correspondence between this model and the **LDAP** model.

The model is based on the general definitions Replica, Object, and Attribute given in the Pervasive Concepts section of the Glossary, section [1.1.1](#), and repeated here for convenience:

A *replica* is a variable containing a set of objects.

An *attribute* is an identifier for a set of values.

An *object* is set of attributes, each with its associated values. Two attributes of an object have special significance:

- *Identifying attribute*. A designated single-valued attribute appears on every object; the value of this attribute identifies the object. For the set of objects in a replica, the values of the identifying attribute are distinct.
- *Parent-identifying attribute*. A designated single-valued attribute appears on every object; the value of this attribute identifies the object's parent. That is, this attribute either contains the value of the parent's identifying attribute, or contains a reserved value identifying no object. For the set of objects in a replica, the values of this parent-identifying attribute define an **oriented tree** with objects as vertices and child-parent references as directed edges, with the child as an edge's tail and the parent as an edge's head.

Note that an object is a value, *not* a variable; a replica is a variable. The process of adding, modifying, or deleting an object in a replica replaces the entire value of the replica with a new value.

As the word replica suggests, it is often the case that two **replicas** contain "the same objects." In this usage, objects in two replicas are considered "the same" if they have the same value of the identifying attribute and if there is a process in place (replication) to converge both the set of objects in existence and the values of the non-identifying attributes as **originating updates** take place in replicas. When the members of a set of replicas are considered to be the same, it is common to say "an object" as a shorthand referring to the set of corresponding objects in the replicas.

A *child* object is an object that is not the root of its oriented tree. The *children* of an object *o* is the set of all objects whose parent is *o*.

The directory model used in this specification instantiates the preceding definitions as follows. The identifying attribute is [objectGUID](#) and the parent-identifying attribute is [parent](#), an abstract attribute. Both attributes have **GUID** values. No actual object has [objectGUID](#) equal to the **NULL GUID** (all zeros), and the root object has [parent](#) equal to the **NULL GUID**.

This specification uses the following s-expression representation ([LISP15]) of directory values, attributes, objects, and replicas to provide a notation for examples.

- Represent an attribute and its values as a list (*Attr* *Val*<sub>1</sub> *Val*<sub>2</sub> ... *Val*<sub>n</sub>) where *Attr* is an atom whose name is the attribute's name (its [LDAPDisplayName](#), defined in section 3.1.1.2) and each *Val*<sub>k</sub> is a value. The attribute comes first but the ordering of values in the list is not significant, with the exception of the values of the [objectClass](#) attribute explained below. If a value is a **GUID**, represent it as a 128-bit unsigned integer instead of using a representation that reflects the internal structure of a **GUID**. To aid the readability of examples, the **GUIDs** used in examples are unrealistically small integers.
- Represent an object as a list (*Attrval*<sub>1</sub> *Attrval*<sub>2</sub> ... *Attrval*<sub>n</sub>) where each *Attrval*<sub>k</sub> is the representation of an attribute and its values; the ordering of this list is not significant.
- Represent a replica as a list (*Obj*<sub>1</sub> *Obj*<sub>2</sub> ... *Obj*<sub>n</sub>) where each *Obj*<sub>k</sub> is the representation of an object; the ordering of this list is not significant.

The following list

```
(
  ( (objectGUID 5) (parent 0) (dc "microsoft") )
  ( (objectGUID 2) (parent 5) (ou "NTDEV") )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston") )
)
```

is one representation of the value of some replica containing three objects. The object with [objectGUID](#) = 5 is the root, the object with [objectGUID](#) = 2 is the only child of the root, and the object with [objectGUID](#) = 9 is the only grandchild of the root. Each object in this example has one additional attribute whose meaning has not yet been described.

Representing an attribute as its [LDAPDisplayName](#) makes examples readable. In the actual state model, an attribute is identified by an **ATTRTYP**. An **ATTRTYP** is a 32-bit unsigned integer that can be mapped to and from an object representing an attribute. This mapping is specified in section 3.1.1.2.6.

Active Directory's [objectGUID](#) attribute has special behavior. A fresh **GUID** is assigned to the [objectGUID](#) attribute of an object during its creation (LDAP Add) and this attribute is read-only thereafter. This is the first of many examples of an attribute with special behavior. Section 3.1.1.5 specifies the behavior of every attribute that has special behavior.

Active Directory includes the [objectSid](#) attribute on certain objects, called **security principal objects**. The [objectSid](#) attribute has special behavior. A fresh SID is assigned to the [objectSid](#) attribute of an object during its creation (LDAP Add) and this attribute is read-only thereafter, unless the object moves to another NC (LDAP Modify DN; see section 3.1.1.5 for the specification of such moves). More on [objectSid](#) generation in section 3.1.1.1.5.

#### 3.1.1.1.4 objectClass, RDN, DN, Constructed Attributes

A directory object is constrained by the directory's **schema**, which is a set of predicates. A few schema concepts need to be mentioned here. A full understanding of these concepts is not required to understand this section; additional information is available in the Glossary or section [3.1.1.2](#).

When an object is created, it is assigned a **most specific structural object class**, plus the sequence of object classes this class inherits from. The set of inherited classes always includes the class [top](#). The value of an object's [objectClass](#) attribute is the full set of object classes (each identified by [LDAPDisplayName](#)) assigned to the object. Elaborating the previous example, the following list

```
(
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS) )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit) )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user) )
)
```

represents three objects including their first and last [objectClass](#) values. The intermediate [objectClass](#) values are elided. Unlike all other multi-valued attributes, the ordering of [objectClass](#) values is significant - [top](#) is always listed first; the most specific structural object class is always listed last. So, for instance, the most specific structural object class of the root is [domainDNS](#).

Representing a class as its [LDAPDisplayName](#) makes examples readable. In the actual state model, a class is identified by an ATTRTYP. An ATTRTYP is a 32-bit unsigned integer that can be mapped to and from the schema object representing a class. This mapping is specified in section [3.1.1.2.6](#).

In Active Directory, each object has an **RDN attribute**, which is determined by the most specific structural object class of the object when the object is created. The RDN attribute is the attribute that defines an object's name relative to its parent. In Active Directory, the RDN attribute of an object class has String(Unicode) syntax, i.e., its value is a **Unicode** string, and the RDN attribute of an object always has exactly one value. (See section [3.1.1.2](#) for more on the topic of **attribute syntax**.)

Confusingly, the Active Directory schema includes an attribute whose [attributeSchema](#) object's [cn](#) is "RDN"; this is the [name](#) attribute; described below. The term "RDN attribute" never refers to the [name](#) attribute in this document.

The **RDN** of an object is a string of the form "att=val" where *att* is the [LDAPDisplayName](#) of the RDN attribute of the object and *val* is the value of the RDN attribute on this object. In the example above, the object class [user](#) has RDN attribute [cn](#), as can be confirmed by consulting [\[MS-ADSC\]](#). Therefore the RDN of the object with [objectGUID](#) = 9 is "cn=Peter Houston". An RDN can also be written using the [attributeID](#) of the RDN attribute in place of its [LDAPDisplayName](#); the example just given becomes "2.5.4.3=Peter Houston". The RDN form based on [LDAPDisplayName](#) is used throughout this document.

Active Directory requires that the value parts of the RDNs of all **children** of an object be distinct. This guarantees that the RDNs of all children of an object are distinct.

The DN of an object is defined recursively as follows. The DN of the root has an assigned value; the way Active Directory assigns this value is described later in section [3.1.1.1.5](#). The DN of a **child object** is the RDN of the child, followed by "," and the DN of the parent. In the example above,



suppose the assigned DN of the root object is "dc=microsoft,dc=com". Then the DN of the object with [objectGUID](#) = 9 is "cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com".

The correspondence between this model and the **LDAP** data model is as follows. An object with its attributes and values corresponds to an **LDAP entry** with its attributes and values. This model and **LDAP** agree on the definition of the [objectClass](#) attribute. The definition of RDN in this model is a subset of **LDAP's** definition; all RDNs in this model are valid **LDAP** RDNs, but not vice versa. Given the RDN definition, the definition of DN in this model is the same as **LDAP's** definition. In the **LDAP** data model the child-parent relationship is represented in the DNs of the child and parent, whereas in the Active Directory data model the child-parent relationship is represented in the [parent](#) attribute and the DN is derived. Active Directory does not expose the model's [parent](#) attribute through **LDAP**.

Active Directory includes the [distinguishedName](#) attribute on every object; the value is the object's DN. Elaborating the previous example to include a value of [distinguishedName](#) on each object:

```
(
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (distinguishedName "dc=microsoft,dc=com") )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit)
    (distinguishedName "ou=NTDEV,dc=microsoft,dc=com" ) )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user)
    (distinguishedName
      "cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com") )
)
```

But including [distinguishedName](#) on each object this way is misleading because the [distinguishedName](#) attribute is not stored as a string on each object. If it were stored as a string on each object, renaming an object would require updating every object in the subtree rooted at the renamed object. For a large subtree this would take a long time, and would either interfere with other directory activity (if performed as a single transaction) or would expose observable inconsistency to clients (if performed as multiple transactions). Active Directory does neither of these, so its state model can't imply that it does.

The [distinguishedName](#) attribute is not declared in the schema as a **constructed attribute** but in the state model it behaves like one. Normal attributes, including attributes with special behavior such as [objectGUID](#), have their values stored as part of an object's representation. **Constructed attributes** have values computed from normal attributes (for read) and/or have effects on the values of normal attributes (for write). The distinction between normal and **constructed attributes** is part of the state model, because only normal attributes replicate. Therefore **constructed attributes** are modeled only in the schema (section [3.1.1.2](#)), not in the per-object state model. Section [3.1.1.4](#) specifies all of Active Directory's **constructed attributes**. Because [distinguishedName](#), like a **constructed attribute**, adds no per-object state to the state model, it is not shown in later state model examples.

Active Directory includes the [name](#) attribute on every object. An object's value of [name](#) equals the value of the object's RDN attribute. Removing the incorrect modeling of [distinguishedName](#) from the previous example, then elaborating the example to include [name](#):

```
(
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (name "microsoft") )
)
```

```
(
  (objectGUID 2) (parent 5) (ou "NTDEV")
  (objectClass top ... organizationalUnit)
  (name "NTDEV") )
  (objectGUID 9) (parent 2) (cn "Peter Houston")
  (objectClass top ... user)
  (name "Peter Houston") )
)
```

The [name](#) attribute has special behavior. Even if an object is renamed (LDAP Modify DN), the object's [name](#) attribute remains equal to the object's RDN attribute.

Because Active Directory requires that the value parts of the RDNs of all children of an object be distinct, it follows that the [name](#) attribute of all children of an object are distinct.

Active Directory includes the [rdnType](#) attribute on every object. An object's value of [rdnType](#) is the object's RDN attribute at object creation time - the identifier, not its associated value. Elaborating the example to include [rdnType](#):

```
(
  (objectGUID 5) (parent 0) (dc "microsoft")
  (objectClass top ... domainDNS)
  (name "microsoft") (rdnType dc))
  (objectGUID 2) (parent 5) (ou "NTDEV")
  (objectClass top ... organizationalUnit)
  (name "NTDEV") (rdnType ou))
  (objectGUID 9) (parent 2) (cn "Peter Houston")
  (objectClass top ... user)
  (name "Peter Houston") (rdnType cn))
)
```

The [rdnType](#) attribute, like the [parent](#) attribute, is not declared in the Active Directory schema. [MS-DRSR] section [5:rdnType](#), specifies the special behavior of the [rdnType](#) attribute.

### 3.1.1.1.5 NC, NC Replica

The type DSNAME is defined as a C structure in [MS-DRSR] section [5:DSNAME](#); this state model uses the simpler *dsname*, which contains the same information in a tuple of the form

*dsname*: [*dn*: DN; *guid*: GUID; *sid*: SID]

When a *dsname* is stored within the state model, it is formatted as specified in [MS-DRSR] section [5.15.2.1](#).

An NC is a *dsname* containing a non-NULL *dn* and non-NULL **GUID**, used in forming names for a tree of objects in one or more replicas. These replicas are called NC replicas. Given an NC *x*, it is convenient to say "the replicas of *x*" to refer to *x*'s NC replicas.

A replica of NC *x* is a replica as already defined, with its root object *r* constrained as follows:

- *r*![objectGUID](#) = *x*.*guid*
- *r*![distinguishedName](#) = *x*.*dn*
- if *x*.*sid* ≠ NULL then *r*![objectSid](#) = *x*.*sid*, otherwise *r*![objectSid](#) has no value

Mutation of a general replica is unconstrained. In a replica of a specific NC the root object cannot be replaced, because doing so would change the [objectGUID](#) (and [objectSid](#) if any), and this must equal the NC's *guid*. In a replica of a given NC the root object's DN cannot be changed, because the root object's DN must equal the NC's *dn*.

All replicas in Active Directory are NC replicas, not general replicas.

NC replicas are mutable via several protocols including **LDAP**. The term *originating update* means any mutation to an NC replica performed via any protocol except replication.

Active Directory performs *replication* between replicas of the same NC to converge their states, so an update originated on one replica is reflected in all the others. The replication algorithm has the property that if originating updates to all replicas ceases and communication between replicas is maintained, the application-visible states of the replicas will eventually converge to a common value. Applications of Active Directory can read from several replicas of a given NC and observe the differences, but applications typically bind to a single replica.

Active Directory supports four NC types:

1. *Domain NC*. An NC that represents a security **domain**. The [objectSid](#) attribute exists only on objects within a **domain** NC. The *sid* field of a **domain** NC is not NULL. **AD/DS** supports **domain** NCs, **AD/LDS** does not.
2. *Application NC*. An NC that does not represent a security **domain**. Compared with **domain** NCs, application NCs and their replicas can be created more freely. The *sid* field of an application NC is NULL.
3. *Config NC*. An NC that stores Active Directory configuration information. The *sid* field of a config NC is NULL.
4. *Schema NC*. An NC that stores Active Directory schema information. The *sid* field of a schema NC is NULL.

The *dn* of a **domain** NC or an AD/DS application NC takes the form:

$$dc=n_1, dc=n_2, \dots dc=n_k$$

where each  $n_i$  satisfies the syntactic requirements of a DNS name component [RFC1034]. Such a DN corresponds to the DNS name:

$$n_1. n_2. \dots .n_k$$

This is the *DNS name of the NC*. The mapping just specified follows [RFC2247].

In AD/LDS, an application NC can have any valid DN, therefore an AD/LDS application NC does not necessarily have a DNS name.

Replicas of a **domain** NC have one of these two sub-types:

1. *Full*. A replica whose objects contain their full state as defined by all originating writes.
2. *Partial*. A replica whose objects contain a filtered view of the full state as defined by all originating writes. There are three types of the partial replica:
  1. GC partial NC replica: the **filter** removes all attributes (and their values) that are not in the partial replica's **GC partial attribute set**.

2. Filtered partial NC replica: the filter removes all the attributes (and their values) that are in the **filtered attribute set**. The **default NC**, **config NC**, and **application NC** on a RODC are filtered partial NC replicas.
3. Filtered GC partial NC replica: the filter removes all the attributes (and their values) that are not in the partial replica's **partial attribute set**, as well as all the attributes (and their values) in the **filtered attribute set**. The non-default **domain** NCs hosted on a RODC are filtered GC partial NC replicas.

Replicas of other NC types are always full. A full replica is either *writable*, that is, accepts originating updates, or is read-only. A partial replica is read-only.

This section has introduced many concepts without describing how they are reflected in the state model. To a great extent this obligation will be discharged in other sections of this document. The schema NC is described in section [3.1.1.2](#), while the other NC types are described in section [7.1](#). Here are three elaborations of the state model that can be explained without making a forward reference:

1. NC replicas are modeled by making a *dsname*, converted into a string formatted as specified in [MS-DRSR] section 5.15.2.1, the first element of a replica.
2. The root object of a **domain** NC or an AD/DS application NC has class [domainDNS](#). The RDN attribute of [domainDNS](#) is [dc](#). Therefore both the [dc](#) and [name](#) attributes of the root object of a **domain** NC or an AD/DS application NC equal the first component of the NC's DNS name. The root object of an AD/LDS application NC can have any object class except [dmd](#) or [configuration](#).
3. In AD/DS, the generation of [objectSid](#) values is constrained by the *sid* of a **domain** NC as follows. The *sid* of a **domain** NC, the *domain SID*, is a SID with four subauthorities. The root object of a **domain** NC has [objectSid](#) equal to the **domain** SID, as required by the definition of NC replica. Every **security principal** object *o* in a **domain** NC has *o![objectSid](#) equal to the **domain** SID plus the RID portion (that is, it has five subauthorities). The RID portion of *o![objectSid](#) is a number not assigned as the RID portion of the [objectSid](#) to any other object of the **domain**, including objects that existed earlier but have been deleted.**

Section [3.1.1.5.2.4](#) specifies how AD/DS assigns RIDs. The same section specifies how AD/LDS generates [objectSid](#) values for new AD/LDS **security principals**.

Continuing the example, let the example NC be a **domain** NC, and let the object with name "Peter Houston" be assigned the RID value 2055 (decimal). Then the state of the example NC is:

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;
  dc=microsoft,dc=com"
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (name "microsoft") (rdnType dc)
    (objectSid 0x0105...94E1F2E6) )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit)
    (name "NTDEV") (rdnType ou) )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user)
    (name "Peter Houston") (rdnType cn)
    (objectSid 0x0105...94E1F2E607080000) )
)
```

The DNS name of this **domain** NC is *microsoft.com*. Note that the **domain** SID is a prefix of the "Peter Houston" object's [objectSid](#). Portions of the (long) SID values have been elided for clarity; consider the elided portions to be the hex digits:

```
0000000000051500000089598D33D3C56B68
```

and the example SID will be a valid SID.

### 3.1.1.1.6 Attribute Syntaxes, Object References, Referential Integrity, and Well-Known Objects

The complete set of attribute syntaxes supported by Active Directory is specified in section [3.1.1.2](#). The model representation of values of each attribute syntax is specified in [MS-DRSR] section [5.15.2](#). The model representation of each syntax can be returned in an **LDAP** response without conversion.

The following five attribute syntaxes are called **object reference** syntaxes:

- Object(DS-DN)
- Object(DN-String)
- Object(DN-Binary)
- Object(Access-Point)
- Object(OR-Name)

Attributes with object reference syntaxes have special behavior, called *referential integrity*. No other attribute syntaxes have special behavior intrinsic to the syntax. The referential integrity behavior applies only to the object reference portion of the syntax. The Object(DS-DN) is a single object reference; the other four syntaxes combine a single object reference with other information that has no special behavior intrinsic to the syntax. So it suffices to specify the referential integrity behavior of an Object(DS-DN) attribute.

To specify referential integrity, some background on object deletion is needed; object deletion is specified fully in section [3.1.1.5](#). Object deletion (the **LDAP** Delete operation) is performed in two stages. In the first stage, the deleted object is converted to a **tombstone**. A tombstone is a special object, part of a replica's state. The state of a deleted object's tombstone resembles the state of the object before deletion; it has the same [objectGUID](#), but a different DN. A tombstone is *not* an object from the **LDAP** perspective: A tombstone is not returned by a normal **LDAP** Search request, only by a Search request with extended control LDAP\_SERVER\_SHOW\_DELETED\_OID as described in section [3.1.1.3](#). After a significant delay (the **tombstone lifetime**), a tombstone is *garbage collected*, which removes it from the replica's state.

In situations where a deletion does not need to be replicated, an object is **expunged** (that is, removed in a single step from the replica's state) instead. Expunge is used to remove **lingering objects** (section [3.1.1.3.3.14](#)), to remove the object being moved during cross-domain move (section [3.1.1.5.4.2](#)) and to delete a **dynamic object** (section [7.1.7](#)).

An application is not limited to specifying a DN when creating an object reference; using the syntax specified in section [3.1.1.2](#), it can specify a *dsname* containing any combination of values as long as it specifies at least one. The *dsname* is resolved to an object using *dsname* equality as defined in [MS-DRSR] section 5: **DSNAME**.

The state kept with an attribute to represent an object reference is a *dsname*, using the representation specified in [\[MS-DRSR\]](#) section 5.15.2.1.

When reading an object reference, an application can request the full *dsname* instead of a DN by passing the LDAP\_SERVER\_EXTENDED\_DN\_OID extended control as described in section [3.1.1.3](#).

A single-valued Object(DS-DN) attribute *a* on object *src* behaves as follows:

- When an **LDAP** Add or Modify creates an object reference within attribute *src.a*, the server uses the DN (or *dsname* with at least one field specified) specified in the Add or Modify to locate an existing object *dst*. If no such object exists, including the case where the object has been deleted and exists as a tombstone, the request fails with **LDAP** error *noSuchObject*. The values *dst!distinguishedName*, *dst!objectGUID* and *dst!objectSid* are used to populate the *dsname* representing the object reference within attribute *src.a*. If the object *dst* has no [objectSid](#) attribute, the "SID=" portion of the *dsname* representation is omitted.
- If object *dst* has not been deleted, reading attribute *a* gives the *dsname* of object *dst*, even if *dst* has been renamed since *a* was written.
- If the object *dst* has been deleted or expunged, reading *src.a* gives a *dsname* with a *dn* field that corresponds to no object. Either this DN is impossible to create via **LDAP** Add and **LDAP** Modify DN, or this DN changes (i.e., the value of *src.a* changes) when an **LDAP** Add or Modify DN would give some other object this DN.

The multi-valued case is similar; a multi-valued attribute is capable of containing multiple object references that behave as described.

Each object reference syntax exists in two versions. The description just given is for the *non-link* version. The other version is the *forward link*. The Object(DS-DN) syntax also exists in a *back link* version.

A **forward link** Object(DS-DN) **attribute** supports the definition of a corresponding **back link** Object(DS-DN) **attribute**. The back link attribute is a read-only **constructed attribute**; writes to the back link attribute are disallowed. If an object *o* contains a reference to object *r* in forward link attribute *f*, and there exists a back link attribute *b* corresponding to *f*, then a **back link value** referencing *o* exists in attribute *b* on object *r*. The correspondence between the forward and back link attributes is expressed in the schema; see section [3.1.1.2](#) for details. A forward link attribute can exist with no corresponding back link attribute, but not vice versa.

If the syntax of a forward link attribute is not Object(DS-DN), a corresponding back link attribute has syntax Object(DS-DN), *not* the syntax of the forward link. The non-reference portion of the forward link, if any, is ignored in computing the back link. If ignoring the non-reference portion of the forward link results in duplicate back references, the duplicates are present in the values of the back link attribute.

The referential integrity behavior of a forward link attribute differs from that of a non-link attribute as follows:

- When an object *o* is deleted or expunged, any forward link reference to *o* is removed from the attribute that contains it.

Since a back link attribute is constructed, its referential integrity behavior follows from that of the corresponding forward link attribute.

The distinction between non-link and forward link references is not visible in the part of the state model described in this section; it is a schema difference only. There is no difference in the state

kept with an attribute to represent the object reference. There is a difference in the replication metadata accompanying the object reference, as will be described in section [3.1.1.1.9](#).

The behavior described in this section is for object references within a single NC replica. Additional behaviors, specified in section [3.1.1.1.12](#), are possible when an object reference crosses an NC replica boundary.

Extend the running example by adding a [group](#) object named "DSYS" as a child of "ou=NTDEV,dc=microsoft,dc=com". The object class [group](#) includes the attribute [member](#) with Object(DS-DN) syntax. In this example the "DSYS" group has the [user](#) object "Peter Houston" as its only [member](#).

```
(
  "<GUID=5>;<SID=0x0105...00000000>;dc=microsoft,dc=com"
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (name "microsoft") (rdnType dc)
    (objectSid 0x0105...94E1F2E6) )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit)
    (name "NTDEV") (rdnType ou) )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user)
    (name "Peter Houston") (rdnType cn)
    (objectSid 0x0105...94E1F2E607080000) )
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (member
      "<GUID=9>;<SID=0x0105...07080000>;
      cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com" ) )
)
```

Note that the [group](#) "DSYS" is a **security principal** object within the **domain** NC, with the distinct RID value 2059 (decimal).

The root object of each NC contains the attribute [wellKnownObjects](#). The purpose of this attribute is to provide a location-independent way to access certain objects within the NC. For instance, the "lost and found" **container** where most tombstones live can be found using [wellKnownObjects](#).

The [wellKnownObjects](#) attribute has syntax Object(DN-Binary). Each value consists of an object reference *ref* and a byte string *binary* that is 16 bytes long. The byte string *binary* contains a **GUID** identifying a **well-known object** within an NC; the object reference *ref* is a reference to the corresponding object. A table of the **GUIDs** that identify well-known objects is given in section [7.1.1.4](#).

The following procedure implements well-known object location using the [wellKnownObjects](#) attribute. This procedure will be used throughout the rest of this specification.

- procedure GetWellknownObject(*nc*: NC, *guid*: GUID): DSName
  - If there is no replica of NC *nc* on the server executing this procedure, return null.
  - Let *v* be the value of *nc*![wellKnownObjects](#) on the server's replica satisfying *v.binary* = *guid*; if no such *v* exists return null.



- Return *v.ref*.

Assignments to the [wellKnownObjects](#) attribute are specially checked as described in section [3.1.1.5](#).

The **LDAP** protocol supports access to well-known objects using an extended *dsname* syntax as described in section [3.1.1.3](#).

### 3.1.1.1.7 Forest, Canonical Name

An Active Directory **forest** is a set of NCs. Every forest contains one schema NC and one config NC. The other types of NCs in a forest depends on whether it is an AD/DS forest or an AD/LDS forest:

- AD/DS: Every forest also contains one or more **domain** NCs, and zero or more application NCs.
- AD/LDS: Every forest also contains zero or more application NCs.

The NCs within a forest are related by their assigned DN as follows:

- In AD/DS there must exist a **domain** NC *root* such that the config NC's *dn* equals *Cat("cn=Configuration,", root.dn)* (where *Cat* is the string concatenation function). This unique **domain** NC is called the **root domain** NC of the forest.

Describe this DN relationship as "The config NC is a child of the root domain NC." Technically these NCs are not related in the same way as a child object and its **parent object** are related within an NC; the [parent](#) relationship stops at the root of an NC. But their DNs are related in the same way as the DNs of a child object and its parent object within an NC.

In AD/LDS, the config NC does not have a parent NC. An AD/LDS forest contains no **domain** NCs, so there is no forest root domain NC, either. The DN of an AD/LDS config NC takes the form "CN=Configuration, CN={G}" where G is a **GUID** in dashed-string form ([\[RFC4122\]](#) section 3). For example,

CN=Configuration, CN={FD783EE9-0216-4B83-8A2A-60E45AECCB81}

is a possible DN of the config NC in an AD/LDS forest.

- The schema NC is a child of the config NC, with RDN "cn=Schema".
- If *short* and *long* are NCs with DNS names (domain NCs or application NCs), and *short* is a suffix of *long*, then each DNS name obtained by removing DNS name components successively from the front of *long* until the result is *short* must also name NCs with DNS names. For instance, if a forest contains both NCs *microsoft.com* and *nttest.ntdev.microsoft.com*, it must also contain NC *ntdev.microsoft.com*.
- If *app* is an application NC and *dom* is a **domain** NC, then *dom* must not be a child of *app*.
- If *root* is the root **domain** NC and *dom* is another **domain** NC in the forest, then *root* must not be a child of *dom*.

Extend the running example by adding the config NC and schema NC:

```
(
  "<GUID=4>;cn=Configuration,dc=microsoft,dc=com"
  ( (objectGUID 4) (parent 0) (cn "Configuration")
    (objectClass top ... configuration)
    (name "Configuration") (rdnType cn) )
)
```



```

)
(
  "<GUID=8>;cn=Schema,cn=Configuration,dc=microsoft,dc=com"
  ( (objectGUID 8) (parent 0) (cn "Schema")
    (objectClass top ... dMD)
    (name "Schema") (rdnType cn) )
)
(
  "<GUID=5>;<SID=0x0105...00000000>;dc=microsoft,dc=com"
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (name "microsoft") (rdnType dc)
    (objectSid 0x0105...94E1F2E6) )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit)
    (name "NTDEV") (rdnType ou))
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user)
    (name "Peter Houston") (rdnType cn)
    (objectSid 0x0105...94E1F2E607080000) )
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (members
      "<GUID=9>;<SID=0x0105...07080000>;
      cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com" ) )
)

```

This example illustrates the *dn* relationships between the root **domain** NC, config NC, and schema NC. It shows that in a forest, the [parent](#) relationship does not cross NC boundaries. It also illustrates the object classes of the config NC and schema NC root objects, and the lack of a *sid* in these NCs. It does not show the contents of these NCs, which are specified in sections [7.1](#) and [3.1.1.2](#).

Every object in a forest has a **canonical name**. The canonical name of an object is a syntactic transformation of its DN into something resembling a pathname that still identifies the object. A canonical name is a DNS name, followed by a "/", followed by a sequence of zero or more names separated by "/". The DNS name is the translation of the final sequence of "dc=" DN components into an equivalent DNS name (following [RFC2247]). The sequence of names is the sequence of names in the non-"dc=" DN components, appearing in the reverse order to the order they appeared in the DN. Here are several examples of this translation drawn from the example above:

```

DN:          cn=Peter Houston, ou=NTDEV, dc=microsoft,
             dc=com
canonical name: microsoft.com/NTDEV/Peter Houston
DN:          cn=Configuration, dc=microsoft, dc=com
canonical name: microsoft.com/Configuration
DN:          dc=microsoft, dc=com
canonical name: microsoft.com/

```

Active Directory supports a **constructed attribute** [canonicalName](#) on every object. Its value is the object's canonical name.

### 3.1.1.1.8 GC

In AD/DS, the **global catalog (GC)** is a partial view of a forest's NCs, with these properties:

- The GC view includes **domain** NCs, the config NC, and the schema NC.
- The GC view is partial. It includes all objects in the included NCs, but only those attributes defined as members of the partial attribute set in the schema NC (as specified in section [3.1.1.2](#)). If the GC is an RODC, the attribute list is further restricted to those attributes not present in the filtered attribute set in the schema NC (as specified in section [3.1.1.2](#)).
- The GC view is read-only.

The GC has no state model impact outside the schema NC, which defines the forest's partial attribute set. The implementation of the GC (that is, actually providing the specified view to **LDAP** clients) does have impact, explained in section [3.1.1.1.9](#).

In AD/LDS there is no support for the GC.

### 3.1.1.1.9 DCs, usn Counters, and the Originating Update Stamp

The model defines the state of a **domain controller (DC)** as a tuple of type DC:

```
type DC = [  
  serverGuid: GUID,  
  usn: 64-bit integer,  
  prefixTable: PrefixTable,  
  defaultNC: full domain NC replica,  
  configNC: config NC replica,  
  schemaNC: schema NC replica,  
  partialDomainNCs: set of partial domain NC replica,  
  appNCs: set of application NC replica,  
  pdcChangeLog: PDCChangeLog  
  nt4ReplicationState: NT4ReplicationState  
  ldapConnections: LDAPConnections,  
  replicationQueue: ReplicationQueue,  
  kccFailedConnections: KCCFailedConnections,  
  kccFailedLinks: KCCFailedLinks,  
  rpcClientContexts: RPCClientContexts,  
  rpcOutgoingContexts: RPCOutgoingContexts  
]
```

The variable **dc** is the only global variable in this specification. It contains the state of the DC:

```
dc: DC
```

*serverGuid* is initialized to a fresh **GUID** when the *dc* is created and does not change thereafter. Section [7.1.1.2.2.1.2.1.1](#) describes the *nTDSDSA* object; *serverGuid* equals the *objectGUID* of the DC's *nTDSDSA* object. *serverGuid* is independent of the specific machine playing the role of this DC.

*usn* is a counter used in assigning replication metadata to every originating update to an NC replica in the DC, as detailed later in this section. The *invocationId* of *dc*'s *nTDSDSA* object is an "epoch number" for *usn*; if an observer reads a *dc* at times  $t_1$  and  $t_2$  with  $t_1 < t_2$ , and *invocationId* is the same, then *usn* at time  $t_1$  is less than or equal to *usn* at time  $t_2$ .

*prefixTable* is the PrefixTable used to translate all ATTRTYP values stored in this DC's NC replicas; Section [3.1.1.2.6](#) specifies the translation process.

The **default NC** replica of an AD/DS DC, modeled as *dc.defaultNC*, is a full **domain** NC replica of some **domain** NC in the forest. This is the only full **domain** NC replica in the state of a DC. In an AD/LDS DC, *dc.defaultNC* is null.

The fields *dc.configNC* and *dc.schemaNC* contain replicas of the forest's config NC and schema NC.

If *dc* is not an AD/DS GC server (determined by state in the config NC as specified in Section [7.1.1.2.2.1.2.1.1](#)), then *dc.partialDomainNCs* is null. Otherwise it contains a partial **domain** NC replica for each **domain** NC in the forest, excluding the default **domain** NC of *dc*.

The field *dc.appNCs* contains replicas of any subset of the application NCs in the forest.

An AD/DS DC can be an RODC; [MS-DRSR] section [5:AmIRODC](#) specifies how this is determined by state in the config NC. All NC replicas of an RODC are read-only; that is, they do not accept originating writes. In other DCs, all NC replicas are writable except for *dc.partialDomainNCs*.

The *nt4ReplicationState* and *pdnChangeLog* variables contain state used by the IDL\_DRSGetNT4ChangeLog method ([MS-DRSR] section [4.1.11.3](#)). Section [3.1.1.7](#) specifies the format of these variables and how they are maintained during state changes in AD/DS.

The *ldapConnections*, *replicationQueue*, *kccFailedConnections*, *kccFailedLinks*, *rpcClientContexts*, and *rpcOutgoingContexts* fields of a DC are volatile state. Each volatile field is set to the empty sequence on server startup. The other fields are persistent state, updated using transactions.

The construction of the *kccFailedConnections* and *kccFailedLinks* fields of a DC are discussed in Section [7.2](#). The construction of the *replicationQueue*, *kccFailedConnections*, and *rpcOutgoingContexts* fields are discussed in [\[MS-DRSR\]](#).

Each originating update on a DC creates replication metadata - *AttributeStamp* and *LinkValueStamp* - values as will now be described.

*AttributeStamp* and *LinkValueStamp* values contain times read from the system clock of the server creating the value. If clocks on different DCs disagree by a significant fraction of the tombstone lifetime, then it is probable that different DCs will eventually disagree about whether some objects have been deleted or not; see Section [3.1.1.1.15](#). DCs use Kerberos for mutual **authentication**, and Kerberos refuses to mutually authenticate two DCs whose clocks are more than 5 minutes out of sync. The tombstone lifetime is generally several months, so synchronization within 5 minutes is much better than required to avoid object lifetime issues.

The type *AttributeStamp* is defined authoritatively in [MS-DRSR] section [5:AttributeStamp](#). In summary, it is the following tuple:

```
AttributeStamp: [  
    dwVersion: 32-bit Integer;  
    timeChanged: 64-bit number of seconds  
                  since January 1, 1601, 12:00:00am;  
    uuidOriginating: GUID;  
    usnOriginating: 64-bit Integer]
```

Similarly, the type *LinkValueStamp* is defined authoritatively in [MS-DRSR] section [5:LinkValueStamp](#). In summary, it is an *AttributeStamp* tuple extended on the bottom with the following fields:

- *timeCreated*: 64-bit number of seconds since January 1, 1601, 12:00:00am;
- *timeDeleted*: 64-bit number of seconds since January 1, 1601, 12:00:00am;

An *AttributeStamp stamp* is associated with all replicated attributes, except forward link attributes updated when the **forest functional level** is greater than DS\_BEHAVIOR\_WIN2000, that have ever had values on an object. For forward link attributes updated when the forest functional level is greater than DS\_BEHAVIOR\_WIN2000, a *LinkValueStamp stamp* is associated with each value of the attribute, both current link values and tombstoned link values. More on tombstoned link values below.

Let *o!a.stamp* denote the *AttributeStamp* associated with replicated attribute *a* on object *o*. When an originating update creates or modifies replicated attribute *a* on object *o*, the value of *o!a.stamp* is determined as follows:

- *dwVersion*: If the attribute did not exist on this object before the originating update (that is, an **LDAP** Add operation of this object, or an **LDAP** Modify operation creating the initial value of this attribute on this object) *dwVersion* equals one. Otherwise *dwVersion* equals *o!a.stamp.dwVersion* before the update, plus one.
- *timeChanged*: The time of the originating update, according to the system clock on this DC.
- *uuidOriginating*: the [invocationId](#) of the dc's [nTDSDSA](#) object.
- *usnOriginating*: *dc.usn*.

Once a replicated attribute exists on an object, it will continue to exist for the lifetime of the object, in order to carry the **stamp**. If all values have been removed from the attribute, the attribute will be absent from the **LDAP** perspective, but it remains present in the state model in order to preserve the **stamp**. If a value is added to *o!a* and *o!a.stamp* exists, even if *o!a* had no values before the addition, the value of *o!a.stamp.dwVersion* is used as described above in creating the new **stamp's** *dwVersion*.

Let *o!a.r* denote a single link value *r* that is part of a replicated forward link attribute *a*, and let *o!a.r.stamp* denote the *LinkValueStamp* associated with this value. An originating update cannot modify a single link value *r* that is part of a forward link attribute, except to delete it or to re-create it. A link value *r* is deleted, but exists as a *tombstone*, if *r.stamp.timeDeleted* ≠ 0. When the current time minus *r.stamp.timeDeleted* exceeds the tombstone lifetime, the link value *r* is garbage-collected, that is, removed from its containing forward link attribute.

When an originating update creates a link value *r* of a forward link attribute *a* of object *o*, the *LinkValueStamp* *o!a.r.stamp* is computed as follows:

- *dwVersion*: 1
- *timeChanged*: The time of the originating update, according to the system clock on this DC.
- *uuidOriginating*: the [invocationId](#) of dc's [nTDSDSA](#) object.
- *usnOriginating*: *dc.usn*.
- *timeCreated*: The time of the originating update, according to the system clock on this DC.
- *timeDeleted*: Zeros.

When an originating update re-creates a link value *r* of a forward link attribute *a* of object *o*, that is, a create occurs when the same link value exists as a tombstone, the *LinkValueStamp* *o!a.r.stamp* is computed as follows:

- *dwVersion*: *o!a.r.stamp.dwVersion* before the originating update, plus one.
- *timeChanged*: The time of the originating update, according to the system clock on this DC.
- *uuidOriginating*: the [invocationId](#) of *dc*'s [nTDSDSA](#) object.
- *usnOriginating*: *dc.usn*.
- *timeCreated*: *o!a.r.stamp.timeCreated* before the originating update.
- *timeDeleted*: Zeros.

When an originating update deletes a link value *r* of a forward link attribute *a* of object *o*, the *LinkValueStamp* *o!a.r.stamp* is computed as follows:

- *dwVersion*: *o!a.r.stamp.dwVersion* before the originating update, plus one.
- *timeChanged*: The time of the originating update, according to the system clock on this DC.
- *uuidOriginating*: the [invocationId](#) of *dc*'s [nTDSDSA](#) object.
- *usnOriginating*: *dc.usn*.
- *timeCreated*: *o!a.r.stamp.timeCreated* before the originating update.
- *timeDeleted*: The time of the originating update, according to the system clock on this DC.

The **stamp** values created by originating updates are used by protocols described in [MS-DRSR]. Some **stamp** values maintained in this state model are not used by those protocols; see [MS-DRSR] section [4.1.10.5.5](#) (FilterAttribute) for specifics on the **stamps** that are filtered out.

When all updates associated with an originating update request are complete, the variable *dc.usn* is increased by at least one. Between originating updates, the variable *dc.usn* does not decrease.

The effects of an originating update are captured in the state model by committing a transaction before sending the response. The transaction has the ACID properties [GRAY] and provides at least degree 2 isolation of concurrent read and update requests [GRAY].

Each read request is performed as a transaction. When multiple read requests are used to retrieve a large set of results, each request is its own transaction. Section [3.1.1.5](#) specifies the transaction boundaries used for all originating updates. To preview: An originating update is almost always performed as a single transaction; a few are processed as multiple transactions. In some cases an originating update request will cause transactions to occur after the response has been sent; Section [3.1.1.5](#) specifies all cases where processing of an update continues after the response.

The following example illustrates the effects of originating updates on **stamp** values. In this example the forest functional level is assumed to be greater than DS\_BEHAVIOR\_WIN2000, so *LinkValueStamps* are used for updates to forward link attributes. In the example, **stamp** values are represented as lists whose elements are the elements of the **stamp**, in the order listed in the type definition. Thus *dwVersion* is always first, and *timeDeleted* is last in a *LinkValueStamp*. An *AttributeStamp* is placed between the attribute's [IDAPDisplayName](#) and the first value, if any. A *LinkValueStamp* is placed immediately following the link value.

This example shows the **stamp** values on two attributes of a single [group](#) object: the [description](#) attribute and the [member](#) attribute (a forward link attribute). In the initial state neither attribute is present:

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  .
  .
  .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
  )
)
```

An LDAP Modify adds a value for [description](#). This DC's [invocationId](#) is 103 and its *usn* is 501 at the time of the originating update.

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  .
  .
  .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (description (1 0x2FA9A74EA 103 501) "QWERTY")
  )
)
```

An LDAP Modify adds a value for [member](#). This originating update occurred one second after the previous one, with no updates in between. This pattern continues for the rest of this example.

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  .
  .
  .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (description (1 0x2FA9A74EA 103 501) "QWERTY")
    (member
      "<GUID=9>;<SID=0x0105...07080000>;
      cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com"
      (1 0x2FA9A74EB 103 502 0x2FA9A74EB 0) )
    )
)
```

An LDAP Modify removes the values of both [description](#) and [member](#):

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  .
  .
  .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (description (2 0x2FA9A74EC 103 503) )
    (member

```

```

    "<GUID=9>;<SID=0x0105...07080000>;
      cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com"
    (2 0x2FA9A74EC 103 503 0x2FA9A74EB 0x2FA9A74EC) )
  )
)

```

An LDAP Modify sets [member](#) back to the value it had before the previous update. The **stamp** it receives is *not* what it had before:

```

(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  . . .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (description (2 0x2FA9A74EC 103 503) )
    (member
      "<GUID=9>;<SID=0x0105...07080000>;
        cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com"
      (3 0x2FA9A74ED 103 504 0x2FA9A74EB 0) )
    )
  )
)

```

Finally an LDAP Modify sets [description](#) to a new value:

```

(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  . . .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (description (3 0x2fa9a74ee 103 505) "SHRDLU")
    (member
      "<GUID=9>;<SID=0x0105...07080000>;
        cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com"
      (3 0x2FA9A74ED 103 504 0x2FA9A74EB 0) )
    )
  )
)

```

### 3.1.1.1.10 GC Server

An AD/DS DC can be a **GC server** as determined by state in the config NC as specified in section [7.1.1.2.2.1.2.1.1](#)). A GC Server provides **LDAP** access to the GC view of the forest via a special **LDAP** port as specified in section [3.1.1.3](#).

### 3.1.1.1.11 FSMO Roles

Each DC accepts originating updates for most attributes of most objects within its **writable NC replicas**. But certain updates are only accepted if the DC is the single designated "master" DC for the update, as specified in this section. The mechanism is called **FSMO roles**, which stands for **Flexible Single Master Operations Roles**.

If some or all of the updates to an object are single-mastered, that object belongs to a defined set of objects. [MS-DRSR] section [4.1.10.5.3](#) (GetReplScope) specifies these sets, which are called *FSMO roles*. Each FSMO role is contained within a single NC. Each **domain** NC contains three FSMO roles called *InfrastructureMasterRole*, *RidAllocationMasterRole*, and *PdcEmulationMasterRole*. A config NC contains one FSMO role called *PartitionNamingMasterRole*. A schema NC contains one FSMO role called *SchemaMasterRole*. An application NC has no FSMO roles.

Since a DC operating as AD/LDS does not host **domain** NCs, it cannot own any of the three roles contained by **domain** NCs. It can own the Schema Master and Partition Naming FSMO roles.

In a given NC, each FSMO role is represented by an object. [MS-DRSR] section 4.1.10.5.3 (GetReplScope) specifies these objects, which are called *FSMO role objects*.

The [fSMORoleOwner](#) attribute of each **FSMO role object** is an object reference to the [nTDSDSA](#) object of the DC that owns the role, that is, performs updates to objects in the role. [nTDSDSA](#) objects and how they represent DCs are specified in section [7.1](#).

An originating update to an object within a FSMO role generates an **LDAP** referral if the DC that receives the request cannot perform the update; the referral is to the DC represented by the [nTDSDSA](#) object referenced by the FSMO role object's [fSMORoleOwner](#) attribute on the DC that received the request.

The processing of updates affected by FSMO roles is fully specified in section [3.1.1.5](#).

The IDL\_DRSGetNCChanges method ([MS-DRSR] section [4.1.10](#)) makes an originating write to the [fSMORoleOwner](#) attribute of a FSMO role object while preserving single-mastering of updates to the FSMO role. The ability to update the [fSMORoleOwner](#) attribute in this way is exposed through **LDAP** as the root DSE updates [becomeDomainMaster](#), [becomeInfrastructureMaster](#), [becomePdc](#), [becomePdcWithCheckPoint](#), [becomeRidMaster](#), and [becomeSchemaMaster](#) specified in section [3.1.1.3](#).

Reading the rootDSE attribute [validFSMOs](#) on a DC returns the set of all FSMO roles (represented as FSMO role objects) that the DC will update; this is specified in section [3.1.1.3](#).

### 3.1.1.1.12 Cross-NC Object References

Section [3.1.1.1.6](#) specifies the referential integrity behavior of attributes with object reference syntaxes. That section only specifies the case of references within a single NC. This section specifies the differences for the case of object references that cross an NC boundary.

Suppose *src* and *dst* are objects in different NCs, *src* has an attribute *a* with an object reference syntax, and *dc* is a DC hosting a writable replica of *src*'s NC.

- When an **LDAP** Add or Modify creates an object reference within attribute *src.a*, the server uses the DN (or *dsname* with at least one field specified) specified in the Add or Modify to locate an existing object *dst*. The behavior is identical to the single NC case, with two exceptions.
  1. Locating the object *dst* can fail because the dc does not host a replica of *dst* and dc fails to communicate with a server that hosts a replica; the response is **LDAP** error *unavailable*.
  2. Certain cross NC references are not allowed; the specific references that are not allowed are specified in section [3.1.1.2.2.3](#). If the reference is not allowed, the response is **LDAP** error *constraintViolation*.
- After the assignment, the referential integrity behavior is the same as if the reference did not cross an NC boundary, except that reference *src.a* reflects the state of object *dst* at some time *t* in the past, not at the current time. If the distributed system of DCs in the forest is functioning



normally, the difference between the current time and the time  $t$  of the previous sentence is bounded by an administrator-configurable amount of time. Functioning normally in the previous sentence means that the DCs are running and communicating as needed, with only transient failures.

The mechanism the system uses for restoring the integrity of object references is specified in section [3.1.1.6](#).

### 3.1.1.1.13 NC Replica Graph

This section uses directed graphs to model replication topology. Use [KNUTH1] section 2.3.4.2 as a reference for the terms *directed graph*, *vertex*, *arc*, *initial vertex*, *final vertex*, *path*, and *strongly-connected*.

This section introduces concepts that are used in specifying the Knowledge Consistency Checker (KCC) in section [7.2](#). The concepts are simplified here because this section ignores the SMTP replication transport (see [\[MS-SRPL\]](#)) and RODCs. Section [7.2](#) specifies the concepts in full generality.

Associated with each NC replica is a [repsFrom](#) abstract attribute as specified in [MS-DRSR] section [5:repsFrom](#). The value of this attribute is a set of tuples. Each tuple contains a field *uuidDsa* that contains the [objectGUID](#) of an [nTDSDSA](#) object. The [nTDSDSA](#) object represents a DC as specified in section [7.1](#).

Given a forest and an NC within the forest, define the **NC replica graph** as follows:

- Each DC of the given forest is a vertex of the directed graph
- For each DC  $d$  containing a replica of the given NC:
  - Set  $r$  to the given NC's [repsFrom](#) on the DC  $d$ , as a sequence in any order
  - For  $i$  in  $[0 .. r.length-1]$ :
    - $r[i].uuidDsa$  is a directed arc to  $d$  (the final vertex of the arc) from the DC represented by the [nTDSDSA](#) object with [objectGUID](#) =  $r[i].uuidDsa$  (the initial vertex of the arc)

Each arc in the directed graph represents a replication relationship. The DC at the final vertex of an arc performs **cycles** of IDL\_DRSGetNCChanges requests ([MS-DRSR] section [4.1.10.1](#)) to the DC at the initial vertex of that arc, applying the results of these requests to update the replica of the given NC at the final vertex. The events that trigger a cycle of IDL\_DRSGetNCChanges request over a given arc of the NC replica graph are specified in the next section.

The KCC is an automated management component of Active Directory that controls the [repsFrom](#) values on each DC and thereby controls the NC replica graph for each NC. One of the KCC's goals is to keep each NC replica graph of the forest in a good state, defined as follows:

1. Each DC in the NC replica graph contains a replica of the given NC.
2. If the DC at the initial vertex of an arc contains a partial replica of the given NC, so does the DC at the final vertex of that arc.
3. If  $d$  is any DC that contains a partial replica of the given NC, there is a path to  $d$  from some DC that contains a full replica of the given NC.
4. Define  $F$  as the set of all DCs that contain full replicas of the given NC. The sub-graph of the NC replica graph whose vertex set is  $F$  is strongly-connected.

The KCC performs this management by first creating *connection* objects (specified in section [7.1](#)), then creating *repsFrom* values from those connection objects (specified in section [7.2](#)). An administrator can create specially-marked connection objects (marking specified in section [7.2](#)) that the KCC will not modify, but will use in creating *repsFrom* values.

#### 3.1.1.1.14 Scheduled and Event-Driven Replication

If *client* and *server* are two DCs in the NC replica graph of a given NC and forest, where *server* is the initial vertex of an arc and *client* is the final vertex of the same arc, *client* will perform a replication cycle from *server* by calling IDL\_DRSGetNCChanges until the cycle is complete in either of these two cases:

1. The DC *client*'s *repsFrom* tuple for *server* contains a *schedule* field that calls for replication at the current time. The *schedule* contains a REPLTIMES structure specified in [MS-DRSR] section [5:REPLTIMES](#). This is *scheduled replication*.
2. The DC *server* calls the IDL\_DRSReplicaSync method ([MS-DRSR] section [4.1.23.2](#)) on the *client*. This is *event-driven replication*. The events that cause this form of replication are specified below.

A precondition for event-driven replication involves *server*'s *repsTo* abstract attribute, specified in [MS-DRSR] section [5:repsTo](#). The *repsTo* abstract attribute is a sequence tuples, like *repsFrom*. Like *repsFrom*, each *repsTo* tuple contains a field *uuidDsa* that contains the *objectGUID* of an *nTDSDSA* object. The *nTDSDSA* object represents a DC as specified in section [7.1](#). If *server*'s *repsTo* abstract attribute contains a tuple whose *uuidDsa* field contains the *objectGUID* of *client*'s *nTDSDSA* object, *server* performs event-driven replication to *client*.

It remains to specify how a DC's *repsTo* abstract attribute is populated, and to specify the events that trigger event-driven replication.

A DC's *repsTo* abstract attribute is populated as follows:

1. A DC *server*'s *repsTo* abstract attribute is populated for event-driven replication to *client* if *client*'s *repsFrom* tuple for *server* has the DRS\_ADD\_REF bit set in its *replicaFlags* field, and *client* calls the IDL\_DRSGetNCChanges method on *server* during scheduled replication. The DC *client* sets the DRS\_ADD\_REF bit in *Request.ulFlags* on the scheduled call to IDL\_DRSGetNCChanges on *server* ([MS-DRSR] section [4.1.10.4.1](#)) and *server* updates *repsTo* for event-driven replication to *client* as a result ([MS-DRSR] section [4.1.10.5.2](#)).

Since the KCC running on *client* writes *client*'s *repsFrom*, this behavior is controlled by the state of KCC objects as specified in section [7.2](#).

2. A DC *server*'s *repsTo* abstract attribute is populated for event-driven replication to DC *client* if the IDL\_DRSReplicaAdd method ([MS-DRSR] section [4.1.19.2](#)) is called on *client*, specifying *server* as the replication source (either *pmsgIn.V1.pszSourceDsaAddress* or *pmsgIn.V2.pszDsaSrc*, depending upon the request version used). If the IDL\_DRSReplicaAdd adds a new tuple to *client*'s *repsFrom*, it proceeds to call IDL\_DRSUpdateRefs ([MS-DRSR] section [4.1.26.2](#)) on *server* to update *server*'s *repsTo* abstract attribute.

Since IDL\_DRSReplicaAdd is an **RPC** method, this behavior is controlled by any authorized requestor of this method. Within Active Directory itself, IDL\_DRSReplicaAdd is called by the KCC to maintain *repsFrom*.

The events that trigger event-driven replication from a DC *server* are as follows:

1. The DC *server* receives an update, either originating or replicated, as specified in section [3.1.1.5.1.6](#) (Urgent replication).

2. A configurable time expires after DC server receives any update, as specified in section [3.1.1.5.1.5](#) (Replication notification).

#### 3.1.1.1.15 Replication Latency and Tombstone Lifetime

*Replication latency* is the delay between the time of an originating write to an NC and the time when this write is reflected in all replicas of that NC. Some writes are superseded before reaching all replicas, but for this simplified definition observe an attribute write that is not followed by other writes to that attribute for a long time.

Administrators of Active Directory control replication latency by setting several variables, specified in section [7.1](#) and section [7.2](#). These variables ultimately control the schedules used for scheduled replication, and control the use of event-driven replication. Replication latency is not fully predictable in a real system because it depends upon the volume of read requests to DCs, the volume of originating update requests to DCs, and the availability of DCs and communications links.

If the typical replication latency is larger than the *tombstone lifetime* (the value of the [tombstoneLifetime](#) attribute of the Directory Services object specified in section [7.1.1.2.4.1.1](#), interpreted as a number of days), some tombstones will be garbage collected before they have replicated to every NC replica. As a result some objects will never be deleted in some replicas. To restore consistency of object existence, an administrator cleans up such *lingering objects* with utility programs.

#### 3.1.1.2 Active Directory Schema

In Active Directory, the schema contains definitions for the objects that can be stored in the directory, and it enforces the rules that govern both the structure and the content of the directory. The schema consists of a set of classes, attributes, and syntaxes. A *class* is a category of objects that share a set of common characteristics. It is a formal description of a discrete, identifiable type of object that can be stored in the directory. Each object in the directory is an instance of one or more classes in the schema. Attributes define the types of information that an object can hold. For each class, the schema specifies the mandatory attributes and optional attributes that constitute the set of shared characteristics of the class. A syntax is the data type of a particular attribute. Syntaxes determine what data type an attribute can have. Active Directory uses a set of predefined syntaxes. The predefined syntaxes do not actually appear in the directory, and new syntaxes cannot be added.

The schema itself is represented in Active Directory by a set of objects known as *schema objects*. For each class in the schema, there is a schema object that defines the class. This object is a [classSchema](#) object. For each attribute in the schema, there is a schema object that defines the attribute. This object is an [attributeSchema](#) object. Therefore, every class is actually an instance of the [classSchema](#) class, and every attribute is an instance of the [attributeSchema](#) class. Administrators and applications can extend the schema by adding new attributes and classes and by modifying existing ones.

A schema object cannot be deleted, but it can be made *defunct* by setting the [isDefunct](#) attribute to true. A schema object that is not defunct is *active*. The primary effect of the defunct state is to prevent the schema object from being used in the creation or modification of new objects. For instance, attempts to perform an **LDAP** Add of an object with a defunct class fails, just as an attempt to perform an **LDAP** Add of a non-existent class fails. The full effects of the defunct state are specified later in this section.

##### 3.1.1.2.1 Schema NC

The schema NC contains all of the objects that define object classes and attributes used in a forest.

The root object of the schema NC, called the **schema container**, is an instance of class [dMD](#).

The contents of the schema NC is established when a forest is created. To enable a DC of a forest to be upgraded to a newer version of Windows Server, a *schema upgrade* process is first performed. This process updates the portion of the schema that Windows Server depends upon.

The attribute [objectVersion](#) on the schema container object stores the schema version of the forest. This attribute is set during the creation of the first **domain** in a forest, and is changed during schema upgrade after the schema is successfully upgraded to a newer version. To add a DC running a particular Windows Server version to an existing forest, the [objectVersion](#) of the forest's schema container must be greater than or equal to the value for that Windows Server version. The correspondence between Windows Server versions and values of the schema container [objectVersion](#) is:

- Windows 2000 Server: 13
- Windows Server 2003: 30
- Windows Server 2003 R2: 31
- Windows Server 2008 (AD/DS): 44
- Windows Server 2008 (AD/LDS): 30

Attribute [schemaInfo](#) on the schema container stores a String(octet) value of length 21 bytes. This attribute is updated on every original schema Add or Modify in the same transaction, and it is replicated to all the domain controllers in the forest upon completion of schema NC replication. The first byte of [schemaInfo](#) is 0xFF. The next four bytes are a 32-bit integer in big-endian byte order, used as the version of the update. The last 16 bytes is the [invocationId](#) of the DC where the schema change is made. The version starts from 1 for a new forest. Once a schema change is done, the version is incremented by one, and the [invocationId](#) of the DC where the schema change is done is written into the **GUID** part of the string. The [invocationId](#) attribute is specified in section [3.1.1.1.9](#).

For example, here is a value of [schemaInfo](#):

```
0xFF 0x00 0x00 0x07 0xC7 0x20 0x79 0x92 0xE6 0x84 0xB6 0xF6 0x40 0x99 0x47 0x21 0x8B 0xC9
0xE0 0xF1 0xF3
```

After a schema change is done on the schema master, the following is the new value:

```
0xFF 0x00 0x00 0x07 0xC8 0x20 0x79 0x92 0xE6 0x84 0xB6 0xF6 0x40 0x99 0x47 0x21 0x8B 0xC9
0xE0 0xF1 0xF3
```

There is child of the schema container with RDN cn=Aggregate and class [subSchema](#). This object has several **constructed attributes** that are compliant with [RFC2251](#) section 4.5.2, through which the client can retrieve the forest's current schema. See **constructed attributes** in section [3.1.1.4.5](#). This object cannot be modified.

### 3.1.1.2.2 Syntaxes

#### 3.1.1.2.2.1 Introduction

This section describes the **LDAP** syntaxes used in attributes in Active Directory DCs.

### 3.1.1.2.2.2 LDAP Representations

The **LDAP** syntaxes supported by DCs are as shown in the following table. The set of syntaxes supported is not extensible by schema modifications. Each syntax is identified by the combination of the [attributeSyntax](#), [oMSyntax](#) and, in select cases, [oObjectClass](#) attributes of an [attributeSchema](#) object. The cases for which [oObjectClass](#) is not used are indicated by the presence of a hyphen in the [oObjectClass](#) column in the table. The combinations shown in the table below are exhaustive; this table is consistent and identical for all versions of Windows Server.

While [oObjectClass](#) conceptually contains an **OID**, it is declared in the schema as String(Octet) syntax, requiring that values read from and written to it be expressed as the **BER** encoding of the OID (BER encoding is defined in [\[ITUX690\]](#)). In the table below, both the BER-encoded form and the dotted string form of the OID are given.

LDAP Syntax Name	<a href="#">attributeSyntax</a>	<a href="#">oMSyntax</a>	<a href="#">oObjectClass</a>
Boolean	2.5.5.8	1	-
Enumeration	2.5.5.9	10	-
Integer	2.5.5.9	2	-
LargeInteger	2.5.5.16	65	-
Object(Access-Point)	2.5.5.14	127	0x2B 0x0C 0x02 0x87 0x73 0x1C 0x00 0x85 0x3E (1.3.12.2.1011.28.0.702)
Object(DN-String)	2.5.5.14	127	0x2A 0x86 0x48 0x86 0xF7 0x14 0x01 0x010 0x01 0x0C (1.2.840.113556.1.1.1.12)
Object(OR-Name)	2.5.5.7	127	0x56 0x06 0x01 0x02 0x05 0x0B 0x1D (2.6.6.1.2.5.11.29)
Object(DN-Binary)	2.5.5.7	127	0x2A 0x86 0x48 0x86 0xF7 0x14 0x01 0x01 0x01 0x0B (1.2.840.113556.1.1.1.11)
Object(DS-DN)	2.5.5.1	127	0x2B 0x0C 0x02 0x87 0x73 0x1C 0x00 0x85 0x4A (1.3.12.2.1011.28.0.714)
Object(Presentation-Address)	2.5.5.13	127	0x2B 0x0C 0x02 0x87 0x73 0x1C 0x00 0x85 0x5C (1.3.12.2.1011.28.0.732)
Object(Replica-Link)	2.5.5.10	127	0x2A 0x86 0x48 0x86 0xF7 0x14 0x01 0x01 0x01 0x06 (1.2.840.113556.1.1.1.6)
String(Case)	2.5.5.3	27	-
String(IA5)	2.5.5.5	22	-
String(NT-Sec-Desc)	2.5.5.15	66	-
String(Numeric)	2.5.5.6	18	-
String(Object-Identifier)	2.5.5.2	6	-
String(Octet)	2.5.5.10	4	-

LDAP Syntax Name	<a href="#">attributeSyntax</a>	<a href="#">oMSyntax</a>	<a href="#">oMObjectClass</a>
String(Printable)	2.5.5.5	19	-
String(Sid)	2.5.5.17	4	-
String(Teletex)	2.5.5.4	20	-
String(Unicode)	2.5.5.12	64	-
String(UTC-Time)	2.5.5.11	23	-
String(Generalized-Time)	2.5.5.11	24	-

The representation for many of the above syntaxes is adopted from [\[RFC2252\]](#). The following table lists the syntaxes whose representation is adopted from that RFC, the [\[RFC2252\]](#) name of that syntax, and the associated section of [\[RFC2252\]](#) that specifies the representation.

LDAP Syntax Name	RFC 2252 Name	Section of RFC 2252
Boolean	Boolean	6.4
Enumeration	INTEGER	6.16
Integer	INTEGER	6.16*
LargeInteger	INTEGER	6.16*
Object(DS-DN)	DN	6.9 (see also <a href="#">[RFC2253]</a> )**
Object(Presentation-Address)	Presentation Address	6.28***
Object(Replica-Link)	Binary	6.2
String(IA5)	IA5 String	6.15†
String(Numeric)	Numeric String	6.23††
String(Object-Identifier)	OID	6.25
String(Octet)	Binary	6.2
String(Printable)	Printable String	6.29
String(Unicode)	Directory String	6.10
String(UTC-Time)	UTC Time	6.31†††
String(Generalized-Time)	Generalized Time	6.14†††

\* The Integer syntax in Active Directory is restricted to 32-bit integers. The LargeInteger syntax is restricted to 64-bit integers.

\*\* While Active Directory uses the [\[RFC2252\]](#) and [\[RFC2253\]](#) representation of DN's, it can also use alternative forms of the DN representation when it accepts requests and sends responses, if requested by the client. This is documented in LDAP\_SERVER\_EXTENDED\_DN\_OID section [3.1.1.3.4.1.5](#).

\*\*\* No validation is done by the DC to confirm that the value conforms to the representation specified in [\[RFC1278\]](#).

† Values restricted to ASN.1 IA5 strings (as specified in [\[ITUX680\]](#)).

†† Values restricted to ASN.1 Numeric strings (as specified in [\[ITUX680\]](#)).

††† Times are measured in granularity of one second.

The remaining syntaxes are represented as shown in the following sections.

#### 3.1.1.2.2.2.1 Object(DN-String)

A value with this syntax is a **UTF-8** string in the following format:

**S: byte\_count:string\_value:object\_DN**

where **byte\_count** is the number (in decimal) of bytes in the **string\_value** string, and **object\_DN** is a DN in Object(DS-DN) form. Since **string\_value** is a UTF-8 string, one character can require more than one byte to represent it.

#### 3.1.1.2.2.2.2 Object(Access-Point)

A value with this syntax is a UTF-8 string in the following format:

**presentation\_address#X500:object\_DN**

where **presentation\_address** is a value encoded in the Object(Presentation-Address) syntax and **object\_DN** is a DN in Object(DS-DN) form.

#### 3.1.1.2.2.2.3 Object(DN-Binary)

A value with this syntax is a UTF-8 string in the following format:

**B:char\_count:binary\_value:object\_DN**

where **char\_count** is the number (in decimal) of hexadecimal digits in **binary\_value**, **binary\_value** is the hexadecimal representation of a binary value, and **object\_DN** is a DN in Object(DS-DN) form. Each byte is represented by a pair of hexadecimal characters in **binary\_value**, with the first character of each pair corresponding to the most-significant nibble of the byte. The first pair in **binary\_value** corresponds to the first byte of the binary value, with subsequent pairs corresponding to the remaining bytes in sequential order. Note that **char\_count** is always even in a syntactically-valid Object(DN-Binary) value.

#### 3.1.1.2.2.2.4 Object(OR-Name)

A value with this syntax is a UTF-8 string in the following format:

**object\_DN**

where **object\_DN** is a DN in Object(DS-DN) form.

#### 3.1.1.2.2.2.5 String(Case)

A value with this syntax is a case-sensitive UTF-8 string, but the server does not enforce that a value of this syntax must be a valid UTF-8 string.

#### 3.1.1.2.2.2.6 String(NT-Sec-Desc)

A value with this syntax contains a **Windows security descriptor** in binary form. The binary form is that of a SECURITY\_DESCRIPTOR structure and is specified in [\[MS-DTYP\]](#) section **2.4.6**. It is otherwise encoded the same as the String(Octet) syntax.

#### 3.1.1.2.2.2.7 String(Sid)

A value with this syntax contains a SID in binary form. The binary form is that of a SID structure (the SID structure is specified in [\[MS-DTYP\]](#) section **2.4.2**. It is otherwise encoded the same as the String(Octet) syntax.

#### 3.1.1.2.2.2.8 String(Teletex)

A value with this syntax is a UTF-8 string restricted to characters with values between 0x20 and 0x7E, inclusive.

#### 3.1.1.2.2.3 Referential Integrity

Attributes with object reference syntaxes have special behavior, called *referential integrity*, as specified in section [3.1.1.1.6](#). The following are object reference syntaxes:

- Object(Access-Point)
- Object(DN-String)
- Object(OR-Name)
- Object(DS-Binary)
- Object(DS-DN)

For the four syntaxes other than Object(DS-DN), referential integrity only applies to the **object\_DN** portion of the value.

Active Directory imposes restrictions on which objects can be referenced by an attribute that has referential integrity. An attribute can reference any object in the same NC as the object on which that attribute is located. Additionally, attributes on an object in the **domain** NC, schema NC, or config NC can reference any object in any **domain** NC in the forest, any object in the schema NC or the config NC, or the root object of any application NC. For objects in application NCs, such attributes can reference any object in the config NC or the schema NC, or the root object of any application NC, in addition to any object in the same application NC as the object doing the referencing. All other references are disallowed by the server.

These restrictions are identical for AD/DS and for AD/LDS. Because AD/LDS does not support **domain** NCs, the only cross-NC references in an AD/LDS forest are from any NC to any object in the config and schema NCs or to the root of an application NC.

#### 3.1.1.2.2.4 Supported Comparison Operations

In addition to determining what can be stored in an attribute, the syntaxes determine what comparison operations the server permits on an attribute in an **LDAP** search filter, as well as how the server performs those comparisons. The following table maps each of the **LDAP** syntaxes to a comparison rule. All syntaxes of the same comparison rule support the same comparison operations and are compared using the same comparison rules.



<b>LDAP Syntax</b>	<b>Comparison Rule</b>
<i>Boolean</i>	<i>Bool</i>
<i>Enumeration</i>	<i>Integer</i>
<i>Integer</i>	<i>Integer</i>
<i>LargeInteger</i>	<i>Integer</i>
<i>Object(Access-Point)</i>	<i>DN-String</i>
<i>Object(DN-String)</i>	<i>DN-String</i>
<i>Object(OR-Name)</i>	<i>DN-Binary</i>
<i>Object(DN-Binary)</i>	<i>DN-Binary</i>
<i>Object(DS-DN)</i>	<i>DN</i>
<i>Object(Presentation-Address)</i>	<i>PresentationAddress</i>
<i>Object(Replica-Link)</i>	<i>Octet</i>
<i>String(Case)</i>	<i>CaseString</i>
<i>String(IA5)</i>	<i>CaseString</i>
<i>String(NT-Sec-Desc)</i>	<i>SecDesc</i>
<i>String(Numeric)</i>	<i>CaseString</i>
<i>String(Object-Identifier)</i>	<i>OID</i>
<i>String(Octet)</i>	<i>Octet</i>
<i>String(Printable)</i>	<i>CaseString</i>
<i>String(Sid)</i>	<i>Sid</i>
<i>String(Teletex)</i>	<i>NoCaseString</i>
<i>String(Unicode)</i>	<i>UnicodeString</i>
<i>String(UTC-Time)</i>	<i>Time</i>
<i>String(Generalized-Time)</i>	<i>Time</i>

The following table (split into three parts for readability) shows which of the choices in a **LDAP** Filter (that is, which comparison operations) are supported for each comparison rule. The LDAP Filter structure is defined in [\[RFC2251\]](#) section 4.5.1. Each comparison rule (for example, the rule for comparing two Bool values) is discussed following the table. The "and", "or", and "not" choices in an **LDAP** Filter are not included in this table because they are not comparisons performed against an attribute value. Active Directory treats approxMatch as equivalent to equalityMatch. For details on the three extensible matching rules, see section [3.1.1.3.4.4](#).

<b>Comparison Rule</b>	<b>present</b>	<b>equalityMatch</b>	<b>approxMatch</b>
<i>Bool</i>	X	X	X

<b>Comparison Rule</b>	<b>present</b>	<b>equalityMatch</b>	<b>approxMatch</b>
<i>Integer</i>	X	X	X
<i>DN-String</i>	X	X	X
<i>DN-Binary</i>	X	X	X
<i>DN</i>	X	X	X
<i>PresentationAddress</i>	X	X	X
<i>Octet</i>	X	X	X
<i>CaseString</i>	X	X	X
<i>SecDesc</i>	X		
<i>OID</i>	X	X	X
<i>Sid</i>	X	X	X
<i>NoCaseString</i>	X	X	X
<i>UnicodeString</i>	X	X	X
<i>Time</i>	X	X	X

<b>Comparison Rule</b>	<b>lessOrEqual</b>	<b>greaterOrEqual</b>	<b>substrings</b>
<i>Bool</i>	X	X	
<i>Integer</i>	X	X	
<i>DN-String</i>			
<i>DN-Binary</i>			
<i>DN</i>			
<i>PresentationAddress</i>			
<i>Octet</i>	X	X	X
<i>CaseString</i>	X	X	X
<i>SecDesc</i>			

<b>Comparison Rule</b>	<b>lessOrEqual</b>	<b>greaterOrEqual</b>	<b>substrings</b>
<i>OID</i>			
<i>Sid</i>	X	X	X
<i>NoCaseString</i>	X	X	X
<i>UnicodeString</i>	X	X	X
<i>Time</i>	X	X	

(Note: In the table shown below, the constant names in the headers for the extensibleMatch columns are prefixed with "LDAP\_MATCHING\_RULE\_". For example, "...BIT\_AND" is actually "LDAP\_MATCHING\_RULE\_BIT\_AND".)

<b>Comparison Rule</b>	<b>extensibleMatch: ...BIT_AND</b>	<b>extensibleMatch: ...BIT_OR</b>	<b>extensibleMatch: ...TRANSITIVE_EVAL</b>
<i>Bool</i>			
<i>Integer</i>	X	X	
<i>DN-String</i>			X*
<i>DN-Binary</i>			X*
<i>DN</i>			X*
<i>PresentationAddress</i>			
<i>Octet</i>			
<i>CaseString</i>			
<i>SecDesc</i>			
<i>OID</i>			
<i>Sid</i>			

<b>Comparison Rule</b>	<b><i>extensibleMatch:</i> ...BIT_AND</b>	<b><i>extensibleMatch:</i> ...BIT_OR</b>	<b><i>extensibleMatch:</i> ...TRANSITIVE_EVAL</b>
<i>NoCaseString</i>			
<i>UnicodeString</i>			
<i>Time</i>			

\* Supported only if the attribute is a **link attribute**. Evaluates to Undefined otherwise.

#### **3.1.1.2.2.4.1 Bool Comparison Rule**

A value of true is considered to be greater than a value of false.

#### **3.1.1.2.2.4.2 Integer Comparison Rule**

A signed comparison of integer values is performed.

#### **3.1.1.2.2.4.3 DN-String Comparison Rule**

Values of String(DN-String) or String(Access-Point) are equal if the **object\_DN** components name the same object and the **string\_value** or **presentation\_address** components are equal according to the UnicodeString comparison rule.

Evaluation of an LDAP\_MATCHING\_RULE\_TRANSITIVE\_EVAL matching rule is performed as documented in section [3.1.1.3.4.4](#). Only the **object\_DN** component is considered when evaluating a filter clause which uses this rule; **string\_value** or **presentation\_address** is ignored.

#### **3.1.1.2.2.4.4 DN-Binary Comparison Rule**

Values of String(DN-Binary) or String(OR-Name) are equal if the **object\_DN** components name the same object and the **binary\_value** or **OR\_address** components are identical in length and in content.

Evaluation of an LDAP\_MATCHING\_RULE\_TRANSITIVE\_EVAL matching rule is performed as documented in section [3.1.1.3.4.4](#). Only the **object\_DN** component is considered when evaluating a filter clause which uses this rule; **binary\_value** or **OR\_address** is ignored.

#### **3.1.1.2.2.4.5 DN Comparison Rule**

DN values are equal when they name the same object.

Evaluation of an LDAP\_MATCHING\_RULE\_TRANSITIVE\_EVAL matching rule is performed as documented in section [3.1.1.3.4.4](#).

#### **3.1.1.2.2.4.6 PresentationAddress Comparison Rule**

Two Object(Presentation-Address) values are equal when they have the same length and content.

#### 3.1.1.2.2.4.7 Octet Comparison Rule

Two values are equal when they are the same length and have identical contents. A value S1 is less than a value S2, where L is the smaller of the length of S1 and the length of S2, if either the first L bytes of S1 are less than the first L bytes of S2, or if the first L bytes of S1 and S2 are identical but the length of S1 is less than the length of S2. For the first L bytes of S1 to be less than the first L bytes of S2 means there exists an N (where  $N < L$ ) such that bytes 0...N-1 of S1 and S2 are identical, and byte N of S1 is less than byte N of S2.

For substring purposes, each byte in the value is treated as if it was a character. Values are compared using the ordinary rules for a SubstringFilter, as defined in [\[RFC2251\]](#) section 4.5.1. The "characters" are treated as if they were case-sensitive; that is, two characters are considered identical if and only if the bytes that represent them are identical.

#### 3.1.1.2.2.4.8 CaseString Comparison Rule

When compared using this comparison rule, two values are equal if they have identical length and contents. A value S1 is less than a value S2, where L is the smaller of the length of S1 and the length of S2, if either the first L bytes of S1 are less than the first L bytes of S2, or if the first L bytes of S1 and S2 are identical but the length of S1 is less than the length of S2. For the first L bytes of S1 to be less than the first L bytes of S2 means there exists an N (where  $N < L$ ) such that bytes 0...N-1 of S1 and S2 are identical, and byte N of S1 is less than byte N of S2.

For substring purposes, this comparison rule treats values as if they were case-sensitive strings of characters, and obey the ordinary rules for a SubstringFilter, as defined in [\[RFC2251\]](#) section 4.5.1. In this comparison, two characters are considered identical if and only if the bytes that represent them are identical.

#### 3.1.1.2.2.4.9 SecDesc Comparison Rule

SecDescs are compared as octet strings as in section [3.1.1.2.2.4.7](#) above.

#### 3.1.1.2.2.4.10 OID Comparison Rule

Two String(Object-Identifier) values are equal when they are the same OID.

#### 3.1.1.2.2.4.11 Sid Comparison Rule

String(SID) values are treated as the binary representation of the SID (see [\[MS-DTYP\]](#) section 2.4.2). The binary representations of the SID are compared using the Octet comparison rule.

#### 3.1.1.2.2.4.12 NoCaseString Comparison Rule

This comparison rule is identical to the CaseString comparison rule, except that for each comparison, characters are treated in a case-insensitive fashion. For equality, ordering (greater-than-or-equals and less-than-or-equals), and substrings, two characters are identical if the bytes which represent them are identical or if the characters differ from each other only by their case. The "C" locale, as defined in [ISO-9899], is used for determining whether two characters differ by case.

#### 3.1.1.2.2.4.13 UnicodeString Comparison Rule

Comparison of values using this comparison rule is performed via the **Unicode** comparison function specified in section [7.5](#). If an LDAP\_SERVER\_SORT\_OID extended control (see section [3.1.1.3.4](#)) is attached to the search request and specifies a locale in its orderingRule field, the locale specified is used for the **Unicode** comparison. Otherwise, the **Unicode** comparison is performed using United

States English (LCID 0409). The comparison function is independent of the server locale and therefore gives the same result on all DCs. The comparison function operates on **Unicode** strings containing characters from all alphabets and does not, for instance, involve reducing the string to the alphabet used by United States English before performing the comparison. This comparison function is used to determine both equality and ordering (greater-than-or-equals and less-than-or-equals), as well as to determine equality of substrings when performing a substring comparison.

This comparison rule is used in processing search filters, *not* in sorting search results. See section [3.1.1.3.4.1.13](#) for per-locale sorting of search results.

### 3.1.1.2.2.4.14 Time Comparison Rule

Time T1 is greater than time T2 if T1 denotes a time subsequent to T2.

### 3.1.1.2.3 Attributes

The attributes of class [attributeSchema](#) are specified in the following table.

The term "Unique" (in quotation marks) in the table below, and in the similar table for [classSchema](#) in section [3.1.1.2.4.8](#), means that the value satisfies the following constraint:

- If the forest functional level is less than DS\_BEHAVIOR\_WIN2003, the value is unique among all values of this attribute in the set containing every [attributeSchema](#) and [classSchema](#) object in the schema NC.
- If the forest functional level is DS\_BEHAVIOR\_WIN2003 or greater, the value is unique among all values of this attribute in the set containing every [attributeSchema](#) and [classSchema](#) object S in the schema NC that satisfies at least one of the following three conditions:
  - S![isDefunct](#) ≠ true, that is, S is active.
  - FLAG\_ATTR\_IS\_RDN is present in S![systemFlags](#) (defined in the table below).
  - S = C![rDNAttID](#) (section [3.1.1.2.4.8](#)) for some [classSchema](#) object C.

The term System-only in the table below means that the attribute is defined with [systemOnly](#) true. The value of the system-only attributes below can be specified on Add (except where noted) but cannot be modified on existing objects by **LDAP** Modify requests (except as specified in section [3.1.1.5.3.2](#)), only by the system. The table is ordered with the system-only attributes before the other attributes.

Attribute	Description
<a href="#">objectClass</a>	Equals the sequence [ <a href="#">top</a> , <a href="#">attributeSchema</a> ]. System-only.
<a href="#">attributeID</a>	"Unique" OID that identifies this attribute. System-only.
<a href="#">schemaIDGUID</a>	"Unique" <b>GUID</b> that identifies this attribute, used in <b>security descriptors</b> . If not specified on Add, the DC generates a fresh <b>GUID</b> . System-only.
<a href="#">msDS-IntId</a>	Not specified on Add (if specified in the Add request, the DC returns <b>LDAP</b> error <i>unwillingToPerform</i> ); the value (a 32-bit unsigned integer in the subrange [0x80000000..0xBFFFFFFF]) is generated by the DC. Present on <a href="#">attributeSchema</a> objects added when forest functional level is DS_BEHAVIOR_WIN2003 or greater with FLAG_SCHEMA_BASE_OBJECT not present in <a href="#">systemFlags</a> (below). The value of <a href="#">msDS-IntId</a> is the

Attribute	Description
	ATTRTYP of this <a href="#">attributeSchema</a> object. Unique among all values of this attribute on objects in the schema NC, regardless of forest functional level. System-only.
<a href="#">linkID</a>	Optional. If present, and not zero, this is a link attribute, and the <a href="#">linkID</a> value is unique among all values of this attribute on objects in the schema NC, regardless of forest functional level. If <a href="#">linkID</a> is even, the attribute is a forward link attribute; otherwise it is a back link attribute. The <a href="#">linkID</a> for back link attribute equals to the <a href="#">linkID</a> of the corresponding forward link attribute plus one. Special auto-generation behavior for the <a href="#">linkID</a> attribute is specified in section <a href="#">3.1.1.2.3.1</a> . System-only.
<a href="#">mAPIID</a>	Optional. "Unique" integer that identifies this attribute, used by <b>Messaging Application Programming Interface (MAPI)</b> clients. Not present on <a href="#">attributeSchema</a> objects in AD/LDS. Special auto-generation behavior for the mAPIID attribute is specified in section <a href="#">3.1.1.2.3.2</a> . System-only. If the DC functional level is DS_BEHAVIOR_WIN2008 or greater, the <a href="#">mAPIID</a> attribute can be modified on <a href="#">attributeSchema</a> objects that do not include FLAG_SCHEMA_BASE_OBJECT as the <a href="#">systemFlags</a> attribute. Otherwise, the <a href="#">mAPIID</a> attribute cannot be modified.
<a href="#">attributeSyntax</a>	One of the three attributes that identify the syntax of the attribute. See section <a href="#">3.1.1.2.2</a> . System-only.
<a href="#">oMSyntax</a>	One of the three attributes that identify the syntax of the attribute. See section <a href="#">3.1.1.2.2</a> . System-only.
<a href="#">oObjectClass</a>	Optional. One of the three attributes that identify the syntax of the attribute. See section <a href="#">3.1.1.2.2</a> . System-only.
<a href="#">isSingleValued</a>	True if this attribute is single-valued; false, if it is multi-valued. If an attribute is multi-valued, all values have the syntax specified for the attribute. System-only.
<a href="#">extendedCharsAllowed</a>	Optional. If true, character set constraint is not enforced on values of this attribute. Applies to attributes of syntax String(IA5), String(Numeric), String(Teletex), String(Printable). System-only.
<a href="#">systemFlags</a>	Optional. Flags that determine specific system operations; see section <a href="#">2.2.10</a> for values. The <a href="#">systemFlags</a> values specific to an <a href="#">attributeSchema</a> object are: FLAG_ATTR_NOT_REPLICATED: this attribute is non-replicated; FLAG_ATTR_REQ_PARTIAL_SET_MEMBER: This attribute is a member of PAS regardless the value of attribute <a href="#">isMemberOfPartialAttributeSet</a> ; FLAG_ATTR_IS_CONSTRUCTED: this attribute is a <b>constructed attribute</b> ; FLAG_ATTR_IS_OPERATIONAL: this attribute is an operational attribute, as defined in <a href="#">[RFC2251]</a> section 3.2.1; FLAG_SCHEMA_BASE_OBJECT: this class is part of the base schema. Modifications to a base schema object are restricted as described in section <a href="#">3.1.1.2.5</a> . FLAG_ATTR_IS_RDN: a hint to the DC that this attribute can be used an RDN attribute of a class.

Attribute	Description
	System-only.
<a href="#">systemOnly</a>	Optional. The value of a system-only attribute cannot be modified on existing objects by <b>LDAP</b> Modify requests (except as specified in section <a href="#">3.1.1.5.3.2</a> ), only by the system. System-only.
<a href="#">cn</a>	RDN for the schema object.
<a href="#">LDAPDisplayName</a>	"Unique" name that identifies this attribute, used by <b>LDAP</b> clients. If not specified on Add, the DC generates a value as specified in section <a href="#">3.1.1.2.3.4</a> . The syntax of <a href="#">LDAPDisplayName</a> is described in <a href="#">[RFC2251]</a> section 4.1.4.
<a href="#">attributeSecurityGUID</a>	Optional. <b>GUID</b> by which the security system identifies the <b>property set</b> of this attribute. See the specification of property sets in section <a href="#">3.1.1.2.3.3</a> .
<a href="#">rangeLower</a>	Optional. Lower range of values that are allowed for this attribute. For syntax Integer, LargeInteger, Enumeration, String(UTC-time), String(Generalized-time), <a href="#">rangeLower</a> equals the minimum allowed value; For syntax Object(DN-binary), Object(DN-String), <a href="#">rangeLower</a> equals the minimum length of the binary_value or string_value portion of the given value. For String(Unicode), <a href="#">rangeLower</a> is the minimum length in characters. <a href="#">rangeLower</a> is not used on syntax Boolean and Object(DS-DN). For all other syntaxes, <a href="#">rangeLower</a> equals the minimum length in bytes. Note that <a href="#">rangeLower</a> is a 32-bit integer and cannot express the full range of LargeInteger, String(UTC-time), and String(Generalized-time).
<a href="#">rangeUpper</a>	Optional. Upper range of values that are allowed for this attribute. For syntax Integer, LargeInteger, Enumeration, String(UTC-time), String(Generalized-time), <a href="#">rangeUpper</a> equals the maximum allowed value; For syntax Object(DN-binary), Object(DN-String), <a href="#">rangeUpper</a> equals the maximum length of the binary_value or string_value portion of the given value. For String(Unicode), <a href="#">rangeUpper</a> is the maximum length in character. <a href="#">rangeUpper</a> is not used on syntax Boolean and Object(DS-DN). For all other syntaxes, <a href="#">rangeUpper</a> equals the maximum length in bytes. Note that <a href="#">rangeUpper</a> is a 32-bit integer and cannot express the full range of LargeInteger, String(UTC-time), and String(Generalized-time).
<a href="#">searchFlags</a>	Optional. The <a href="#">searchFlags</a> attribute specifies whether an attribute is indexed, among other things; see section <a href="#">2.2.9</a> for values. It contains bitwise flags as follows: fATTINDEX: * fPDNTATTINDEX: * fANR: Add this attribute to the Ambiguous Name Resolution (ANR) set. If this flag is set, then fATTINDEX must also be set. See <a href="#">3.1.1.3.1.3.3</a> for ANR search. fPRESERVEONDELETE: Preserve this attribute on logical deletion. This flag is ignored on link attributes. fCOPY: Interpreted by <b>LDAP</b> clients, not by server. This attribute is copied on object copy. fTUPLEINDEX: * fSUBTREEATTINDEX: *



Attribute	Description
	<p>fCONFIDENTIAL: This attribute is confidential, special <b>access check</b> is needed; see section Reads:Access Checks in section <a href="#">3.1.1.4</a>.</p> <p>fNEVERVALUEAUDIT: Auditing of changes to values contained in this attribute should not be performed. Auditing is outside the state model.</p> <p>fRODCFilteredAttribute: This attribute is part of the filtered attribute set. This flag is only effective on a DC whose DC functionality level is DS_BEHAVIOR_WIN2008 or greater. See section <a href="#">3.1.1.2.3.5</a> for additional restrictions.</p> <p>The effects of <a href="#">searchFlags</a> marked * are outside the state model. They direct the server to construct certain indexes that affect system performance. These flags may be ignored by other implementations, but must not be used in a conflicting way that would affect the performance of Windows DCs.</p>
<a href="#">schemaFlagsEx</a>	<p>Optional. The <a href="#">schemaFlagsEx</a> attribute specifies whether an attribute can be part of the filtered attribute set; see section <a href="#">2.2.11</a> for values. It contains bitwise flags as follows:</p> <p>FLAG_ATTR_IS_CRITICAL: If this flag is set, and the fRODCFilteredAttribute flag in <a href="#">searchFlags</a> is also set, the fRODCFilteredAttribute flag is ignored. If fRODCFilteredAttribute is not set, setting this flag has no effect. This flag is effective only on a DC whose DC functionality level is DS_BEHAVIOR_WIN2008 or greater; it is ignored by a DC that is not at that level or greater.</p>
<a href="#">isMemberOfPartialAttributeSet</a>	<p>Optional. If true, the attribute is a member of the forest's partial attribute set.</p> <p>An attribute is a member of the forest's partial attribute set if and only if either (1) this attribute is true or (2) the FLAG_ATTR_REQ_PARTIAL_SET_MEMBER bit is set in the <a href="#">systemFlags</a> attribute.</p> <p>If this attribute is true and the FLAG_ATTR_NOT_REPLICATED bit is set in the <a href="#">systemFlags</a> attribute, the attribute is accessible through the global catalog servers but the value of the attribute does not replicate.</p>

### 3.1.1.2.3.1 Auto-Generated linkID

If the DC functional level is DS\_BEHAVIOR\_WIN2003 or greater, and an [attributeSchema](#) object is created with **LDAP** Add, and the Add request assigns the OID 1.2.840.113556.1.2.50 as the value of the [linkID](#) attribute, the DC sets the [linkID](#) attribute to an even integer that does not already appear as the [linkID](#) on a schema object. The attribute created by the Add is a forward link attribute.

If the DC functional level is DS\_BEHAVIOR\_WIN2003 or greater, and an [attributeSchema](#) object is created with **LDAP** Add, and the Add request assigns either the [attributeID](#) or the [LDAPDisplayName](#) of an existing forward link attribute as the value of the [linkID](#) attribute, the DC sets the [linkID](#) attribute to the [linkID](#) of the given forward link attribute plus one. The attribute created by the Add is a back link attribute corresponding to the given forward link attribute.

The distinguished values that trigger auto-generation behavior for the [linkID](#) attribute do not conform to the declared syntax of the [linkID](#) attribute, and the DC accepts these values without the **LDAP** error that would normally occur in such a case.

### 3.1.1.2.3.2 Auto-Generated mAPIID

If the DC functional level is DS\_BEHAVIOR\_WIN2008 or greater, and an [attributeSchema](#) object is created with **LDAP** Add, and the Add request assigns the OID 1.2.840.113556.1.2.49 as the value of the [mAPIID](#) attribute, the DC sets the [mAPIID](#) attribute to an integer that does not already appear as the [mAPIID](#) on a schema object.

The distinguished values that trigger auto-generation behavior for the [mAPIID](#) attribute do not conform to the declared syntax of the [mAPIID](#) attribute, and the DC accepts these values without the **LDAP** error that would normally occur in such a case.

### 3.1.1.2.3.3 Property Set

An attribute whose [attributeSchema](#) object has a value for the [attributeSecurityGUID](#) attribute belongs to a property set; the property set is identified by the property set **GUID**, which is the [attributeSecurityGUID](#) value.

A property set **GUID** can be used instead of the [schemaIDGUID](#) of an attribute when defining a security descriptor, as specified in section [5.1.3.2](#).

The following table lists the property sets present in the default AD/DS schema.

Name	Property set GUID
Domain Password & Lockout Policies	C7407360-20BF-11D0-A768-00AA006E0529
General Information	59BA2F42-79A2-11D0-9020-00C04FC2D3CF
Account Restrictions	4C164200-20C0-11D0-A768-00AA006E0529
Logon Information	5F202010-79A5-11D0-9020-00C04FC2D4CF
Group Membership	BC0AC240-79A9-11D0-9020-00C04FC2D4CF
Phone and Mail Options	E45795B2-9455-11D1-AEBD-0000F80367C1
Personal Information	77B5B886-944A-11D1-AEBD-0000F80367C1
Web Information	E45795B3-9455-11D1-AEBD-0000F80367C1
Public Information	E48D0154-BCF8-11D1-8702-00C04FB96050
Remote Access Information	037088F8-0AE1-11D2-B422-00A0C968F939
Other Domain Parameters (for use by SAM)	B8119FD0-04F6-4762-AB7A-4986C76B3F9A
DNS Host Name Attributes	72E39547-7B18-11D1-ADEF-00C04FD8D5CD
MS-TS-GatewayAccess (*)	FFA6F046-CA4B-4FEB-B40D-04DFEE722543
Private Information (*)	91E647DE-D96F-4B70-9557-D63FF4F3CCD8
Terminal Server License Server (*)	5805BC62-BDC9-4428-A5E2-856A0F4C185E

(\*) The last three property sets are present only in Windows Server 2008 AD/DS forests.

New property sets can be created by adding [controlAccessRight](#) objects to the Extended-Rights container as described in section [5.1.3.2.1](#). The [rightsGuid](#) attribute of the [controlAccessRight](#) object is the property set **GUID**.

AD/LDS installs a reduced schema by default. The default AD/LDS schema only includes the following property sets:

- General Information
- Account Restrictions
- Logon Information
- Group Membership
- Phone and Mail Options
- Personal Information
- Web Information
- Public Information

#### 3.1.1.2.3.4 IdapDisplayName Generation

When [IDAPDisplayName](#) is not given explicitly when creating an attribute or class, the system will generate a default one from the value of [cn](#) with the following routine:

```
String generateLdapDisplayName(IN cn: String)
{
    Identify the substrings in cn that are delimited by
    one or more characters in the set {' ', '-', '_'},
    let S be a string array containing all the substrings;
    Let T be a string array with the same number of elements
    as S, such that
    1. First string in T (T[1]) is exactly the same string
    as S[1], except the first character of T[1] is the
    lower case form of the first character of S[1];
    2. For the remaining strings, T[i] is the same as S[i],
    except the first character of T[i] is the upper case
    of the first character of S[i];
    Let string st be the concatenation of the strings in T;
    Return st;
}
```

Example, if the [cn](#) of a new class is Sam-Domain, the default [IDAPDisplayName](#) is [samDomain](#).

#### 3.1.1.2.3.5 Flag fRODCFilteredAttribute in Attribute SearchFlags

An attribute cannot be a member of **filtered attribute set** if one of following the conditions is true:

- FLAG\_ATTR\_NOT\_REPLICATED bit is set in attribute [systemFlags](#) of the [attributeSchema](#) object;
- FLAG\_ATTR\_REQ\_PARTIAL\_SET\_MEMBER bit is set in attribute [systemFlags](#) of the [attributeSchema](#) object;

- FLAG\_ATTR\_IS\_CONSTRUCTED bit is set in attribute [systemFlags](#) of the [attributeSchema](#) object;
- FLAG\_ATTR\_IS\_CRITICAL bit is set in attribute [schemaFlagsEx](#) of the [attributeSchema](#) object;
- Attribute [systemOnly](#) of the [attributeSchema](#) object is true;
- The attribute is in the below list: [currentValue](#), [dBCSPwd](#), [unicodePwd](#), [ntPwdHistory](#), [priorValue](#), [supplementalCredentials](#), [trustAuthIncoming](#), [trustAuthOutgoing](#), [lmPwdHistory](#), [initialAuthIncoming](#), [initialAuthOutgoing](#), [msDS-ExecuteScriptPassword](#), [displayName](#), [codePage](#), [creationTime](#), [lockoutDuration](#), [lockOutObservationWindow](#), [logonHours](#), [lockoutThreshold](#), [maxPwdAge](#), [minPwdAge](#), [minPwdLength](#), [nETBIOName](#), [pwdProperties](#), [pwdHistoryLength](#), [pwdLastSet](#), [securityIdentifier](#), [trustDirection](#), [trustPartner](#), [trustPosixOffset](#), [trustType](#), [rid](#), [domainReplica](#), [accountExpires](#), [nTMixedDomain](#), [operatingSystem](#), [operatingSystemVersion](#), [operatingSystemServicePack](#), [fSMORoleOwner](#), [trustAttributes](#), [trustParent](#), [flatName](#), [sIDHistory](#), [dNSHostName](#), [lockoutTime](#), [servicePrincipalName](#), [isCriticalSystemObject](#), [msDS-TrustForestTrustInfo](#), [msDS-SPNSuffixes](#), [msDS-AdditionalDnsHostName](#), [msDS-AdditionalSamAccountName](#), [msDS-AllowedToDelegateTo](#), [msDS-KrbTgtLink](#), [msDS-AuthenticatedAtDC](#), [msDS-SupportedEncryptionTypes](#).

If one of the conditions is true, the attribute will not be in the **filtered attribute set** even if the flag `FRDODCFilteredAttribute` is set in attribute [searchFlags](#) of the [attributeSchema](#) object.

### 3.1.1.2.4 Classes

#### 3.1.1.2.4.1 Class Categories

There are four categories of classes:

**Structural Classes:** Structural classes are the classes that can have instances in the directory.

**Abstract Classes:** Abstract classes are templates that are used to derive new classes. Abstract classes cannot be instantiated in the directory.

**Auxiliary Classes:** Auxiliary classes contain a list of attributes. Adding the auxiliary class to the definition of a structural or abstract class adds the auxiliary class's attributes to the definition. An auxiliary class cannot be instantiated by itself in the directory.

**88 Classes:** 88 Classes do not fall into any of the above categories. An 88 class can be used as an abstract class, a structural class, or an auxiliary class.

Structural class, abstract class, and auxiliary class are defined in section 8.3 of [\[X501\]](#). 88 class is AD-specific, adapted from 1988 X.500 standard, see [\[ITU-X501-1988\]](#). 88 class is included for compatibility with this older standard, and is not intended to be used in new schema extensions.

#### 3.1.1.2.4.2 Inheritance

**Inheritance** is the ability to build new classes from existing classes. The new class is defined as a subclass of another class, called its *superclass*. A subclass inherits from its superclass the mandatory and optional attributes and its structural parent classes in the directory hierarchy. All classes are subclasses, directly or indirectly, of a single **abstract object class**, called [top](#). In Active Directory, a class has exactly one superclass; [top](#) is its own superclass. An ordered set of superclasses of a class, ending with class [top](#), is its *superclass chain* ([\[X501\]](#)). The superclass chain of a class does not include the class itself, except that the superclass chain of [top](#) is the single-element sequence [ [top](#) ].

Abstract classes can inherit only from abstract classes, auxiliary classes can inherit from all classes except structural classes, and structural classes can inherit from all classes except auxiliary classes. 88 classes can inherit from all classes.

#### 3.1.1.2.4.3 objectClass

Attribute [objectClass](#) is a multi-valued attribute that appears on all the objects in the directory. When instantiating a structural class, the [objectClass](#) attribute of the new object contains a sequence of class names. The first element is always class [top](#). The last element is the name of the structural class that was instantiated (referred to as the most specific class). The rest of the classes in the superclass chain are listed in between in the order of inheritance from class [top](#). For example, a [user](#) object has the following four-element sequence as the value of [objectClass](#):

[ [top](#), [person](#), [organizationalPerson](#), [user](#) ]

For information on instantiating auxiliary classes see section [3.1.1.2.4.6](#) below.

#### 3.1.1.2.4.4 Structure Rules

Structure rules define the possible tree structures. In Active Directory, the structure rules are completely expressed by the [possSuperiors](#) and [systemPossSuperiors](#) attributes that are present on each [classSchema](#) object. The union of values in these two attributes specifies the classes that are allowed to be parents of an object instance of the class in question.

#### 3.1.1.2.4.5 Content Rules

Content rules determine the mandatory and optional attributes of the class instances that are stored in the directory. In Active Directory, the content rules are completely expressed by the [mustContain](#), [mayContain](#), [systemMustContain](#), and [systemMayContain](#) attributes of the schema definitions for each class. The union of values in the [mustContain](#) and [systemMustContain](#) attributes specifies the attributes that are required to be present on an object instance of the class in question. The union of values in the [mustContain](#), [systemMustContain](#), [mayContain](#) and [systemMayContain](#) attributes specifies the attributes that are allowed to be present on an object instance of the class in question.

#### 3.1.1.2.4.6 Auxiliary Class

In the initial release of Windows 2000, Active Directory only provided support for statically linking auxiliary classes to the [classSchema](#) definition of another object class. When an auxiliary class *aux* is statically linked to some other class *c*, it is as if all of the mandatory and optional attributes of the auxiliary class *aux* are added to the class *c*. If *c* is a structural object class, then the [objectClass](#) attribute of objects of class *c* does not contain the name of the auxiliary class *aux*.

Beginning with Windows Server 2003 operating systems, Active Directory provides support for dynamically instantiating auxiliary classes on objects using **LDAP** Add and Modify. In this case, the dynamically-attached auxiliary class only affects the individual object to which it is attached, as opposed to a statically-attached auxiliary class, which is attached to a class and effects every object of that class.

In attribute [objectClass](#) of an object with dynamic auxiliary classes, class [top](#) remains as the first value, followed by the set of dynamic auxiliary classes and the classes in their superclass chains, excluding those already present in the superclass chain of the most specific structural class. The classes in the superclass chain of the most specific structural class are listed after that, in the order of inheritance from [top](#). The most specific structural class remains last in the sequence.

For example, a [user](#) object with auxiliary class [mailRecipient](#) dynamically added has the following five-element sequence as the value of [objectClass](#)

[ top, mailRecipient, person, organizationalPerson, user ]

Dynamic Auxiliary Classes are supported when the forest functional level is DS\_BEHAVIOR\_WIN2003 or greater.

#### 3.1.1.2.4.7 RDN Attribute of a Class

Each class designates an RDN attribute. The RDN attribute's name and value provide the RDN for the class, for example "ou=ntdev", "cn=Peter Houston". If not specified in a class by attribute [rDNAttID](#), the RDN attribute is inherited from the super class of the class. The RDN attribute is of syntax String(Unicode).

#### 3.1.1.2.4.8 Class classSchema

The attributes of class [classSchema](#) are specified in the following table.

The term "Unique" (in quotation marks) in the table below is defined in section [3.1.1.2.3](#).

The term system-only in the table below is defined in section [3.1.1.2.3](#).

Attribute	Description
<a href="#">objectClass</a>	Equals the sequence [ <a href="#">top</a> , <a href="#">classSchema</a> ]. System-only.
<a href="#">governsID</a>	"Unique" OID that identifies this class. System-only.
<a href="#">schemaIDGUID</a>	"Unique" <b>GUID</b> that identifies this class, used in security descriptors. If not specified on Add, the DC generates a fresh <b>GUID</b> . System-only.
<a href="#">msDS-IntId</a>	Not specified on Add (if specified in the Add request, the DC returns <b>LDAP</b> error <i>unwillingToPerform</i> ); the value (a 32-bit unsigned integer in the subrange [0x80000000..0xBFFFFFFF]) is generated by the DC. Present on <a href="#">classSchema</a> objects added when forest functional level is DS_BEHAVIOR_WIN2003 or greater with FLAG_SCHEMA_BASE_OBJECT not present in <a href="#">systemFlags</a> (below). The value of <a href="#">msDS-IntId</a> is the ATTRTYP of this <a href="#">classSchema</a> object. Unique among all values of this attribute on objects in the schema NC, regardless of forest functional level. System-only.
<a href="#">rDNAttID</a>	Optional. <a href="#">attributeID</a> of the RDN attribute. If the <a href="#">rDNAttID</a> is not present, the RDN attribute is inherited from the superclass of this class. System-only.
<a href="#">subClassOf</a>	<a href="#">governsID</a> of the superclass of this class. System-only.
<a href="#">systemMustContain</a>	Optional. <a href="#">attributeIDs</a> of the mandatory attributes of this class that are required for system operation. System-only.
<a href="#">systemMayContain</a>	Optional. <a href="#">attributeIDs</a> of the optional attributes of this class that are required for system operation. System-only.
<a href="#">systemPossSuperiors</a>	Optional. <a href="#">governsIDs</a> of the classes that can be parents of this class within an NC tree, where these parent-child relationships are required for system operation. System-only.
<a href="#">systemAuxiliaryClass</a>	Optional. <a href="#">governsIDs</a> of the auxiliary classes that are statically linked to this class, where these auxiliary classes contain attributes required for system

Attribute	Description
	operation. System-only.
<a href="#">objectClassCategory</a>	Class category (section <a href="#">3.1.1.2.4.1</a> ), encoded as follows: 0: 88 Class 1: Structural class 2: Abstract class 3: Auxiliary class System-only.
<a href="#">systemFlags</a>	Optional. Flags that determine specific system operations; see section <a href="#">2.2.10</a> for values. The single <a href="#">systemFlags</a> value specific to a <a href="#">classSchema</a> object is: FLAG_SCHEMA_BASE_OBJECT: this class is part of the base schema. Modifications to a base schema object are restricted as described in section <a href="#">3.1.1.2.5</a> . System-only.
<a href="#">systemOnly</a>	Optional. Only a DC can create (section <a href="#">3.1.1.5.2.2</a> ) and modify (section <a href="#">3.1.1.5.3.2</a> ) instances of a system-only class. System-only.
<a href="#">cn</a>	RDN for the schema object.
<a href="#">LDAPDisplayName</a>	"Unique" name that identifies this class, used by <b>LDAP</b> clients. If not specified on Add, the DC generates a value as specified in section <a href="#">3.1.1.2.3.4</a> . The syntax of <a href="#">LDAPDisplayName</a> is described in <a href="#">[RFC2251]</a> section 4.1.4.
<a href="#">mustContain</a>	Optional. <a href="#">attributeIDs</a> of the mandatory attributes of this class in addition to the <a href="#">systemMustContain</a> attributes.
<a href="#">mayContain</a>	Optional. <a href="#">attributeIDs</a> of the optional attributes of this class in addition to the <a href="#">systemMayContain</a> attributes.
<a href="#">possSuperiors</a>	Optional. <a href="#">governsIDs</a> of the classes that can be parents of this class within an NC tree, in addition to the <a href="#">systemPossSuperiors</a> classes.
<a href="#">auxiliaryClass</a>	Optional. <a href="#">governsIDs</a> of the auxiliary classes that are statically linked to this class, in addition to the <a href="#">systemAuxiliaryClass</a> classes.
<a href="#">defaultSecurityDescriptor</a>	Optional. The default security descriptor (in SDDL format, <a href="#">[MS-DTYP]</a> section 2.5.1) that is assigned to new instances of this class if no security descriptor is specified during creation of the class or is merged into a security descriptor if one is specified. The rules for security descriptor merging are specified in <a href="#">[MS-DTYP]</a> section 2.5.2.2.
<a href="#">defaultObjectCategory</a>	A reference to some <a href="#">classSchema</a> object. This value is the default value of the <a href="#">objectCategory</a> attribute of new instances of this class if none is specified during <b>LDAP</b> Add.
<a href="#">defaultHidingValue</a>	Optional. If <a href="#">defaultHidingValue</a> is true on a <a href="#">classSchema</a> object, then when an Add creates an instance of this class (that is, where this class is the most specific class), and the Add does not specify a value for the <a href="#">showInAdvancedViewOnly</a> attribute, it is as if the Add had specified true for the <a href="#">showInAdvancedViewOnly</a> attribute.  The <a href="#">showInAdvancedViewOnly</a> attribute is interpreted by <b>LDAP</b> clients, not by the DC. If true, certain user interfaces do not display the object.

### 3.1.1.2.5 Schema Modifications

This section documents the special behavior of schema objects with respect to **LDAP** Add, Modify, Modify DN, and Delete requests.

Only the DC that owns the schema master FSMO role performs originating updates of objects in the schema NC, as specified in section [3.1.1.1.11](#).

All transactions that perform originating updates to objects in the schema NC are serialized, even if the updates do not appear to conflict and thus do not seem to require serialization.

Many attributes of [attributeSchema](#) and [classSchema](#) objects are system-only, as specified in sections [3.1.1.2.3](#) and [3.1.1.2.4](#). An **LDAP** Modify request that attempts to modify a system-only attribute (except as specified in section [3.1.1.5.3.2](#)) fails with **LDAP** error *constraintViolation*.

A Delete of an [attributeSchema](#) or [classSchema](#) object fails, with **LDAP** error *unwillingToPerform*.

#### 3.1.1.2.5.1 Consistency and Safety Checks

This section documents schema object special behaviors that are not closely tied to the defunct state. These special behaviors are divided into two classes:

- Consistency checks
- Safety checks

Consistency checks maintain the consistency of the schema. Safety checks reduce the possibility of a schema update by one application breaking another application.

If an Add or Modify request fails either a consistency or a safety check, the response is **LDAP** error *unwillingToPerform*.

##### 3.1.1.2.5.1.1 Consistency Checks

The term "Unique" (in quotation marks) in the statements below is defined in section [3.1.1.2.3](#).

An Add or Modify request on an [attributeSchema](#) object succeeds only if the resulting object passes all of the following tests.

- The value of [LDAPDisplayName](#) is syntactically valid, per [\[RFC2251\]](#) section 4.1.4.
- The values of [attributeID](#), [LDAPDisplayName](#), [mAPIID](#) (if present) and [schemaIDGUID](#) are "Unique".
- A nonzero [linkID](#), if any, is unique among all values of the [linkID](#) attribute on objects in the schema NC, regardless of forest functional level. If a [linkID](#) is an odd number, it is not one, and an object exists whose [linkID](#) is the even number one smaller.
- The values of [attributeSyntax](#), [oMSyntax](#), and [oObjectClass](#) match some defined syntax (section [3.1.1.2.2](#)).
- Flag fANR is only present in the [searchFlags](#) attribute if the syntax is String(Unicode), String(IA5), String(Printable), String(Teletex) or String(Case).
- If [rangeLower](#) and [rangeUpper](#) are present, [rangeLower](#) is smaller than or equal to [rangeUpper](#).



An Add or Modify request on a [classSchema](#) object succeeds only if the resulting object passes all of the following tests.

- The value of [IDAPDisplayName](#) is syntactically valid, per [\[RFC2251\]](#) section 4.1.4.
- The values of [governsID](#), [IDAPDisplayName](#), and [schemaIDGUID](#) are "Unique".
- All attributes that are referenced in the [systemMayContain](#), [mayContain](#), [systemMustContain](#), and [mustContain](#) lists exist and are active.
- All classes that are referenced in the [subClassOf](#), [systemAuxiliaryClass](#), [auxiliaryClass](#), [systemPossSuperiors](#), and [possSuperiors](#) lists exist and are active.
- All classes in the [systemAuxiliaryClass](#) and [auxiliaryClass](#) attributes have either 88 class or auxiliary class specified as their [objectClassCategory](#).
- All classes in the [systemPossSuperiors](#) and [possSuperiors](#) attributes have either 88 class or structural class specified as their [objectClassCategory](#).
- The superclass chain of a class follows the rules for inheritance as specified in section [3.1.1.2.4.2](#).
- The [dynamicObject](#) class is not referenced by the [subClassOf](#) attribute of a class.
- The attribute specified in the [rDNAttID](#) attribute has syntax String(Unicode).
- Attribute [defaultSecurityDescriptor](#), if present, is a valid SDDL string.

### 3.1.1.2.5.1.2 Safety Checks

The following checks reduce the possibility of schema updates by one application breaking another application.

These checks apply to all schema objects:

- A Modify adds no attributes to the [mustContain](#) or [systemMustContain](#) of an existing class.
- A Modify does not add an auxiliary class to the [auxiliaryClass](#) or [systemAuxiliaryClass](#) of an existing class, if doing so would effectively add either [mustContain](#) or [systemMustContain](#) attributes to the class.
- A Modify does not change the [objectClassCategory](#) of an existing class.
- A Modify does not change a **constructed attribute** (an attribute with FLAG\_ATTR\_IS\_CONSTRUCTED in [systemFlags](#)).
- A Modify does not change class [top](#), except to add back link attributes as may-contains, either by adding back link attributes to [mayContain](#) of [top](#), or by adding auxiliary classes to [auxiliaryClass](#) of [top](#) whose only effect on [top](#) is adding back link attributes as may-contains.
- A Modify does not change the [subSchema](#) object.
- A Modify does not change fRODCFilteredAttribute bit of the [searchFlags](#) attribute of an [attributeSchema](#) object, if the DC functionality level is DS\_BEHAVIOR\_WIN2008 or higher, and the [attributeSchema](#) object cannot be a member of the filtered attribute set (see section [3.1.1.2.3.5](#)).

These checks apply to schema objects that include FLAG\_SCHEMA\_BASE\_OBJECT in the [systemFlags](#) attribute:

- A Modify does not change the [LDAPDisplayName](#) or [cn](#) of an [attributeSchema](#) or [classSchema](#) object, or the [defaultObjectCategory](#) of a [classSchema](#) object.
- A Modify does not change the [classSchema](#) objects [attributeSchema](#), [classSchema](#), [subSchema](#) and [dMD](#).
- A Modify does not change the fCONFIDENTIAL bit of the [searchFlags](#) attribute of an [attributeSchema](#) object.
- A Modify does not change the [attributeSecurityGUID](#) on the following fixed list of [attributeSchema](#) objects: [accountExpires](#), [badPwdCount](#), [codePage](#), [countryCode](#), [description](#), [displayName](#), [domainReplica](#), [forceLogoff](#), [homeDirectory](#), [homeDrive](#), [memberOf](#), [lastLogoff](#), [lastLogon](#), [lockOutObservationWindow](#), [lockoutDuration](#), [lockoutThreshold](#), [logonCount](#), [logonHours](#), [logonWorkstation](#), [maxPwdAge](#), [member](#), [minPwdAge](#), [minPwdLength](#), [modifiedCount](#), [objectSid](#), [oEMInformation](#), [profilePath](#), [primaryGroupID](#), [pwdHistoryLength](#), [pwdProperties](#), [sAMAccountName](#), [scriptPath](#), [serverState](#), [serverRole](#), [uASCompat](#), [comment](#), [pwdLastSet](#), [userAccountControl](#), [userParameters](#).

### 3.1.1.2.5.2 Defunct

A schema object with [isDefunct](#) = true is defunct; a schema object that is not defunct is active. This section documents the special behavior of [attributeSchema](#) and [classSchema](#) objects related to the defunct state.

The effect of being defunct depends upon the forest functional level as specified in the following subsections. The following statements are independent of the forest functional level.

- The [isDefunct](#) attribute being not present on an [attributeSchema](#) or [classSchema](#) object is equivalent to [isDefunct](#) = false; modifications that move between these two representations of the active state have no special behavior.
- If an **LDAP** Modify changes the [isDefunct](#) attribute (giving it a value of true or false, or removing it), this change must be the only change in the **LDAP** Modify request; otherwise, the request fails with **LDAP** error *unwillingToPerform*.
- If a Modify sets [isDefunct](#) to true but the [attributeSchema](#) or [classSchema](#) object is base (i.e. has [FLAG\\_SCHEMA\\_BASE\\_OBJECT](#) present in its [systemFlags](#) attribute), the Modify fails, with **LDAP** error *unwillingToPerform*.
- LDAP Add cannot create instances of a defunct class (section [3.1.1.5.2.2](#)), and **LDAP** Add and Modify cannot create instances of a defunct attribute (section [3.1.1.5.2.2](#), [3.1.1.5.3.2](#)).
- Making an [attributeSchema](#) or [classSchema](#) object defunct has no effect on the state of existing objects that use the defunct attribute or class, but it changes the behavior of reads and updates of such objects as described in sections [3.1.1.4.8](#) (Search), [3.1.1.5.2.2](#) (Add), [3.1.1.5.3.2](#) (Modify), and [3.1.1.5.5](#) (Delete).

### 3.1.1.2.5.2.1 Forest Functional Level Less Than WIN2003

If the forest functional level is less than DS\_BEHAVIOR\_WIN2003, a DC behaves as follows with respect to the defunct state.

- The [isDefunct](#) attribute can be changed from not present (or false) to true on an [attributeSchema](#) or [classSchema](#) object. This modification is subject to the following checks:

- If the modification is to an [attributeSchema](#) object and the object is a [mustContain](#), [systemMustContain](#), [mayContain](#), or [systemMayContain](#) of an active class, the modification fails.
- If the modification is to a [classSchema](#) object and the object is a [subClassOf](#), [auxiliaryClass](#), or [possSuperiors](#) of an active class, the modification fails.

The **LDAP** error if the [isDefunct](#) modification fails is *unwillingToPerform*.

- When [isDefunct](#) is true on an [attributeSchema](#) or [classSchema](#) object, an **LDAP** Modify can set [isDefunct](#) to false (or remove the [isDefunct](#) attribute). This modification is subject to the following check:
  - If the modification is to a [classSchema](#) object and the object references any defunct attributes through its [mustContain](#), [systemMustContain](#), [mayContain](#), or [systemMayContain](#) attributes, or references any defunct classes through its [subClassOf](#), [auxiliaryClass](#), or [possSuperiors](#) attributes, the modification fails.

The **LDAP** error if the [isDefunct](#) modification fails is *unwillingToPerform*.

- No other modification to a defunct [attributeSchema](#) or [classSchema](#) object is allowed. The **LDAP** error if the modification fails is *noSuchObject*.

### 3.1.1.2.5.2.2 Forest Functional Level WIN2003 or Greater

If the forest functional level is DS\_BEHAVIOR\_WIN2003 or greater, a DC behaves as follows with respect to the defunct state.

- An **LDAP** Modify can change the [isDefunct](#) attribute from not present (or false) to true on an [attributeSchema](#) or [classSchema](#) object. This modification is subject to the following checks, in addition to the checks performed when the forest functional level is less than DS\_BEHAVIOR\_WIN2003:
  - If the modification is to an [attributeSchema](#) object and the object is a [mustContain](#), [systemMustContain](#), [mayContain](#), [systemMayContain](#), or [rDNAttID](#) of an active class, the modification fails.
  - If the modification is to a [classSchema](#) object and the object is a [subClassOf](#), [auxiliaryClass](#), or [possSuperiors](#) of an active class, the modification fails.

The **LDAP** error if the [isDefunct](#) modification fails is *unwillingToPerform*.

- An **LDAP** Modify can change the [isDefunct](#) attribute from true to false (or not present) on an [attributeSchema](#) or [classSchema](#) object. This modification is subject to the following checks, in addition to the checks performed when the forest functional level is less than DS\_BEHAVIOR\_WIN2003:
  - If the modification is to a [classSchema](#) object and the object references any defunct attributes through its [mustContain](#), [systemMustContain](#), [mayContain](#), [systemMayContain](#) or [rDNAttID](#) attributes, or references any defunct classes through its [subClassOf](#), [auxiliaryClass](#), or [possSuperiors](#) attributes, the modification fails.
  - The same uniqueness checks are performed when setting [isDefunct](#) to false as would have been performed if the same object were being added to a schema where it was not present. In particular, the uniqueness checks on [attributeID](#), [governsID](#), [schemaIDGUID](#), [mAPIID](#), [linkID](#), and [LDAPDisplayName](#) must pass.

The **LDAP** error if the [isDefunct](#) modification fails is *unwillingToPerform*.

- An **LDAP** Modify can change the other attributes of defunct schema objects subject to the same checks that apply to changes to active schema objects.

Therefore, for instance, a Modify can change the [LDAPDisplayName](#) of a defunct [attributeSchema](#) object, or the [LDAPDisplayName](#), [mustContain](#), [mayContain](#), [subClassOf](#), [auxiliaryClass](#), and [possSuperiors](#) of a defunct [classSchema](#) object.

Because the checks that apply to changes to active schema objects are still in force, Modify cannot (for instance) change the [attributeID](#), [governsID](#), [schemaIDGUID](#), [mAPIID](#), [linkID](#), [attributeSyntax](#), [oMSyntax](#), and [oMOBJECTClass](#) attributes of defunct schema objects.

- Section [3.1.1.4.8](#) specifies the effects of the defunct state on reads of OID-valued attributes that identify schema objects ([mustContain](#), [systemMustContain](#), [mayContain](#), [systemMayContain](#), [subClassOf](#), [auxiliaryClass](#), and [possSuperiors](#)).

### 3.1.1.2.6 ATTRTYP

Any OID-valued quantity stored on an object is stored as an ATTRTYP, a 32-bit unsigned integer. The ATTRTYP space is 32 bits wide and is divided into these ranges:

Range	Description
[0x00000000..0x7FFFFFFF]	ATTRTYPs that map to OIDs via the prefix table.
[0x80000000..0xBFFFFFFF]	ATTRTYPs used as values of <a href="#">msDS-IntId</a> attribute.
[0xC0000000..0xFFFFEFFF]	Reserved for future use.
[0xFFFF0000.. 0xFFFFFFFF]	Reserved for internal use (never appear on wire).

The mapping from ATTRTYP A to OID O works as follows:

- If A in [0x00000000..0x7FFFFFFF], A maps to O via a prefix table as specified in [\[MS-DRSR\]](#) section 5.15.4 (procedure *OidFromAttid*).
- If A in [0x80000000..0xBFFFFFFF], let X be the object such that X![msDS-IntId](#) equals A. If X is an [attributeSchema](#) object, O is X![attributeID](#); otherwise X is an [classSchema](#) object, and O is X![governsID](#).

### 3.1.1.3 LDAP Protocol

Active Directory is a server for the **LDAP** protocol. This section specifies the extensions and variations of the **LDAP** protocol supported by Active Directory. Except as otherwise noted, all material applies to both AD/DS and AD/LDS. Also, except as noted, all information applies to all versions of AD/DS and AD/LDS.

This section is structured as follows:

- Section [3.1.1.3.1](#) documents the interpretation of **LDAP** RFCs made by Active Directory, and deviations from the **LDAP** RFCs.
- The **rootDSE** (empty DN) is a mechanism for clients of an **LDAP** server to interact with the server itself, rather than with particular objects contained by the server. Section [3.1.1.3.2](#) specifies the rootDSE reads supported by Active Directory, and section [3.1.1.3.3](#) specifies the rootDSE updates.

- LDAP has several extension mechanisms in addition to the rootDSE. Section [3.1.1.3.4](#) specifies the **LDAP** extensions that Active Directory supports.

### 3.1.1.3.1 LDAP Conformance

The purpose of this section is to document how the implementation of Active Directory DCs interprets the **LDAP** v3 RFCs, including differences from those RFCs. Except as noted below, Active Directory is compliant to [\[RFC3377\]](#). All error codes returned by Active Directory are taken from the resultCode enumeration of the LDAPResult structure defined in [\[RFC2251\]](#) section 4.1.10.

#### 3.1.1.3.1.1 Schema

This section discusses the implementation of the schema in Active Directory DCs, as it relates to the IETF RFC standards for **LDAP** schemas.

##### 3.1.1.3.1.1.1 subSchema

Per [\[RFC2251\]](#) and [\[RFC2252\]](#), Active Directory exposes a [subSchema](#) object that is pointed to by the [subSchemaSubEntry](#) attribute on the rootDSE. In accord with [\[RFC2251\]](#) section 3.2.2, this [subSchema](#) object contains the required [cn](#), [objectClass](#), [objectClasses](#), and [attributeTypes](#) attributes. Additionally, it contains the [dITContentRules](#) attribute. It does not contain the [matchingRules](#), [matchingRuleUse](#), [dITStructureRules](#), [nameForms](#), or [ldapSyntaxes](#) attributes. It contains the [modifyTimeStamp](#) attribute, but not the [createTimeStamp](#) attribute. In contrast to [\[RFC2252\]](#) section 7.2, in Active Directory the [subSchema](#) class is defined to be structural rather than auxiliary.

The meaning of the [attributeTypes](#), [objectClasses](#), and [dITContentRules](#) attributes are as described in those RFCs. However, the values stored in these attributes use only a subset of the AttributeTypeDescription, ObjectClassDescription, and DITContentRuleDescription grammars described in [\[RFC2252\]](#). The grammars used by Active Directory are shown below. Other than the removal of certain elements, these grammars are identical to those of [\[RFC2252\]](#).

```
AttributeTypeDescription = "(" whsp
    numericoid whsp          ; attributeID
    [ "NAME" qdescrs ]       ; LDAPDisplayName
    [ "SYNTAX" whsp noidlen whsp ] ; see RFC 2252 section 4.3
    [ "SINGLE-VALUE" whsp ]   ; default multi-valued
    [ "NO-USER-MODIFICATION" whsp ] ; default user modifiable
    whsp ")"
ObjectClassDescription = "(" whsp
    numericoid whsp          ; governsID
    [ "NAME" qdescrs ]       ; LDAPDisplayName
    [ "SUP" oids ]           ; governsIDs of superior object classes
    [ ( "ABSTRACT" / "STRUCTURAL" / "AUXILIARY" ) whsp ]
                                ; default structural
    [ "MUST" oids ]          ; attributeIDs of required attributes
    [ "MAY" oids ]           ; attributeIDs of optional attributes
    whsp ")"
DITContentRuleDescription = "("
    numericoid               ; governsID of structural object class
    [ "NAME" qdescrs ]       ; LDAPDisplayName
    [ "AUX" oids ]           ; governsIDs of auxiliary classes
    [ "MUST" oids ]          ; attributeIDs of required attributes
    [ "MAY" oids ]           ; attributeIDs of optional attributes
    ")"
```

In addition to the above, Active Directory contains two additional [subSchema](#) attributes, named [extendedClassInfo](#) and [extendedAttributeInfo](#). These return additional data about the classes and attributes in a format similar to [objectClasses](#) and [attributeTypes](#), respectively. The grammar used for [extendedClassInfo](#) is as follows:

```
ObjectClassDescriptionExtended = "(" whsp
    numericoid whsp                ; governsID
    [ "NAME" qdescrs ]             ; ldapDisplayName
    [ "CLASS-GUID" whsp guid ]     ; schemaIDGUID
    whsp ")"
```

The NAME field is as in the ObjectClassDescription grammar. The CLASS-GUID field contains the value of the class's [schemaIDGUID](#) attribute. That value, which is a **GUID**, is expressed not in the dashed-string **GUID** format of [\[RFC4122\]](#) section 3 but rather as the hexadecimal representation of the binary format of the **GUID**. For example, the **GUID** whose dashed-string representation is "3fdfee4f-47f4-11d1-a9c3-0000f80367c1" would be expressed as "4feedf3ff447d111a9c30000f80367c1" in the CLASS-GUID field.

The grammar for [extendedAttributeInfo](#) is as follows:

```
AttributeTypeDescriptionExtended = "(" whsp
    numericoid whsp                ; attributeID
    [ "NAME" qdescrs ]             ; ldapDisplayName
    [ "RANGE-LOWER" whsp numericstring ] ; rangeLower
    [ "RANGE-UPPER" whsp numericstring ] ; rangeUpper
    [ "PROPERTY-GUID" whsp guid ]     ; schemaIDGUID
    [ "PROPERTY-SET-GUID" whsp guid ] ; attributeSecurityGUID
    [ "INDEXED" whsp ]               ; fATTINDEX in searchFlags
    [ "SYSTEM-ONLY" whsp ]           ; systemOnly
    whsp ")"
```

The NAME field is as in the AttributeTypeDescription grammar. The RANGE-LOWER and RANGE-UPPER fields are only present if the attribute's [attributeSchema](#) contains values for the [rangeLower](#) and [rangeUpper](#) attributes, respectively. If present, those fields contain the values of those attributes. The PROPERTY-GUID field contains the value of the attribute's [schemaIDGUID](#). If the attribute has a [attributeSecurityGUID](#) attribute, the PROPERTY-SET-GUID field contains the value of that attribute; otherwise, it contains the value of the NULL **GUID** (the **GUID** consisting entirely of zeros). For both PROPERTY-GUID and PROPERTY-SET-GUID, the **GUID** is represented in the same form as that CLASS-GUID from the ObjectClassDescriptionExtended grammar. If the fATTINDEX bit of the attribute's [searchFlags](#) is set, the INDEXED field is present. If the attribute's [systemOnly](#) attribute is true, the SYSTEM-ONLY field is present.

The [attributeTypes](#), [objectClasses](#), [dITContentRules](#), [extendedClassInfo](#), and [extendedAttributeInfo](#) attributes on the [subSchema](#) object are read-only. They permit applications to discover the schema on the DC, but they are not the mechanism for changing the schema on the DC. DCs change their schema in response to the addition or modification of [classSchema](#) and [attributeSchema](#) objects in the schema NC. These objects also contain attributes which supply additional information about the schema that is not present in the attributes of the [subSchema](#) object, such as the [systemFlags](#) attribute, which specifies additional properties of an attribute (for example, whether it is a **constructed attribute**). The [attributeSchema](#) and [classSchema](#) objects and their associated attributes are specified in section [3.1.1.2](#).

If the forest functional level is DS\_BEHAVIOR\_WIN2003 or greater, the [attributeTypes](#), [dITContentRules](#), [extendedAttributeInfo](#), [extendedClassInfo](#), and [objectClasses](#) attributes on the [subSchema](#) object do not contain defunct attributes or classes, only active attributes or classes.

### 3.1.1.3.1.1.2 Syntaxes

The syntaxes used in Active Directory are based on [\[RFC2252\]](#) section 6. Where Active Directory and [\[RFC2252\]](#) have syntaxes in common, the same means of encoding the value into the syntax is used. However, Active Directory has a number of syntaxes that are not defined in [\[RFC2252\]](#), and vice versa. Additionally, even when Active Directory and [\[RFC2252\]](#) have syntaxes in common, in many cases they use different names for the same syntax, and in all cases they use different OIDs to identify the same syntax.

Active Directory does not use the syntaxes defined in [\[RFC2256\]](#) section 6. The list of syntaxes in Active Directory, their encodings, and how they map to the [\[RFC2252\]](#) syntaxes are documented in [3.1.1.2.2](#).

### 3.1.1.3.1.1.3 Attributes

Sections 5.1 through 5.4 of [\[RFC2252\]](#), as well as section 5 of [\[RFC2256\]](#) and section 2 of [\[RFC2798\]](#) define a set of attributes common to **LDAP** directories. Additionally, portions of the Active Directory schema are derived from [\[RFC1274\]](#) and [\[RFC2307\]](#). The following tables show, for each of these RFCs, the attributes included in Active Directory's default schema in Windows Server 2003 and Windows Server 2008. Some of these attributes were added to the schema of Windows Server 2003 or Windows Server 2003 R2, but were not present in the Windows 2000 schema; [\[MS-ADA1\]](#), [\[MS-ADA2\]](#), and [\[MS-ADA3\]](#) specify the attributes included in each version of the schema.

RFC 1274

Attribute	Included by AD/DS?	Included by AD/LDS?
<a href="#">objectClass</a>	Yes	Yes
<a href="#">knowledgeInformation</a>	Yes	No
<a href="#">serialNumber</a>	Yes	No
<a href="#">streetAddress</a>	Yes	Yes
<a href="#">title</a>	Yes	No
<a href="#">description</a>	Yes	Yes
<a href="#">searchGuide</a>	Yes	Yes
<a href="#">businessCategory</a>	Yes	Yes
<a href="#">postalAddress</a>	Yes	Yes
<a href="#">postalCode</a>	Yes	Yes
<a href="#">postOfficeBox</a>	Yes	Yes
<a href="#">physicalDeliveryOfficeName</a>	Yes	Yes
<a href="#">telephoneNumber</a>	Yes	Yes

Attribute	Included by AD/DS?	Included by AD/LDS?
<a href="#">telexNumber</a>	Yes	Yes
<a href="#">teletexTerminalIdentifier</a>	Yes	Yes
<a href="#">facsimileTelephoneNumber</a>	Yes	Yes
<a href="#">x121Address</a>	Yes	Yes
<a href="#">internationalISDNNumber</a>	Yes	Yes
<a href="#">registeredAddress</a>	Yes	Yes
<a href="#">destinationIndicator</a>	Yes	Yes
<a href="#">preferredDeliveryMethod</a>	Yes	Yes
<a href="#">presentationAddress</a>	Yes	No
<a href="#">supportedApplicationContext</a>	Yes	No
<a href="#">member</a>	Yes	Yes
<a href="#">owner</a>	Yes	No
<a href="#">roleOccupant</a>	Yes	No
<a href="#">seeAlso</a>	Yes	Yes
<a href="#">userPassword</a>	Yes*	Yes*
<a href="#">userCertificate</a>	Yes	No
<a href="#">cACertificate</a>	Yes	No
<a href="#">authorityRevocationList</a>	Yes	No
<a href="#">certificateRevocationList</a>	Yes	No
<a href="#">crossCertificatePair</a>	Yes	No
<a href="#">textEncodedORAddress</a>	Yes	No
<a href="#">roomNumber</a>	Yes	No
<a href="#">photo</a>	Yes	No
<a href="#">userClass</a>	Yes	No
<a href="#">host</a>	Yes	No
<a href="#">manager</a>	Yes	No
<a href="#">documentIdentifier</a>	Yes	No
<a href="#">documentTitle</a>	Yes	No
<a href="#">documentVersion</a>	Yes	No



Attribute	Included by AD/DS?	Included by AD/LDS?
<a href="#">documentAuthor</a>	Yes	No
<a href="#">documentLocation</a>	Yes	No
<a href="#">secretary</a>	Yes	No
<a href="#">otherMailbox</a>	Yes	No
<a href="#">associatedDomain</a>	Yes	No
<a href="#">associatedName</a>	Yes	No
<a href="#">homePostalAddress</a>	Yes	No
<a href="#">personalTitle</a>	Yes	No
<a href="#">organizationalStatus</a>	Yes	No
<a href="#">buildingName</a>	Yes	No
<a href="#">audio</a>	Yes	No
<a href="#">documentPublisher</a>	Yes	No
<a href="#">aliasedObjectName</a>	No	No
<a href="#">commonName</a>	No	No
<a href="#">surname</a>	No	No
<a href="#">countryName</a>	No	No
<a href="#">localityName</a>	No	No
<a href="#">stateOrProvinceName</a>	No	No
<a href="#">organizationName</a>	No	No
<a href="#">mhsDeliverableContentLength</a>	No	No
<a href="#">mhsDeliverableContentTypes</a>	No	No
<a href="#">mhsDeliverableEits</a>	No	No
<a href="#">mhsDLMembers</a>	No	No
<a href="#">mhsDLSubmitPermissions</a>	No	No
<a href="#">mhsMessageStoreName</a>	No	No
<a href="#">mhsORAddresses</a>	No	No
<a href="#">mhsPreferredDeliveryMethods</a>	No	No
<a href="#">mhsSupportedAutomaticActions</a>	No	No
<a href="#">mhsSupportedContentTypes</a>	No	No

Attribute	Included by AD/DS?	Included by AD/LDS?
<a href="#">mhsSupportedOptionalAttributes</a>	No	No
<a href="#">userid</a>	No	No
<a href="#">rfc822Mailbox</a>	No	No
<a href="#">info</a>	No	No
<a href="#">favouriteDrink</a>	No	No
<a href="#">homeTelephoneNumber</a>	No	No
<a href="#">lastModifiedTime</a>	No	No
<a href="#">lastModifiedBy</a>	No	No
<a href="#">domainComponent</a>	No	No
<a href="#">aRecord</a>	No	No
<a href="#">mXRecord</a>	No	No
<a href="#">nSRecord</a>	No	No
<a href="#">sOARRecord</a>	No	No
<a href="#">cNAMERRecord</a>	No	No
<a href="#">mobileTelephoneNumber</a>	No	No
<a href="#">pagerTelephoneNumber</a>	No	No
<a href="#">friendlyCountryName</a>	No	No
<a href="#">uniqueIdentifier</a>	No	No
<a href="#">janetMailbox</a>	No	No
<a href="#">mailPreferenceOption</a>	No	No
<a href="#">dSAQuality</a>	No	No
<a href="#">singleLevelQuality</a>	No	No
<a href="#">subtreeMinimumQuality</a>	No	No
<a href="#">subtreeMaximumQuality</a>	No	No
<a href="#">personalSignature</a>	No	No
<a href="#">dITRedirect</a>	No	No

\* Active Directory uses the [userPassword](#) attribute to set or change passwords only in limited circumstances. See section [3.1.1.3.1.5](#).

RFC 2252

Attribute	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">createTimeStamp</a>	Yes	Yes
<a href="#">modifyTimeStamp</a>	Yes	Yes
<a href="#">subSchemaSubEntry</a>	Yes	Yes
<a href="#">attributeTypes</a>	Yes	Yes
<a href="#">objectClasses</a>	Yes	Yes
<a href="#">namingContexts</a>	Yes	Yes
<a href="#">supportedExtension</a>	Yes	Yes
<a href="#">supportedControl</a>	Yes	Yes
<a href="#">supportedSASLMechanisms</a>	Yes	Yes
<a href="#">supportedLDAPVersion</a>	Yes	Yes
<a href="#">dITContentRules</a>	Yes	Yes
<a href="#">creatorsName</a>	No	No
<a href="#">modifiersName</a>	No	No
<a href="#">matchingRules</a>	No	No
<a href="#">matchingRulesUse</a>	No	No
<a href="#">altServer</a>	No	No
<a href="#">ldapSyntaxes</a>	No	No
<a href="#">dITStructureRules</a>	No	No
<a href="#">nameForms</a>	No	No

#### RFC 2256

Attribute	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">objectClass</a>	Yes	Yes
<a href="#">knowledgeInformation</a>	Yes	No
<a href="#">cn</a>	Yes	Yes
<a href="#">sn</a>	Yes	No
<a href="#">serialNumber</a>	Yes	No
<a href="#">c</a>	Yes	Yes
<a href="#">l</a>	Yes	Yes
<a href="#">st</a>	Yes	Yes

Attribute	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">street</a>	Yes	Yes
<a href="#">o</a>	Yes	Yes
<a href="#">ou</a>	Yes	Yes
<a href="#">title</a>	Yes	No
<a href="#">description</a>	Yes	Yes
<a href="#">searchGuide</a>	Yes	Yes
<a href="#">businessCategory</a>	Yes	Yes
<a href="#">postalAddress</a>	Yes	Yes
<a href="#">postalCode</a>	Yes	Yes
<a href="#">postOfficeBox</a>	Yes	Yes
<a href="#">physicalDeliveryOfficeName</a>	Yes	Yes
<a href="#">telephoneNumber</a>	Yes	Yes
<a href="#">telexNumber</a>	Yes	Yes
<a href="#">teletexTerminalIdentifier</a>	Yes	Yes
<a href="#">facsimileTelephoneNumber</a>	Yes	Yes
<a href="#">x121Address</a>	Yes	Yes
<a href="#">internationalISDNNumber</a>	Yes	Yes
<a href="#">registeredAddress</a>	Yes	Yes
<a href="#">destinationIndicator</a>	Yes	Yes
<a href="#">preferredDeliveryMethod</a>	Yes	Yes
<a href="#">presentationAddress</a>	Yes	No
<a href="#">supportedApplicationContext</a>	Yes	No
<a href="#">member</a>	Yes	Yes
<a href="#">owner</a>	Yes	No
<a href="#">roleOccupant</a>	Yes	No
<a href="#">seeAlso</a>	Yes	Yes
<a href="#">userPassword</a>	Yes*	Yes*
<a href="#">userCertificate</a>	Yes	No
<a href="#">cACertificate</a>	Yes	No

Attribute	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">authorityRevocationList</a>	Yes	No
<a href="#">certificateRevocationList</a>	Yes	No
<a href="#">crossCertificatePair</a>	Yes	No
<a href="#">name</a>	Yes	Yes
<a href="#">givenName</a>	Yes	No
<a href="#">initials</a>	Yes	No
<a href="#">generationQualifier</a>	Yes	No
<a href="#">x500uniqueIdentifier</a>	Yes	No
<a href="#">distinguishedName</a>	Yes	Yes
<a href="#">uniqueMember</a>	Yes	Yes
<a href="#">houseIdentifier</a>	Yes	No
<a href="#">deltaRevocationList</a>	Yes	No
<a href="#">dmdName</a>	Yes	Yes
<a href="#">aliasedObjectName</a>	No	No
<a href="#">dnQualifier</a>	No	No
<a href="#">protocolInformation</a>	No	No
<a href="#">supportedAlgorithms</a>	No	No

\* Active Directory uses the [userPassword](#) attribute to set or change passwords only in limited circumstances. See section [3.1.1.3.1.5](#).

#### RFC 2798

Attribute	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">carLicense</a>	Yes	No
<a href="#">departmentNumber</a>	Yes	No
<a href="#">displayName</a>	Yes	Yes
<a href="#">employeeNumber</a>	Yes	No
<a href="#">employeeType</a>	Yes	No
<a href="#">jpegPhoto</a>	Yes	No
<a href="#">preferredLanguage</a>	Yes	No
<a href="#">userSMIMECertificate</a>	Yes	No

Attribute	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">userPKCS12</a>	Yes	No

#### RFC 2307

Attribute	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">uidNumber</a>	Yes	No
<a href="#">gidNumber</a>	Yes	No
<a href="#">gecos</a>	Yes	No
<a href="#">homeDirectory</a>	Yes	No
<a href="#">loginShell</a>	Yes	No
<a href="#">shadowLastChange</a>	Yes	No
<a href="#">shadowMin</a>	Yes	No
<a href="#">shadowMax</a>	Yes	No
<a href="#">shadowWarning</a>	Yes	No
<a href="#">shadowInactive</a>	Yes	No
<a href="#">shadowExpire</a>	Yes	No
<a href="#">shadowFlag</a>	Yes	No
<a href="#">memberUid</a>	Yes	No
<a href="#">memberNisNetgroup</a>	Yes	No
<a href="#">nisNetgroupTriple</a>	Yes	No
<a href="#">ipServicePort</a>	Yes	No
<a href="#">ipServiceProtocol</a>	Yes	No
<a href="#">ipProtocolNumber</a>	Yes	No
<a href="#">oncRpcNumber</a>	Yes	No
<a href="#">ipHostNumber</a>	Yes	No
<a href="#">ipNetworkNumber</a>	Yes	No
<a href="#">ipNetmaskNumber</a>	Yes	No
<a href="#">macAddress</a>	Yes	No
<a href="#">bootParameter</a>	Yes	No
<a href="#">bootFile</a>	Yes	No
<a href="#">nisMapName</a>	Yes	No

Attribute	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">nisMapEntry</a>	Yes	No

### 3.1.1.3.1.1.4 Classes

Section 7 of [\[RFC2252\]](#), as well as section 7 of [\[RFC2256\]](#) and section 3 of [\[RFC2798\]](#) define a set of classes common to **LDAP** directories. In addition, portions of the Active Directory schema are derived from [\[RFC1274\]](#) and [\[RFC2307\]](#). The following tables show, for each of these RFCs, the classes included in Active Directory's default schema in Windows Server 2003 and Windows Server 2008. Some of these classes were added to the schema of Windows Server 2003 or Windows Server 2003 R2, but were not present in the Windows 2000 schema; [\[MS-ADSC\]](#) specifies the classes included in each version of the schema.

#### RFC 1274

Class	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">top</a>	Yes	Yes
<a href="#">country</a>	Yes	Yes
<a href="#">locality</a>	Yes	Yes
<a href="#">organization</a>	Yes	Yes
<a href="#">organizationalUnit</a>	Yes	Yes
<a href="#">person</a>	Yes	No
<a href="#">organizationalPerson</a>	Yes	No
<a href="#">organizationalRole</a>	Yes	No
<a href="#">groupOfNames</a>	Yes	No
<a href="#">residentialPerson</a>	Yes	No
<a href="#">applicationProcess</a>	Yes	No
<a href="#">applicationEntity</a>	Yes	No
<a href="#">dSA</a>	Yes	No
<a href="#">device</a>	Yes	Yes
<a href="#">certificationAuthority</a>	Yes	No
<a href="#">account</a>	Yes	No
<a href="#">document</a>	Yes	No
<a href="#">room</a>	Yes	No
<a href="#">documentSeries</a>	Yes	No
<a href="#">domain</a>	Yes	Yes

Class	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">rFC822LocalPart</a>	Yes	No
<a href="#">domainRelatedObject</a>	Yes	No
<a href="#">friendlyCountry</a>	Yes	No
<a href="#">simpleSecurityObject</a>	Yes	No
<a href="#">alias</a>	No	No
<a href="#">strongAuthenticationUser</a>	No	No
<a href="#">mhsDistributionList</a>	No	No
<a href="#">mhsMessageStore</a>	No	No
<a href="#">mhsMessageTransferAgent</a>	No	No
<a href="#">mhsOrganizationalUser</a>	No	No
<a href="#">mhsResidentialUser</a>	No	No
<a href="#">mhsUserAgent</a>	No	No
<a href="#">pilotObject</a>	No	No
<a href="#">pilotPerson</a>	No	No
<a href="#">dNSDomain</a>	No	No
<a href="#">pilotOrganization</a>	No	No
<a href="#">pilotDSA</a>	No	No
<a href="#">qualityLabelledData</a>	No	No

#### RFC 2252

Class	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">subSchema</a>	Yes	Yes
<a href="#">extensibleObject</a>	No	No

#### RFC 2256

Class	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">top</a>	Yes	Yes
<a href="#">country</a>	Yes	Yes
<a href="#">locality</a>	Yes	Yes
<a href="#">organization</a>	Yes	Yes



Class	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">organizationalUnit</a>	Yes	Yes
<a href="#">person</a>	Yes	No
<a href="#">organizationalPerson</a>	Yes	No
<a href="#">organizationalRole</a>	Yes	No
<a href="#">groupOfNames</a>	Yes	No
<a href="#">residentialPerson</a>	Yes	No
<a href="#">applicationProcess</a>	Yes	No
<a href="#">applicationEntity</a>	Yes	No
<a href="#">dSA</a>	Yes	No
<a href="#">device</a>	Yes	Yes
<a href="#">certificationAuthority</a>	Yes	No
<a href="#">groupOfUniqueNames</a>	Yes	No
<a href="#">cRLDistributionPoint</a>	Yes	No
<a href="#">dMD</a>	Yes	Yes
<a href="#">alias</a>	No	No
<a href="#">strongAuthenticationUser</a>	No	No
<a href="#">userSecurityInformation</a>	No	No
<a href="#">certificationAuthority-V2</a>	No	No

#### RFC 2798

Class	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">inetOrgPerson</a>	Yes	No

#### RFC 2307

Class	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">posixAccount</a>	Yes	No
<a href="#">shadowAccount</a>	Yes	No
<a href="#">posixGroup</a>	Yes	No
<a href="#">ipService</a>	Yes	No
<a href="#">ipProtocol</a>	Yes	No

Class	Supported by AD/DS?	Supported by AD/LDS?
<a href="#">oncRpc</a>	Yes	No
<a href="#">ipHost</a>	Yes	No
<a href="#">ipNetwork</a>	Yes	No
<a href="#">nisNetgroup</a>	Yes	No
<a href="#">nisMap</a>	Yes	No
<a href="#">nisObject</a>	Yes	No
<a href="#">ieee802Device</a>	Yes	No
<a href="#">bootableDevice</a>	Yes	No

### 3.1.1.3.1.1.5 Auxiliary Classes

Windows 2000 had limited support for **LDAP** auxiliary classes. An auxiliary class would be associated with the schema definition of a particular class C when the auxiliary class was added to the [auxiliaryClass](#) or [systemAuxiliaryClass](#) attribute of the [classSchema](#) object which defines C. In this case, all instances of C will inherit the attributes of the auxiliary class.

The server permits adding or removing an auxiliary class to or from the [auxiliaryClass](#) attribute of C at any point in time. Doing so adds or removes the auxiliary class from every existing instance of C, but does not cause the object class of the auxiliary class to appear in the [objectClass](#) attribute of those instances. Such an auxiliary class can have optional ([mayContain](#)) attributes, but not mandatory ([mustContain](#)) attributes. This is because there can be existing instances of C, in which case adding a new mandatory attribute would cause those existing instances to violate the modified schema.

The server permits adding an auxiliary class to the [systemAuxiliaryClass](#) attribute of C only when C is defined, that is, when C's [classSchema](#) object is added to the schema NC. After a [classSchema](#) object has been created, its [systemAuxiliaryClass](#) attribute cannot be modified. An auxiliary class that is associated with C by the addition of it to C's [systemAuxiliaryClass](#) can have mandatory ([mustContain](#)) as well as optional ([mayContain](#)) attributes. As in the previous case, the auxiliary classes added in this manner are not shown in the [objectClass](#) attribute of the instances of C.

Windows Server 2003 introduced dynamic auxiliary class support in addition to Windows 2000's auxiliary class mechanism. This dynamic auxiliary class mechanism complies with the [\[RFC2252\]](#) model of auxiliary classes. In Windows Server 2003, the server permits adding an auxiliary class to any instance I of a class by a request to add that auxiliary class to I's [objectClass](#). This will cause only that instance I to inherit the attributes of the auxiliary class. The dynamic auxiliary class will be removed from I, after the values of all attributes in the auxiliary class have been cleared by the client, by a request to remove the auxiliary class from I's [objectClass](#). Dynamic auxiliary classes can have both mandatory ([mustContain](#)) and optional ([mayContain](#)) attributes.

If the dynamic auxiliary class that is added to I is a subclass of another auxiliary class, both auxiliary classes are added to I when the child auxiliary class is added to I. However, removing the child auxiliary class does not cause the server to remove its parent from I; each class must be explicitly removed by the client.

For each I, I's [objectClass](#) contains the structural, abstract, and dynamic **auxiliary object classes** of which I is an instance (and their inheritance chains). I's [structuralObjectClass](#) includes only the

structural class of which I is an instance and its inheritance chain. I![msDS-Auxiliary-Classes](#) contains the dynamic auxiliary classes of which I is an instance along with their inheritance chain, except it does not include those classes in the inheritance chain that are in I![structuralObjectClass](#).

### 3.1.1.3.1.2 Object Naming

This section discusses the naming of objects via distinguished names in Active Directory, as it differs from the appropriate RFCs.

#### 3.1.1.3.1.2.1 Naming Attributes

As with section 2.3 of [\[RFC2253\]](#), Active Directory permits any attribute to be used as the AttributeType in an RDN. However, Active Directory imposes the additional restriction that the AttributeType used must be of String(Unicode) syntax. Furthermore, all objects of the same class use the same attribute in their RDN. The attribute to be used in the RDN is specified by the [rDNAttID](#) attribute in the [classSchema](#) object that defines the class. The [rDNAttID](#) attribute contains the attribute to be used in the RDN. Multivalued RDNs are not permitted (see section [3.1.1.3.1.2.3](#)), so if the attribute A specified by [rDNAttID](#) is multivalued, an attempt to add an additional value to A on an object O for which O![rDNAttID](#) = A is rejected with the error *invalidDNSyntax* if it takes place at the time of the object's creation, or the error *notAllowedOnRDN* if it takes place in a subsequent **LDAP** modify operation.

The AttributeValue of the RDN must be unique among sibling objects. For example, the following two DNs cannot coexist in the directory, because two identical AttributeValues ("Abc") would exist in the same **container** ("OU=Users,DC=Fabrikam,DC=com"):

- CN=Abc,OU=Users,DC=Fabrikam,DC=com
- L=Abc,OU=Users,DC=Fabrikam,DC=com

The server will reject an attempt to create such a non-uniquely named object with the error *entryAlreadyExists*. This requirement for unique AttributeValues guarantees the uniqueness of canonical names.

#### 3.1.1.3.1.2.2 NC Naming

The DN of a **domain** NC is derived from the DNS name of the **domain** using the transformation algorithm of section 3 of [\[RFC2247\]](#). The object at the root of each **domain** NC is a [domainDNS](#) object, in accord with section 5.2 of that RFC. The [rDNAttID](#) for the [domainDNS](#) class is [dc](#), in accord with section 4 of the RFC. While the same attribute OID is used for the [dc](#) attribute in Active Directory as in section 4 of the RFC, the syntax of the attribute in Active Directory is String(Unicode) rather than the specified String(IA5). The [dcObject](#) auxiliary class, specified in section 5.1 of the RFC, is not present in Active Directory.

When operating as AD/DS, the DN for the config NC is the RDN "CN=Configuration" followed by the DN of the **domain** NC of the **forest** root domain. When operating as AD/LDS, the DN for the config NC is the RDN "CN=Configuration, CN={guid}", where **guid** is a **GUID** in dashed-string form ([\[RFC4122\]](#) section 3). For example,

CN=Configuration, CN={FD783EE9-0216-4B83-8A2A-60E45AECCB81}

is a possible DN of the config NC when operating as AD/LDS.

The DN for the schema NC is the RDN "CN=Schema" followed by the DN of the config NC.

When operating as AD/DS, an application NC is named in the same way as a **domain** NC; the root of each AD/DS application NC is a [domainDNS](#) object. When operating as AD/LDS, the DN of an application NC consists of one or more RDNs.

### 3.1.1.3.1.2.3 Multivalued and Multiple-Attribute RDNs

Section 2 of [RFC2253](#) defines the following grammar rule for RelativeDistinguishedName, which explicitly allows RDNs to contain multiple attributes and values:

- RelativeDistinguishedName ::= SET SIZE (1..MAX) OF AttributeTypeAndValue

Active Directory is conformant with this rule, with the restriction that MAX equals 1 within the scope of the rule. As a result, multivalued RDNs that consist of multiple attributes (sometimes referred to as "multi-AVA RDNs"), or multiple instances of the same attribute, are both disallowed in Active Directory. An attempt to create such a DN is considered an attempt to create a syntactically invalid DN, and returns the error *invalidDNsyntax*. For example, assuming F is a multivalued attribute of String(Unicode) syntax, the following two DNs are both disallowed because they contain multivalued RDNs.

- F=John Smith+F=David Jones, OU=Users,DC=Fabrikam,DC=com
- F=John Smith+I=Redmond, OU=Users,DC=Fabrikam,DC=com

(Note that, if it is assumed that these DNs represent an object of a class C for which C![rDNAttID](#) = F, the second example is also disallowed because it contains the / attribute in the RDN. The server will return a *namingViolation* error when an attempt is made to add an **object of class** C whose RDN contains a different AttributeType than that declared in C![rDNAttID](#).)

### 3.1.1.3.1.2.4 Alternative Forms of DNs

In addition to the form of the DN defined in [RFC2253](#), Active Directory supports several alternative forms of DNs that can be used to specify objects in requests sent to the DC, for example, as the baseObject in a SearchRequest or as a AttributeValue in a ModifyRequest.

The first alternative form is in the following format:

```
<GUID=object_guid>
```

where **object\_guid** is a **GUID** that corresponds to the value of the [objectGUID](#) attribute of the object being specified. All DCs support **object\_guid** expressed as the hexadecimal representation of the binary form of a **GUID** ([\[MS-DTYP\]](#) section **2.3.2**). Windows Server 2003 and later DCs also support the dashed-string form of a **GUID** ([\[RFC4122\]](#) section 3).

The second alternative form is in the following format:

```
<SID=sid>
```

where **sid** is the **security identifier (SID)** that corresponds to the value of the [objectSid](#) attribute of the object being specified. **sid** is expressed as either the hexadecimal representation of a SID structure in binary form or as a Security Descriptor Definition Language (SDDL) SID string (the SID structure and the format of SDDL SID strings are defined in [\[MS-DTYP\]](#) sections [2.4.2](#) and [2.5.1](#)). The SDDL form of **sid** is not supported by Windows 2000 DCs. When using a SDDL SID string, the string must begin with "S-".

The third alternative form is the following format:

```
<WKGUID=guid, object_DN>
```

**guid** is a **GUID** expressed as the hexadecimal representation of the binary form of the **GUID**. A DN of this form is resolved to an object O by applying the following algorithm:

```
MapWellKnownGuidToDN(GUID guid, DN object_DN)
```

This algorithm resolves a well-known **GUID**, expressed as a **GUID**, **guid**, and an object, **object\_DN**, into the DN of the object O that is identified by that well-known **GUID**.

- If **object\_DN** does not name an object in the directory, reject the DN.
- Otherwise, let C be the object named by **object\_DN**.
- If there exists a value V in C![wellKnownObjects](#) such that the binary portion of V contains the same **GUID** as **guid**, then the DN of O is the DN portion of V.
- Otherwise, if there exists a value V' in C![otherWellKnownObjects](#) such that the binary portion of V' contains the same **GUID** as **guid**, then the DN of O is the DN portion of V'.
- Otherwise, reject the DN.

Normally, Active Directory will return DNs in the [\[RFC2253\]](#) format. However, clients can request that Active Directory return DNs in the "extended DN" format. This format combines an RFC 2253-style DN with a representation of the object's [objectGUID](#) and [objectSid](#) attributes. This form is documented in the **LDAP** section [3.1.1.3.4.1.5](#), which defines the LDAP\_SERVER\_EXTENDED\_DN\_OID control that is used by the client to request that the DC use the "extended DN" form when returning DNs. The "extended DN" form is not accepted as a means of specifying DNs in requests sent to the DC. The "extended DN" form is only used in **LDAP** responses from the DC, and only when the LDAP\_SERVER\_EXTENDED\_DN\_OID control is used to request such a form.

### 3.1.1.3.1.2.5 Alternative Form of SIDs

Attributes of String(SID) syntax contain a SID in binary form. However, a client may instead specify a value for such an attribute as a UTF-8 string that is a valid SDDL SID string beginning with "S-" (see [\[MS-DTYP\]](#) sections [2.4.2](#) and [2.5.1](#)). The server will convert such a string to the binary form of the SID and use that binary form as the value of the attribute.

### 3.1.1.3.1.3 Search Operations

#### 3.1.1.3.1.3.1 Search Filters

Active Directory does not support the extensible match rules defined in [\[RFC2252\]](#) section 8, [\[RFC2256\]](#) section 8, and [\[RFC2798\]](#) section 9. Active Directory exposes three extensible match rules that are defined in section [3.1.1.3.4.4](#). Other than these three rules, the rules that Active Directory uses for comparing values (for example, comparing two String(Unicode) attributes for equality or ordering) are not exposed as extensible match rules. These comparison rules are documented for each syntax type in section [3.1.1.2.2.4](#). When performing an extensible match search against Active Directory, if the type field of the MatchingRuleAssertion is not specified

([\[RFC2251\]](#) section 4.5.1), the extensible match filter clause is evaluated to "Undefined". The `dnAttributes` field of the `MatchingRuleAssertion` is ignored and always treated as if set to false.

Active Directory supports the `approxMatch` filter clause of [\[RFC2251\]](#) section 4.5.1. However, it is implemented identically to `equalityMatch`; for example, the filter is true if the values are equal. No approximation is performed. Filter clauses of the form "`(X=Y)`" and "`(X~=Y)`" may be freely substituted for each other.

Active Directory in Windows 2000 does not implement three-value logic for search filter evaluation as defined in [\[RFC2251\]](#) section 4.5.1. In Windows 2000, filters evaluate to either "true" or "false". Filters that would evaluate to "Undefined", as per the RFC, are instead evaluated to "false". Active Directory in Windows Server 2003 and later versions uses three-value logic for evaluating search filters, in conformance with the RFC.

### 3.1.1.3.1.3.2 Range Retrieval of Attribute Values

When retrieving the values from a multivalued attribute, Active Directory limits the number of values that can be retrieved from one attribute in a single search request. The maximum number of values that will be returned by Active Directory at one time is determined by the `MaxValRange` policy (see section [3.1.1.3.4.6](#)). To permit all the values of a multivalued attribute to be retrieved, Active Directory provides a "range retrieval" mechanism. This mechanism permits a client-specified subset of the values to be retrieved in a search request. By performing multiple search requests, each retrieving a distinct subset, the complete set of values for the attribute can be retrieved.

Range retrieval is requested by attaching a range option to the name of the attribute (for example, the `AttributeDescription`, as specified in [\[RFC2251\]](#) section 4.1.5) to be retrieved by the search request. This option takes the following form:

`range=low-high`

... where **low** is the zero-based index of the first value of the attribute to retrieve, and **high** is the zero-based index of the last value of the attribute to retrieve. For example, to retrieve the 100<sup>th</sup> through the 500<sup>th</sup> values of the `member` attribute, the attributes list in the `SearchRequest` would specify the `AttributeDescription` "`member;range=99-499`". Zero is used for **low** to specify the first entry. A client can substitute an asterisk for **high** to indicate all remaining entries (subject to any limitations imposed by the server on the maximum number of values to return). The server may return fewer values than requested.

When the server receives a range retrieval request, it will include a range option in the `AttributeDescription` returned. This range option will take the same form as above, with **low** indicating the zero-based index of the first value of the attribute that the server returned and **high** indicating the zero-based index of the last value of the attribute that the server returned. However, if the set of attributes returned includes the last value in the attribute, the server will substitute an asterisk for **high**, indicating to the client that there are no more values to be retrieved.

If a `SearchRequest` does not contain a range option for a given attribute, but that attribute has too many values to be returned at one time, the server returns a `SearchResultEntry` containing (1) the attribute requested without the range option and with no values, and (2) the attribute requested with a range option attached and with the values corresponding to that range option.

The ordering of the values returned in a range retrieval request is arbitrary but consistent across multiple range retrieval requests on the same **LDAP connection**, provided that the attribute is not modified between successive range retrieval requests.

### 3.1.1.3.1.3.3 Ambiguous Name Resolution

Ambiguous name resolution is a search algorithm in Active Directory that permits a client to search multiple naming-related attributes on objects via a single clause in a search filter. A substring search against the [aNR](#) attribute is interpreted by the DC as a substring search against a set of attributes, known as the "ANR attribute set". The intent is that the attributes in the ANR attribute set are those attributes that are commonly used to identify an object, such as the [displayName](#) and [name](#) attributes, thereby permitting a client to query for an object when the client possesses some identifying material related to the object but does not know the attribute of the object that contains that identifying material. The ANR attribute set consists of those attributes whose [searchFlags](#) attribute contains the fANR flag (see section [3.1.1.2.3](#)).

A server performs an ambiguous name resolution search by rewriting a search filter that contains one or more occurrences of the [aNR](#) attribute so the filter no longer contains any occurrences of the [aNR](#) attribute, then performing a regular **LDAP** search using the rewritten search filter. The search filter is rewritten according to the following algorithm.

1. If the ANR attribute set does not contain the attribute [legacyExchangeDN](#), then let S be the ANR attribute set and let PLegacy be false. Otherwise, let S be the ANR attribute set excluding [legacyExchangeDN](#) and let PLegacy be true. In either case, S is a set containing attributes A1...An.
2. Let P1 be the value of the fSupFirstLastANR flag of the [dSHeuristics](#) attribute (see section [7.1.1.2.4.1.2](#)). Let P2 be the value of the fSupLastFirstANR flag of the [dSHeuristics](#) attribute.
3. Let F be the search filter of the search request.
4. For each **LDAP** search filter clause C of the form "(aNR=\*)" in F, resolve the clause to "false". (Such a clause tests for the presence of a value for the [aNR](#) attribute itself, and this attribute has no value on any object.)
5. For each **LDAP** search filter clause C of the form "(aNR=**substringFilter**)", where **substringFilter** is an **LDAP** substring filter of the form "*i*\**f*", in F:
  1. If *i* is the empty string, resolve clause C to the value "Undefined" (see section 4.5.1 of [\[RFC2251\]](#)).
  2. If *i* is non-empty, replace clause C with the clause "(aNR=*i*)" and apply the rule for "(aNR=**value**)" in the next step of this algorithm.
6. For each **LDAP** search filter clause C of the form "(aNR=**value**)" or "(aNR~=**value**)" or "(aNR>=**value**)" or "(aNR<=**value**)" in F:
  1. If **value** does not contain any space characters, or if P1 is true and P2 is true, construct an **LDAP** search filter clause C' of the form "(|(A1=**value**\*)...(An=**value**\*))" if PLegacy is false, or of the form "(|(A1=**value**\*)...(An=**value**\*)(legacyExchangeDN=**value**))" if PLegacy is true. (This clause resolves to "true" for an object if **value** is a prefix of the value of any attribute in the ANR set on that object, except an exact match is always performed on the [legacyExchangeDN](#) attribute.)
  2. If **value** does contain one or more space characters, then:
    1. Split **value** into two components, **value1** and **value2**, at the location of the first space, discarding that space.
    2. If PLegacy is false, do the following:



1. If P1 is false and P2 is false, then construct an **LDAP** search filter clause C' of the form "(|(A1=**value**\*)...(An=**value**\*)&(givenName=**value1**\*) (sn=**value2**\*)&(givenName=**value2**\*)(sn=**value1**\*))". (This clause resolves to "true" for an object if **value** is a prefix of the value of any attribute in the ANR set on that object, or if the two parts of the split **value** are prefixes of the [givenName](#) and [sn](#) attributes on that object, regardless of which part matches which attribute.)
  2. If P1 is true and P2 is false, then construct an **LDAP** search filter clause C' of the form "(|(A1=**value**\*)...(An=**value**\*)&(givenName=**value2**\*) (sn=**value1**\*))". (This clause will resolve to "true" for an object if **value** is a prefix of the value of any attribute in the ANR set on that object, or if the first part of the split **value** is a prefix of the [sn](#) attribute and the second part is a prefix of the [givenName](#) attribute on that object.)
  3. If P1 is false and P2 is true, then construct an **LDAP** search filter clause C' of the form "(|(A1=**value**\*)...(An=**value**\*)&(givenName=**value1**\*) (sn=**value2**\*))". (This clause will resolve to "true" for an object if **value** is a prefix of the value of any attribute in the ANR set on that object, or if the first part of the split **value** is a prefix of the [givenName](#) attribute and the second part is a prefix of the [sn](#) attribute on that object.)
3. If PLegacy is true, do the following:
1. If P1 is false and P2 is false, then construct an **LDAP** search filter clause C' of the form "(|(A1=**value**\*)...(An=**value**\*)(legacyExchangeDN=**value**)&(givenName=**value1**\*) (sn=**value2**\*)&(givenName=**value2**\*)(sn=**value1**\*))". (This clause resolves to "true" for an object if **value** equals the value of [legacyExchangeDN](#) on that object or **value** is a prefix of the value of any attribute in the ANR set on that object, or if the two parts of the split **value** are prefixes of the [givenName](#) and [sn](#) attributes on that object, regardless of which part matches which attribute.)
  2. If P1 is true and P2 is false, then construct an **LDAP** search filter clause C' of the form "(|(A1=**value**\*)...(An=**value**\*)(legacyExchangeDN=**value**) &(givenName=**value2**\*) (sn=**value1**\*))". (This clause will resolve to "true" for an object if **value** equals the value of [legacyExchangeDN](#) on that object or **value** is a prefix of the value of any attribute in the ANR set on that object, or if the first part of the split **value** is a prefix of the [sn](#) attribute and the second part is a prefix of the [givenName](#) attribute on that object.)
  3. If P1 is false and P2 is true, then construct an **LDAP** search filter clause C' of the form "(|(A1=**value**\*)...(An=**value**\*)(legacyExchangeDN=**value**) &(givenName=**value1**\*) (sn=**value2**\*))". (This clause will resolve to "true" for an object if **value** equals the value of [legacyExchangeDN](#) on that object or **value** is a prefix of the value of any other attribute in the ANR set on that object, or if the first part of the split **value** is a prefix of the [givenName](#) attribute and the second part is a prefix of the [sn](#) attribute on that object.)

3. Remove clause C from F, and insert C' into F at the position vacated by C.

Note that the replacement clause C' always contains equality matches, regardless of the type of match in the original clause C.

### 3.1.1.3.1.3.4 Searches Using the objectCategory Attribute

When an **LDAP** search filter F contains a clause C of the form "(objectCategory=**V**)", if **V** is not a DN but there exists an object O such that O![objectClass](#) = [classSchema](#) and O![LDAPDisplayName](#) = **V**, then the server treats the search filter as if clause C was replaced in F with the clause "(objectCategory=**V'**)", where **V'** is O![defaultObjectCategory](#).



### 3.1.1.3.1.3.5 Restrictions on rootDSE Searches

When performing a search against the rootDSE and specifying a list of attributes to be returned, the attributes to be returned must be specified by their **LDAP** display name. Specifying the attribute by their numeric OID will be treated by the server the same as specifying a nonexistent attribute. The server supports specifying the attributes to be returned by their numeric OIDs in searches that do not use the rootDSE as the search base.

When performing a search against the rootDSE, the server will ignore the contents of the search filter, except as noted in section [7.3](#).

### 3.1.1.3.1.4 Referrals in LDAP v2 and v3

When using the **LDAP** v3 protocol, Active Directory returns referrals and continuation references in accord with [\[RFC2251\]](#) section 4.5.3. When using the **LDAP** v2 protocol, Active Directory also returns referrals and continuation references, although these are not part of the **LDAP** v2 protocol, as defined in [\[RFC1777\]](#).

When Active Directory generates a referral in the **LDAP** v2 protocol, it sets the resultCode field in the LDAPResult structure (defined in [\[RFC1777\]](#)) to the value 9. This is a value not defined in [\[RFC1777\]](#) or [\[RFC2251\]](#) but that, by convention, is used by **LDAP** v2 servers to indicate the presence of a referral in the response.

The contents of the referral are conveyed in the errorMessage field of the LDAPResult. This field consists of the string "Referral:", followed by a newline character, followed by one or more LDAPURLs (defined in [\[RFC2255\]](#)). Each LDAPURL is separated by a newline character. The meaning of these LDAPURLs is equivalent to that of an LDAPURL in an **LDAP** v3 referral, that is, they indicate a server or servers against which the operation can be retried.

Active Directory uses the same mechanism to return continuation references in **LDAP** v2. When a continuation reference is required, the DC will return a SearchResponse message (defined in [\[RFC1777\]](#)) in which the resultCode and errorMessage fields in the embedded LDAPResult are set as described above for **LDAP** v2 referrals. As with the **LDAP** v2 referrals, the meaning of the LDAPURLs embedded in the errorMessage field is equivalent to their **LDAP** v3 equivalent; that is, they indicate another server or NC in which the search can be continued.

### 3.1.1.3.1.5 Password Modify Operations

Active Directory provides the ability to change a security principal's password (that is, the Windows password for that security principal) by performing **LDAP** modify operations. The password change is modeled as an **LDAP** modify of either the [unicodePwd](#) or [userPassword](#) attribute of the security principal object. The difference between these two attributes is discussed in the sections that follow. However, regardless of whether the password is modified via [unicodePwd](#) or [userPassword](#), the same attribute on the object is modified. If running as AD/DS, both are treated like a write to the clearTextPassword attribute in [\[MS-SAMR\]](#) section 3.2.1.8.5. If running as AD/LDS, a write to [userPassword](#) updates [unicodePwd](#).

#### 3.1.1.3.1.5.1 unicodePwd

Active Directory stores the password on a [user](#) object or [inetOrgPerson](#) object in the [unicodePwd](#) attribute. This attribute is written by an **LDAP** modify under the following restricted conditions. Windows 2000 servers require that the client have a 128-bit (or better) SSL/TLS-encrypted connection to the DC in order to modify this attribute. On Windows Server 2003 and later, the DC also permits modification of the [unicodePwd](#) attribute on a connection protected by 128-bit (or better) **SASL**-layer encryption instead of SSL/TLS. In Windows Server 2008 and later, if the

AllowPasswordOperationsOverNonSecureConnection flag of the [dSHeuristics](#) attribute (section [7.1.1.2.4.1.2](#)) equals true and Active Directory is operating as AD/LDS, then the DC permits modification of the [unicodePwd](#) attribute over a connection that is neither SSL/TLS-encrypted nor SASL-encrypted. The [unicodePwd](#) attribute is never returned by an **LDAP** search.

When a DC receives an **LDAP** modify request to modify this attribute, it follows the following procedure:

- If the modify request contains a delete operation containing a value **Vdel** for [unicodePwd](#) followed by an add operation containing a value **Vadd** for [unicodePwd](#), the server considers the request to be a request to change the password. The server decodes **Vadd** and **Vdel** using the password decoding procedure documented below. **Vdel** is the old password, while **Vadd** is the new password.
- If the modify request contains a single replace operation containing a value **Vrep** for [unicodePwd](#), the server considers the request to be an administrative reset of the password, that is, a password modification without knowledge of the old password. The server decodes **Vrep** using the same password decoding procedure below and uses it as the new password.

For the password change operation to succeed, the server enforces the requirement that the [user](#) or [inetOrgPerson](#) object whose password is being changed must possess the "User-Change-Password" **control access right** on itself, and that **Vdel** must be the current password on the object. For the password reset to succeed, the server enforces the requirement that the client possess the "User-Force-Change-Password" **control access right** on the [user](#) or [inetOrgPerson](#) object whose password is to be reset.

The syntax of the [unicodePwd](#) attribute is Object(Replica-Link). However, the DC requires that the password value be specified in a **UTF-16** encoded **Unicode** string containing the password surrounded by quotation marks, which has been BER-encoded as an octet string per the Object(Replica-Link) syntax. BER encoding and decoding is defined in [\[ITUX690\]](#). To decode such a value **V**, the server follows this password decoding procedure:

- If **V** is not a valid BER-encoding of an octet string, reject the password operation with the error *protocolError*.
- BER-decode **V** to produce **Vdecoded**.
- If the first and last characters of **Vdecoded** are not the UTF-16 **Unicode** representation of quotation marks, reject the password operation with the error *constraintViolation*.
- Remove the first and last characters from **Vdecoded** to produce **Vpassword**.

**Vpassword** is the value the DC uses for the password; the actual password, not a password hash. This encoding is used for both the old and the new passwords in a password change request.

Here is an example of the first steps of password encoding. Suppose you want to set [unicodePwd](#) to the string "new".

```
ASCII "new":      0x6E 0x65 0x77
UTF-16 "new":     0x6E 0x00 0x65 0x00 0x77 0x00
UTF-16 "new"
    with quotes: 0x22 0x00 0x6E 0x00 0x65 0x00 0x77 0x00 0x22 0x00
```

The 10-byte octet string is then BER-encoded and sent in an **LDAP** modify request as described above.

### 3.1.1.3.1.5.2 userPassword

Active Directory supports modifying passwords on objects via the [userPassword](#) attribute, provided that (1) either the DC is running as AD/LDS, or the DC is running as AD/DS and the **domain functional level** is DS\_BEHAVIOR\_WIN2003 or greater, and (2) fUserPwdSupport is true in the [dSHeuristics](#) attribute (section [7.1.1.2.4.1.2](#)). If fUserPwdSupport is false, the [userPassword](#) attribute is treated as an ordinary attribute and has no special semantics associated with it. If fUserPwdSupport is true but the DC is running as AD/DS and the **domain** functional level is less than DS\_BEHAVIOR\_WIN2003, the DC fails the operation with the error *constraintViolation*.

As with the [unicodePwd](#) attribute, changing a password via the [userPassword](#) attribute is modeled as an **LDAP** Modify operation containing a Delete operation followed by an Add operation, and resetting a password is modeled as an **LDAP** Modify operation containing a single Replace operation. The **control access rights** required are the same as for the [unicodePwd](#) attribute, as is the requirement that when changing a password, **Vdel** must match the object's current password.

The special encoding required for updating the [unicodePwd](#) attribute is not used with the [userPassword](#) attribute, i.e., **Vpassword** = **V**. The same restrictions on SSL/TLS- or SASL-protected connections are enforced. The password values are sent to the server as UTF-8 strings, and surrounding quotation marks are not used. For example, the following LDIF sample changes a password from oldPassword to newPassword.

```
dn: CN=John Smith, OU=Users,DC=Fabrikam,DC=com
changetype: modify
delete: userPassword
userPassword: oldPassword
-
add: userPassword
userPassword: newPassword
-
```

The following example uses LDIF to reset the password to newPassword:

```
dn: CN=John Smith, OU=Users,DC=Fabrikam,DC=com
changetype: modify
replace: userPassword
userPassword: newPassword
-
```

Optionally, when performing a password change operation, the add operation portion of the **LDAP** modify can be omitted. The server treats this as a request to change the [user](#) or [inetOrgPerson](#) object's password to the empty string.

### 3.1.1.3.1.6 Dynamic Objects

The Windows Server 2003 and later versions of Active Directory have support for dynamic objects, as specified in [\[RFC2589\]](#). The Active Directory implementation is conformant to that RFC, except that it does not implement the *dynamicSubtrees* attribute used to represent which NCs support dynamic objects.

Dynamic objects are supported in all NCs except for the schema NC and the config NC. A dynamic object cannot be the parent of an object that is not dynamic, and the server will reject such a

request with the error *unwillingToPerform*. When a dynamic object reaches the end of its time-to-live, the object is Expunged from the directory by the server and does not leave behind a tombstone.

#### 3.1.1.3.1.7 Modify DN Operations

Because Active Directory does not support multivalued RDNs (see section [3.1.1.3.1.2.3](#)), the `deleteoldrdn` field of a `ModifyDNRequest` (defined in [\[RFC2251\]](#) section 4.9) must always be set to true. If `deleteoldrdn` is set to false, the server fails the request with the error *unwillingToPerform*.

#### 3.1.1.3.1.8 Aliases

LDAP aliases, the class for which is defined in section 7.2 of [\[RFC2256\]](#) and which are discussed in [\[RFC2251\]](#) section 4.1.10, are not supported in Active Directory.

#### 3.1.1.3.1.9 Error Message Strings

When the server fails an **LDAP** operation with an error, and the server has sufficient resources to compute a string value for the `errorMessage` field of the `LDAPResult`, it includes a string in the `errorMessage` field of the `LDAPResult` (see [\[RFC2251\]](#) section 4.1.10). The string contains further information about the error.

The first eight characters of the `errorMessage` string is a 32-bit integer, expressed in hexadecimal, containing a Windows Error code (see [\[MS-ERREF\]](#)).

#### 3.1.1.3.1.10 Ports

An AD/DS DC accepts **LDAP** connections on the standard **LDAP** and LDAPS (LDAP over SSL/TLS) ports: 389 and 636. If the AD/DS DC is a GC server, it also accepts **LDAP** connections for GC access on port 3268, and LDAPS connections for GC access on port 3269.

An AD/LDS DC accepts **LDAP** and LDAPS connections on ports that are configured when creating the DC.

#### 3.1.1.3.2 rootDSE Attributes

This section specifies the readable attributes on the `rootDSE` of Windows 2000, Windows Server 2003, and Windows Server 2008 DCs (both AD/DS and AD/LDS).

All of these `rootDSE` attributes are read-only; an **LDAP** request to modify any of them will be rejected with the error *unwillingToPerform*.

The `rootDSE` attributes are not described by the schema, but occurrences of `rootDSE` attribute names are underlined in this document as per the convention for any other **LDAP** attribute.

The following table specifies which of these `rootDSE` attributes are supported by each Windows Server version.

Attribute Name	Windows 2000	Windows Server 2003	Windows Server 2008 AD/DS	Windows Server 2008 AD/LDS
<u><code>configurationNamingContext</code></u>	X	X	X	X
<u><code>currentTime</code></u>	X	X	X	X

Attribute Name	Windows 2000	Windows Server 2003	Windows Server 2008 AD/DS	Windows Server 2008 AD/LDS
<u>defaultNamingContext</u>	X	X	X	X
<a href="#">dNSHostName</a>	X	X	X	X
<u>dsSchemaAttrCount</u>	X	X	X	X
<u>dsSchemaClassCount</u>	X	X	X	X
<u>dsSchemaPrefixCount</u>	X	X	X	X
<u>dsServiceName</u>	X	X	X	X
<u>highestCommittedUSN</u>	X	X	X	X
<u>isGlobalCatalogReady</u>	X	X	X	
<u>isSynchronized</u>	X	X	X	X
<u>ldapServiceName</u>	X	X	X	
<u>namingContexts</u>	X	X	X	X
<u>netlogon</u>	X	X	X	
<u>pendingPropagations</u>	X	X	X	X
<u>rootDomainNamingContext</u>	X	X	X	
<u>schemaNamingContext</u>	X	X	X	X
<a href="#">serverName</a>	X	X	X	X
<a href="#">subSchemaSubEntry</a>	X	X	X	X
<u>supportedCapabilities</u>	X	X	X	X
<u>supportedControl</u>	X	X	X	X
<u>supportedLDAPPolicies</u>	X	X	X	X
<u>supportedLDAPVersion</u>	X	X	X	X
<u>supportedSASLMechanisms</u>	X	X	X	X
<u>domainControllerFunctionality</u>		X	X	X
<u>domainFunctionality</u>		X	X	

Attribute Name	Windows 2000	Windows Server 2003	Windows Server 2008 AD/DS	Windows Server 2008 AD/LDS
<a href="#"><u>forestFunctionality</u></a>		X	X	X
<a href="#"><u>msDS-ReplAllInboundNeighbors</u></a>		X	X	X
<a href="#"><u>msDS-ReplAllOutboundNeighbors</u></a>		X	X	X
<a href="#"><u>msDS-ReplConnectionFailures</u></a>		X	X	X
<a href="#"><u>msDS-ReplLinkFailures</u></a>		X	X	X
<a href="#"><u>msDS-ReplPendingOps</u></a>		X	X	X
<a href="#"><u>msDS-ReplQueueStatistics</u></a>		X	X	X
<a href="#"><u>msDS-TopQuotaUsage</u></a>		X	X	X
<a href="#"><u>supportedConfigurableSettings</u></a>		X	X	X
<a href="#"><u>supportedExtension</u></a>		X	X	X
<a href="#"><u>validFSMOs</u></a>		X	X	X
<a href="#"><u>dsaVersionString</u></a>			X	X
<a href="#"><u>msDS-PortLDAP</u></a>			X	X
<a href="#"><u>msDS-PortSSL</u></a>			X	X
<a href="#"><u>msDS-PrincipalName</u></a>			X	X
<a href="#"><u>serviceAccountInfo</u></a>			X	X
<a href="#"><u>spnRegistrationResult</u></a>			X	X

Attribute Name	Windows 2000	Windows Server 2003	Windows Server 2008 AD/DS	Windows Server 2008 AD/LDS
<a href="#">tokenGroups</a>			X	X
<a href="#">usnAtRfm</a>			X	X

The following table shows, for each rootDSE attribute, whether or not the attribute is operational (that is, whether the server returns the attribute only when it is explicitly requested) and the **LDAP** syntax of the returned value.

Attribute Name	Operational?	LDAP Syntax
<a href="#">configurationNamingContext</a>	N	Object(DS-DN)
<a href="#">currentTime</a>	N	String(Generalized-Time)
<a href="#">defaultNamingContext</a>	N	Object(DS-DN)
<a href="#">dNSHostName</a>	N	String(Unicode)
<a href="#">dsSchemaAttrCount</a>	Y	Integer
<a href="#">dsSchemaClassCount</a>	Y	Integer
<a href="#">dsSchemaPrefixCount</a>	Y	Integer
<a href="#">dsServiceName</a>	N	Object(DS-DN)
<a href="#">highestCommittedUSN</a>	N	LargeInteger
<a href="#">isGlobalCatalogReady</a>	N	Boolean
<a href="#">isSynchronized</a>	N	Boolean
<a href="#">ldapServiceName</a>	N	String(Unicode)
<a href="#">namingContexts</a>	N	Object(DS-DN)
<a href="#">netlogon</a>	Y	String(Octet)
<a href="#">pendingPropagations</a>	Y	Object(DS-DN)
<a href="#">rootDomainNamingContext</a>	N	Object(DS-DN)
<a href="#">schemaNamingContext</a>	N	Object(DS-DN)
<a href="#">serverName</a>	N	Object(DS-DN)
<a href="#">subSchemaSubEntry</a>	N	Object(DS-DN)
<a href="#">supportedCapabilities</a>	N	String(Object-Identifier)
<a href="#">supportedControl</a>	N	String(Object-Identifier)

Attribute Name	Operational?	LDAP Syntax
<a href="#"><u>supportedLDAPPolicies</u></a>	N	String(Unicode)
<a href="#"><u>supportedLDAPVersion</u></a>	N	Integer
<a href="#"><u>supportedSASLMechanisms</u></a>	N	String(Unicode)
<a href="#"><u>domainControllerFunctionality</u></a>	N	Integer
<a href="#"><u>domainFunctionality</u></a>	N	Integer
<a href="#"><u>forestFunctionality</u></a>	N	Integer
<a href="#"><u>msDS-ReplAllInboundNeighbors</u></a>	Y	String(Unicode)*
<a href="#"><u>msDS-ReplAllOutboundNeighbors</u></a>	Y	String(Unicode)*
<a href="#"><u>msDS-ReplConnectionFailures</u></a>	Y	String(Unicode)*
<a href="#"><u>msDS-ReplLinkFailures</u></a>	Y	String(Unicode)*
<a href="#"><u>msDS-ReplPendingOps</u></a>	Y	String(Unicode)*
<a href="#"><u>msDS-ReplQueueStatistics</u></a>	Y	String(Unicode)*
<a href="#"><u>msDS-TopQuotaUsage</u></a>	Y	String(Unicode)**
<a href="#"><u>supportedConfigurableSettings</u></a>	Y	String(Unicode)
<a href="#"><u>supportedExtension</u></a>	Y	String(Object-Identifier)
<a href="#"><u>validFSMOs</u></a>	Y	Object(DS-DN)
<a href="#"><u>dsaVersionString</u></a>	Y	String(Unicode)
<a href="#"><u>msDS-PortLDAP</u></a>	Y	Integer
<a href="#"><u>msDS-PortSSL</u></a>	Y	Integer
<a href="#"><u>msDS-PrincipalName</u></a>	Y	String(Unicode)
<a href="#"><u>serviceAccountInfo</u></a>	Y	String(Unicode)
<a href="#"><u>spnRegistrationResult</u></a>	Y	Integer
<a href="#"><u>tokenGroups</u></a>	Y	String (SID)
<a href="#"><u>usnAtRfm</u></a>	Y	LargeInteger

\* These values contain XML. At the client's request, the server will return the value as binary data in String(Octet) syntax instead.

\*\* This value contains XML.

### 3.1.1.3.2.1 configurationNamingContext

Returns the DN of the root of the config NC on this DC.



### 3.1.1.3.2.2 **currentTime**

Returns the current system time on the DC, as expressed as a string in the Generalized Time format defined by ASN.1 (see [ISO-8601] and [\[ITUX680\]](#), as well as the documentation for the **LDAP** String(Generalized-Time) syntax in [3.1.1.2.2.2](#)).

### 3.1.1.3.2.3 **defaultNamingContext**

Returns the DN of the root of the default NC of this DC. For AD/LDS, the [defaultNamingContext](#) attribute does not exist if a value has not been set for the [msDS-DefaultNamingContext](#) attribute of the DC's [nTDSDSA](#) object.

### 3.1.1.3.2.4 **dnsHostName**

Returns the DNS address of this DC.

### 3.1.1.3.2.5 **dsSchemaAttrCount**

Returns an integer specifying the total number of attributes that are defined in the schema.

### 3.1.1.3.2.6 **dsSchemaClassCount**

Returns an integer specifying the total number of classes that are defined in the schema.

### 3.1.1.3.2.7 **dsSchemaPrefixCount**

Returns the number entries in the DC's prefix table: the field `prefixTable` of the variable `dc` specified in [MS-DRSR] section [5:dc](#).

### 3.1.1.3.2.8 **dsServiceName**

Returns the DN of the [nTDSDSA](#) object for the DC.

### 3.1.1.3.2.9 **highestCommittedUSN**

Returns the USN of this DC. In terms of the state model of section [3.1.1.1](#) this is *dc.usn*.

### 3.1.1.3.2.10 **isGlobalCatalogReady**

Returns a Boolean value indicating if this DC is a global catalog that has completed at least one synchronization of its global catalog data with its replication partners. Returns true if it meets this criteria or false if either the global catalog on this DC has not completed synchronization or this DC does not host a global catalog.

### 3.1.1.3.2.11 **isSynchronized**

Returns a Boolean value indicating if the DC has completed at least one synchronization with its replication partners. Returns either true, if it is synchronized; or false, if it is not.

### 3.1.1.3.2.12 **ldapServiceName**

Returns the **LDAP** service name for the **LDAP** server on the DC. The format of the value is **<DNS name of the forest root domain>:<Kerberos principal name>**, where **Kerberos principal**

**name** is a string representation of the Kerberos principal name for the DC's [computer object](#), as defined in section 2.1.1 of [\[RFC1964\]](#).

#### **3.1.1.3.2.13 namingContexts**

Returns a multi-valued set of DNs. For each NC-replica n hosted on this DC, this attribute contains the DN of the root of n.

#### **3.1.1.3.2.14 netlogon**

LDAP searches that request this rootDSE attribute get resolved as **LDAP** ping operations, as specified in section [7.3](#). Active Directory supports **LDAP** searches for this attribute via both UDP and TCP/IP. See [\[RFC1798\]](#) for details on **LDAP** over UDP.

#### **3.1.1.3.2.15 pendingPropagations**

Returns a set of DNs of objects whose [nTSecurityDescriptor](#) attribute (that is, the object's security descriptor) has been updated but the inheritable portion of the update has not yet been propagated to descendant objects (see Security Descriptor Invariants, section [7.1.3](#)). An object is included in the set only if the update that caused the temporary inconsistency in the object's [nTSecurityDescriptor](#) was performed on the **LDAP** connection that is reading the [pendingPropagations](#) rootDSE attribute.

#### **3.1.1.3.2.16 rootDomainNamingContext**

Returns the DN of the root **domain** NC for this DC's forest.

#### **3.1.1.3.2.17 schemaNamingContext**

Returns the DN of the root of the schema NC on this DC.

#### **3.1.1.3.2.18 serverName**

Returns the DN of the **server object**, contained in the config NC, that represents this DC.

#### **3.1.1.3.2.19 subschemaSubentry**

Returns the DN for the location of the [subSchema](#) object where the classes and attributes in the directory are defined. The [subSchema](#) object pointed to by this attribute contains a read-only copy of the schema described in the format specified in section [3.1.1.3.1.1.1](#)

#### **3.1.1.3.2.20 supportedCapabilities**

Returns a multivalued set of OIDs specifying the capabilities supported by this DC. The definition of each OID is explained in section [3.1.1.3.4.3](#).

#### **3.1.1.3.2.21 supportedControl**

Returns a multivalued set of OIDs specifying the **LDAP** controls supported by this DC. The definition of each OID is explained in section [3.1.1.3.4.1](#)

### 3.1.1.3.2.22 supportedLDAPPolicies

Returns a multivalued set of strings specifying the **LDAP** administrative query policies supported by this DC. The policy strings returned are listed in section [3.1.1.3.4.6](#).

### 3.1.1.3.2.23 supportedLDAPVersion

Returns a set of integers specifying the versions of the **LDAP** protocol supported by this DC. Active Directory supports version 2 and version 3 of the **LDAP** protocol so it returns {2,3} as an **LDAP** multivalue.

### 3.1.1.3.2.24 supportedSASLMechanisms

Returns a multivalued set of strings specifying the security mechanisms supported for SASL negotiation (see [\[RFC2222\]](#), [\[RFC2829\]](#), and [\[RFC2831\]](#)). The definition of each value is explained in section [3.1.1.3.4.5](#).

### 3.1.1.3.2.25 domainControllerFunctionality

Returns an integer indicating the functional level of the DC. This is one of the following values.

Value	Description
0	Windows 2000 Mode
2	Windows Server 2003 Mode
3	Windows Server 2008 Mode

### 3.1.1.3.2.26 domainFunctionality

Returns an integer indicating the functional level of the **domain**. This is one of the following values.

Value	Description
0	Windows 2000 Domain Mode
1	Windows Server 2003 Interim Domain Mode
2	Windows Server 2003 Domain Mode
3	Windows Server 2008 Domain Mode

### 3.1.1.3.2.27 forestFunctionality

Returns a integer indicating the functional level of the forest. This is one of the following values.

Value	Description
0	Windows 2000 Forest Mode
1	Windows Server 2003 Interim Forest Mode
2	Windows Server 2003 Forest Mode

Value	Description
3	Windows Server 2008 Forest Mode

### 3.1.1.3.2.28 msDS-ReplAllInboundNeighbors, msDS-ReplConnectionFailures, msDS-ReplLinkFailures, and msDS-ReplPendingOps

Returns alternate representations of the structures returned by IDL\_DRSGetReplInfo() (see [MS-DRSR] section [4.1.13](#)), either as binary data structures or as XML. The relationship between each of these rootDSE attributes and the IDL\_DRSGetReplInfo data is shown in the table below.

rootDSE Attribute Name	Equivalent DS_REPL_INFO_TYPE	XML Structure	Binary Structure
<a href="#">msDS-ReplAllInboundNeighbors</a>	DS_REPL_INFO_NEIGHBORS	DS_REPL_NEIGHBORW	DS_REPL_NEIGHBORW_BLOB
<a href="#">msDS-ReplConnectionFailures</a>	DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES	DS_REPL_KCC_DSA_FAILUREW	DS_REPL_KCC_DSA_FAILUREW_BLOB
<a href="#">msDS-ReplLinkFailures</a>	DS_REPL_INFO_KCC_DSA_LINK_FAILURES	DS_REPL_KCC_DSA_FAILUREW	DS_REPL_KCC_DSA_FAILUREW_BLOB
<a href="#">msDS-ReplPendingOps</a>	DS_REPL_INFO_PENDING_OPS	DS_REPL_OPW	DS_REPL_OPW_BLOB

For each rootDSE attribute named in the first column, the information returned is exactly the same information that is returned by a call to IDL\_DRSGetReplInfo, specifying the value in the second column as the DRS\_MSG\_GETREPLINFO\_REQ\_V1.InfoType or DRS\_MSG\_GETREPLINFO\_REQ\_V2.InfoType. See [\[MS-DRSR\]](#) for the definition of these, as well as for the definition of the following constants and structures used in the table above:

DS\_REPL\_INFO\_NEIGHBORS

DS\_REPL\_INFO\_KCC\_DSA\_CONNECT\_FAILURES

DS\_REPL\_INFO\_KCC\_DSA\_LINK\_FAILURES

DS\_REPL\_INFO\_PENDING\_OPS

DS\_REPL\_NEIGHBORW

DS\_REPL\_KCC\_DSA\_FAILUREW

DS\_REPL\_KCC\_DSA\_FAILUREW

DS\_REPL\_OPW

The remaining structures in the table above are documented in section [2.2](#).

Without any attribute qualifier, the data is returned as XML. The parent element of the XML is the name of the structure contained in the "XML Structure" column in the table, and the child element names and order in the XML exactly follow the names of the fields in that structure as well. The meaning of each child element is the same as the meaning of the corresponding field in the structure. Values of integer types are represented as decimal strings. Values of FILETIME type are represented as XML dateTime values in UTC, for example, "04-07T18:39:09Z", as defined in [\[XMLSCHEMA2/2\]](#). Values of **GUID** fields are represented as **GUIDStrings**.

If the ";binary" attribute qualifier is specified when the attribute is requested, the value of this attribute is returned as binary data, specifically, the structure contained in the "Binary Structure" column. In this representation, fields that would contain strings are represented as integer offsets (relative to the beginning of the binary data) to a null-terminated UTF-16 encoded string embedded in the returned binary data.

### 3.1.1.3.2.29 msDS-ReplAllOutboundNeighbors

This attribute is equivalent to [msDS-ReplAllInboundNeighbors](#), except that it returns representations of each value of the [repsTo](#) abstract attribute for each NC-replica (for example, outbound replication), while [msDS-ReplAllInboundNeighbors](#) returns representations of each value of the [repsFrom](#) abstract attribute (for example, inbound replication). Like [msDS-ReplAllInboundNeighbors](#), the server will return the data in either XML or binary form, depending on the presence of the ";binary" attribute qualifier, and uses the DS\_REPL\_NEIGHBOR and DS\_REPL\_NEIGHBORW\_BLOB structures for its XML and binary representations, respectively.

### 3.1.1.3.2.30 msDS-ReplQueueStatistics

Reading the [msDS-ReplQueueStatistics](#) attribute returns replication queue statistics.

Like the other ms-dsRepl\* rootDSE attributes, the server returns either XML or binary data, depending on the presence of the ";binary" attribute qualifier. For XML, it returns the following representation:

```
<DS_REPL_QUEUE_STATISTICSW>
<ftimeCurrentOpStarted> ftimeCurrentOpStartedValue </ftimeCurrentOpStarted>
<cNumPendingOps> cNumPendingOpsValue </cNumPendingOps>
<ftimeOldestSync> ftimeOldestSyncValue </ftimeOldestSync>
<ftimeOldestAdd> ftimeOldestAddValue </ftimeOldestAdd>
<ftimeOldestMod> ftimeOldestModValue </ftimeOldestMod>
<ftimeOldestDel> ftimeOldestDelValue </ftimeOldestDel>
<ftimeOldestUpdRefs> ftimeOldestUpdRefsValue </ftimeOldestUpdRefs>
</DS_REPL_QUEUE_STATISTICSW>
```

The structure returned by this attribute for the binary representation is DS\_REPL\_QUEUE\_STATISTICSW\_BLOB (section [2.2.5](#)).

The information returned by reading this attribute is derived from the field replicationQueue of the variable dc specified in [MS-DRSR] section [5:dc](#). dc.replicationQueue is used to serialize IDL\_DRSReplicaSync, IDL\_DRSReplicaAdd, IDL\_DRSReplicaModify, IDL\_DRSReplicaDel, and

IDL\_DRSUpdateRefs request processing ([\[MS-DRSR\]](#)) on the DC. msDS-ReplQueueStatistics returns the following information about the current state of this queue:

- **fTimeCurrentOpStartedValue** is the date and time that the current IDL\_DRSReplicaSync, IDL\_DRSReplicaAdd, IDL\_DRSReplicaModify, IDL\_DRSReplicaDel, or IDL\_DRSUpdateRefs request left the queue and started running.
- **cNumPendingOpsValue** is the number of queued IDL\_DRSReplicaSync, IDL\_DRSReplicaAdd, IDL\_DRSReplicaModify, IDL\_DRSReplicaDel, or IDL\_DRSUpdateRefs requests.
- **fTimeOldestSyncValue** is the date and time that the oldest queued IDL\_DRSReplicaSync request entered the queue.
- **fTimeOldestAddValue** is the date and time that the oldest queued IDL\_DRSReplicaAdd request entered the queue.
- **fTimeOldestModValue** is the date and time that the oldest queued IDL\_DRSReplicaModify request entered the queue.
- **fTimeOldestDelValue** is the date and time that the oldest queued IDL\_DRSReplicaDel request entered the queue.
- **fTimeOldestUpdRefsValue** is the date and time that the oldest queued IDL\_DRSUpdateRefs request entered the queue.

**cNumPendingOpsValue** is an integer represented as a decimal string. The remaining values are represented as XML dateTime values in UTC, defined in [\[XMLSCHEMA2/2\]](#).

If a designated request does not exist, the corresponding portion of the [msDS-ReplQueueStatistics](#) response contains a zero filetime in the binary format, and the XML dateTime value "1601-01-01T00:00:00Z" in XML format. For instance, if there is no IDL\_DRSUpdateRefs request in the replication queue, the msDS-ReplQueueStatistics XML response includes:

```
<fTimeOldestUpdRefs>1601-01-01T00:00:00Z</fTimeOldestUpdRefs>
```

### 3.1.1.3.2.31 msDS-TopQuotaUsage

Returns a multivalued set of strings specifying the top 10 quota users in all NC-replicas on this DC. The format of each value is as follows, where quota usage is measured in number of objects:

```
<MS_DS_TOP_QUOTA_USAGE>
<partitionDN> DN of NC-replica </partitionDN>
<ownerSID>Security Identifier (SID) of quota user </ownerSID>
<quotaUsed> Amount of quota used by this quota user </quotaUsed>
<tombstoneCount> Number of tombstoned objects owned by this quota user
</tombstoneCount>
<liveCount> Number of live (non-deleted) objects owned by this quota user </liveCount >
</MS_DS_TOP_QUOTA_USAGE>
```

A client qualifies the attribute description for this attribute in an **LDAP** query with a "range qualifier" to specify a different range of quota users to return other than the top 10. The DC responds to this

by returning the quota usage for the requested range of quota users. Below are examples of range qualifiers and what would be returned.

- An attribute specification of the form *msDS-TopQuotaUsage;Range=0-\** will return the complete list of quota usage
- An attribute specification of the form *msDS-TopQuotaUsage;Range=1-9* will return the 2<sup>nd</sup> highest through the 10<sup>th</sup> highest quota usage
- An attribute specification of the form *msDS-TopQuotaUsage;Range=2-2* will return the 3<sup>rd</sup> highest quota usage

The caller must have the `RIGHT_DS_READ_PROPERTY` access right on the Quotas container (see section [7.1.1.4.3](#)). If the caller does not have this access right, the search operation will succeed but no results will be returned.

#### **3.1.1.3.2.32 supportedConfigurableSettings**

Returns a multivalued set of strings specifying the configurable settings supported by this DC. The setting strings returned are listed in section [3.1.1.3.4.7](#).

#### **3.1.1.3.2.33 supportedExtension**

Returns a multivalued set of OIDs specifying the extended **LDAP** operations that the DC supports. The definition of each OID is explained in section [3.1.1.3.4.2](#).

#### **3.1.1.3.2.34 validFSMOs**

Returns a set of DNs of objects representing the FSMO roles owned by this DC. Each object identifies a distinct FSMO role.

The valid types of FSMO role, and the object used to represent an instance of that type in the validFSMOs attribute, are as follows:

- Schema Master FSMO Role - the root of the schema NC
- Partition Naming FSMO Role - the Partitions container in the config NC
- Infrastructure Master FSMO Role - the Infrastructure container in a **domain** NC
- **Primary Domain Controller (PDC)** Emulator FSMO Role -the root of a **domain** NC
- RID Master FSMO Role -the RID Manager object of a **domain** NC, which is the object referenced by the [rIDManagerReference](#) attribute on the root of the **domain** NC

Because an AD/LDS forest does not contain **domain** NCs, it does not contain instances of the Infrastructure Master, Primary Domain Controller Emulator, and RID Master FSMO roles, and the corresponding objects will not be present in the validFSMOs attribute of any DC running AD/LDS.

A server indicates that it owns a given FSMO Role F only if `IsEffectiveRoleOwner(RoleObject(nc, e))` returns true, where the procedures `IsEffectiveRoleOwner` and `RoleObject` are defined in section [3.1.1.5.1.7](#). The parameters **nc** and **e** are defined as follows for each FSMO Role F:

- Schema Master FSMO
  - **nc**: Schema NC

- **e:** SchemaMasterRole
- Partition Naming FSMO
  - **nc:** Config NC
  - **e:** PartitionNamingMasterRole
- Infrastructure Master FSMO
  - **nc:** Default NC (AD/DS)
  - **e:** InfrastructureMasterRole
- RID Master FSMO
  - **nc:** Default NC (AD/DS)
  - **e:** RidAllocationMasterRole
- PDC Emulator FSMO
  - **nc:** Default NC (AD/DS)
  - **e:** PdcEmulationMasterRole

#### 3.1.1.3.2.35 dsaVersionString

Returns a string indicating the version of Active Directory running on the DC. For instance, when running Windows Server 2008 Beta 2, the Active Directory version string is "6.0.5384.32 (winmain\_beta2.060727-1500)".

This rootDSE attribute is readable by Domain Administrators (section [7.1.1.6.5](#)) and Enterprise Administrators (section [7.1.1.6.10](#)) only.

#### 3.1.1.3.2.36 msDS-PortLDAP

Returns the integer TCP/UDP port number on which the DC is listening for **LDAP** requests. For AD/DS, this always equals 389. For AD/LDS, the port is configurable.

#### 3.1.1.3.2.37 msDS-PortSSL

Returns the integer TCP/UDP port number on which the DC is listening for TLS/SSL-protected **LDAP** requests. For AD/DS, this always equals 636. For AD/LDS, the port is configurable.

#### 3.1.1.3.2.38 msDS-PrincipalName

Returns a string name of the security principal that has authenticated on the **LDAP** connection. If the client authenticated as a Windows security principal, the string contains either (1) the **NetBIOS domain** name, followed by a backslash ('\'), followed by the [sAMAccountName](#) of the security principal, or (2) the SID of the security principal, in SDDL SID string format ([\[MS-DTYP\]](#)). If the client authenticated as an AD/LDS security principal, the string contains the DN of the security principal. If the connection is not authenticated (only possible if `fLDAPBlockAnonOps` in [dSHeuristics](#) is false, see section [7.1.1.2.4.1.2](#)), the string is "NT AUTHORITY\ANONYMOUS LOGON".



### 3.1.1.3.2.39 serviceAccountInfo

Returns a set of strings, each string containing a name-value pair encoded as **name=value**.

The [serviceAccountInfo](#) attribute contains information outside the state model. The possible name-value pairs are as follows:

**replAuthenticationMode:** The value is the value of the [msDS-ReplAuthenticationMode](#) attribute on the root of the config NC, or "1" if that attribute is not set. See section [7.1.1.1.2](#) for the effects of the [msDS-ReplAuthenticationMode](#) attribute.

**accountType:** If the service account is a **domain** account, the value is "**domain**". Otherwise the service account is a local account, and the value is "**local**".

**systemAccount:** If the service account is a system account (meaning it has one of the SIDs SID "S-1-5-20" and "S-1-5-18") the value is "**true**"; otherwise the value is "**false**".

**domainType:** If the DC is running on a computer that is part of an ActiveDirectory **domain** (always the case for an AD/DS DC), the value is "**domainWithKerb**". If the DC is running on a computer that is part of an NT (pre Active Directory) **domain**, the value is "**domainNoKerb**". Otherwise the DC is running on a computer that is not part of a **domain**, and the value is "**nonMember**".

**serviceAccountName:** If the value of **replAuthenticationMode** is "0", the value is the SAM name of the DC's service account. Otherwise this name-value pair is not present.

**machineDomainName:** If **domainType** is "**domainWithKerb**" or "**domainNoKerb**" the value is the NetBIOS name of the **domain**. Otherwise the value is the NetBIOS name of the computer.

### 3.1.1.3.2.40 spnRegistrationResult

When running as AD/DS, this value is 0. When running as AD/LDS, if the DC was unable to register its SPNs ([MS-DRSR] section [2.2.2](#)), this attribute returns the Windows error code associated with the failure. Otherwise, it returns 0.

### 3.1.1.3.2.41 tokenGroups

Returns the SIDs contained in the **security context** as which the client has authenticated the **LDAP** connection. See section [5.1.3](#).

### 3.1.1.3.2.42 usnAtRifm

This attribute contains information outside the state model. If the DC is an RODC and was installed using the Install From Media feature, reading the [usnAtRifm](#) attribute returns the value of dc.usn (section [3.1.1.1.9](#)) that was present in the Active Directory database on the installation media.

## 3.1.1.3.3 rootDSE Modify Operations

This section specifies the modifiable attributes on the rootDSE of Windows 2000, Windows Server 2003, and Windows Server 2008 DCs (both AD/DS and AD/LDS).

rootDSE modify operations are used to trigger behaviors on a specific DC. For example, one such operation causes the DC to acquire the Schema Master FSMO. All of these rootDSE attributes are write-only; an **LDAP** request to read will be treated as if the attribute does not exist.

The following table specifies the set of modifiable rootDSE attributes included in each Windows version.

<b>Attribute Name</b>	<b>Windows 2000</b>	<b>Windows 2000 SP1</b>	<b>Windows Server 2003</b>	<b>Windows Server 2003 SP3</b>	<b>Windows Server 2008 AD/DS</b>	<b>Windows Server 2008 AD/LDS</b>
<u>becomeDomainMaster</u>	X	X	X	X	X	X
<u>becomeInfrastructureMaster</u>	X	X	X	X	X	
<u>becomePdc</u>	X	X	X	X	X	
<u>becomePdcWithCheckPoint</u>	X	X	X	X	X	
<u>becomeRidMaster</u>	X	X	X	X	X	
<u>becomeSchemaMaster</u>	X	X	X	X	X	X
<u>checkPhantoms</u>	X	X	X	X	X	
<u>doGarbageCollection</u>	X	X	X	X	X	X
<u>dumpDatabase</u>	X	X	X	X	X	X
<u>fixupInheritance</u>	X	X	X	X	X	X
<u>invalidateRidPool</u>	X	X	X	X	X	
<u>recalcHierarchy</u>	X	X	X	X	X	
<u>schemaUpdateNow</u>	X	X	X	X	X	X
<u>removeLingeringObject</u>		X	X	X	X	X
<u>doLinkCleanup</u>			X	X	X	X
<u>doOnlineDefrag</u>			X	X	X	X
<u>replicateSingleObject</u>			X	X	X	X
<u>updateCachedMemberships</u>			X	X	X	

Attribute Name	Windows 2 000	Windows 2 000 SP1	Windows Server 2 003	Windows Server 2 003 SP3	Windows Server 2 008 AD/DS	Windows Server 2 008 AD/LDS
<u>doGarbageCollection</u> <u>PhantomsNow</u>				X	X	X
<u>invalidateGCCConnecti</u> <u>on</u>					X	X
<u>renewServerCertificat</u> <u>e</u>					X	X
<u>rODCPurgeAccount</u>					X	
<u>runSamUpgradeTasks</u>					X	
<u>sgmRunOnce</u>						X

Each of these operations is executed by performing an **LDAP** modify operation with a NULL DN for the object to be modified (indicating the rootDSE) and specifying the name of the modify operation as the attribute to be modified. In many of the cases, the type of the modify (add values, replace values, delete values) and the values specified do not matter and are ignored. Whether the type and values matter, and what the client specifies if they do matter, will be indicated for each operation below. Examples are given as LDAP Data Interchange Format (LDIF) samples, described in [RFC2849]. In Windows, LDIF is implemented by the `ldifde.exe` command-line tool.

To perform many of these operations, the caller must be authenticated as a user that has a particular **control access right** or, in some cases, as a user that is a member of a particular group. In each section below, the rights or group membership, if any, that are required of the caller to perform a specific operation are specified.

#### 3.1.1.3.3.1 becomeDomainMaster

Performing this operation causes the DC to request a transfer of the Partition Naming FSMO to itself, per the **FSMO role transfer** procedure documented in [MS-DRSR] section [4.1.10.4.3](#) (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP\_FSMO\_REQ\_ROLE). The requestor must have the "Change-Domain-Master" **control access right** on the Partitions container in the config NC for this to succeed. This operation cannot be performed on an RODC; an RODC will return **LDAP** error *unwillingToPerform*.

The type of modification and values specified in the **LDAP** modify operation do not matter. A LDIF sample that performs this operation is shown below:

```
dn:
changetype: modify
add: becomeDomainMaster
becomeDomainMaster: 1
-
```

### 3.1.1.3.3.2 becomeInfrastructureMaster

Performing this operation causes the DC to request a transfer of the Infrastructure Master FSMO to itself, per the FSMO role transfer procedure documented in [\[MS-DRSR\]](#) section 4.1.10.4.3 (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP\_FSMO\_REQ\_ROLE). The requestor must have the "Change-Infrastructure-Master" **control access right** on the Infrastructure container in the **domain** NC-replica. This operation cannot be performed on an RODC; an RODC will return **LDAP** error *unwillingToPerform*.

The type of modification and values specified in the **LDAP** modify operation do not matter. A LDIF sample that performs this operation is shown below:

```
dn:  
changetype: modify  
add: becomeInfrastructureMaster  
becomeInfrastructureMaster: 1  
-
```

### 3.1.1.3.3.3 becomePdc

Performing this operation causes the DC to request a transfer of the Primary domain controller (PDC) Emulator FSMO to itself, per the FSMO role transfer procedure documented in [\[MS-DRSR\]](#) section 4.1.10.4.3 (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP\_FSMO\_REQ\_PDC). The requestor must have the "Change-PDC" **control access right** on the root of the **domain** NC-replica. This operation cannot be performed on an RODC; an RODC will return **LDAP** error *unwillingToPerform*.

Prior to transferring the PDC FSMO to the DC, if the **domain** is in **mixed mode**, the DC attempts to synchronize with the DC that is currently the Owner of the PDC FSMO in such a way as to avoid causing a full synchronization by BDCs running Windows NT 4.0 (see section [3.1.1.7](#)). However, the FSMO role transfer will be performed even if this synchronization is unsuccessful.

In order to perform this operation, the requestor must provide the **domain's** Security ID (SID), in binary format (defined in [\[MS-DTYP\]](#) section 2.4.2), as the value of the modify operation. In LDIF, this would be performed as shown below. Note that LDIF requires that binary values be base-64 encoded.

```
dn:  
changetype: modify  
add: becomePdc  
becomePdc:: base-64 encoding of the domain SID in binary  
-
```

### 3.1.1.3.3.4 becomePdcWithCheckPoint

This operation is the same as [becomePdc](#) except for the following. If the **domain** is in **mixed mode** and the attempt to synchronize the DC with the current Owner of the PDC FSMO fails, the PDC FSMO Role is not transferred to the DC.

### 3.1.1.3.3.5 becomeRidMaster

Performing this operation causes the DC to request a transfer of the RID Master FSMO to itself, per the FSMO role transfer procedure documented in [\[MS-DRSR\]](#) section [4.1.10.4.3](#)

(PerformExtendedOpRequestMsg, ulExtendedOp = EXOP\_FSMO\_RID\_REQ\_ROLE). The requestor must have the "Change-RID-Master" **control access right** on the RID Manager object, which is the object referenced by the [rIDManagerReference](#) attribute located on the root of the **domain** NC. The requestor must also have **read permission** on the aforementioned [rIDManagerReference](#) attribute. This operation cannot be performed on an RODC; an RODC returns **LDAP** error *unwillingToPerform*.

The type of modification and values specified in the **LDAP** modify operation do not matter. A LDIF sample that performs this operation is shown below:

```
dn:  
changetype: modify  
add: becomeRidMaster  
becomeRidMaster: 1  
-
```

#### 3.1.1.3.3.6 becomeSchemaMaster

Performing this operation causes the DC to request a transfer of the Schema Master FSMO to itself, per the FSMO role transfer procedure documented in [MS-DRSR] section [4.1.10.4.3](#) (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP\_FSMO\_REQ\_ROLE). The requestor must have the "Change-Schema-Master" **control access right** on the root of the schema NC-replica. This operation cannot be performed on an RODC; an RODC will return **LDAP** error *unwillingToPerform*.

The type of modification and values specified in the **LDAP** modify operation do not matter. A LDIF sample that performs this operation is shown below:

```
dn:  
changetype: modify  
add: becomeSchemaMaster  
becomeSchemaMaster: 1  
-
```

#### 3.1.1.3.3.7 checkPhantoms

Performing this operation on the DC that owns the Infrastructure Master FSMO causes the DC to immediately run the reference update task (see section [3.1.1.6.2](#)). The requestor must have the "DS-Check-Stale-Phantoms" **control access right** on the [nTDSDSA](#) object for the DC. No action is taken if this operation is performed against a DC that does not own the Infrastructure Master FSMO.

The type of modification and values specified in the **LDAP** modify operation do not matter.

An LDIF sample that performs this operation is shown below:

```
dn:  
changetype: modify  
add: checkPhantoms  
checkPhantoms: 1  
-
```

### 3.1.1.3.3.8 doGarbageCollection

This operation requests that **garbage collection** be immediately performed on the DC. Tombstones are subject to the invariant that they must be kept for at least the tombstone lifetime (see [3.1.1.6.2](#)) but they may be kept longer. Garbage collection identifies tombstones which have been kept for at least the tombstone lifetime and removes them. On a correctly-functioning DC, there should not be a need to manually trigger garbage collection via this operation. The requestor must have the "Do-Garbage-Collection" **control access right** on the DC's **DSA object**.

This operation is triggered by setting the [doGarbageCollection](#) attribute to "1".

A LDIF sample that performs this operation is shown below:

```
dn:
changetype: modify
add: doGarbageCollection
doGarbageCollection: 1
-
```

### 3.1.1.3.3.9 dumpDatabase

This operation is triggered by setting the attribute to a space-separated list of attributes. The requestor must be a member of the BUILTIN\Administrators group (section [7.1.1.4.11.2](#)).

An LDIF sample that performs this operation for the [description](#) and [sn](#) attributes is shown below:

```
dn:
changetype: modifyadd: dumpDatabase
dumpDatabase: description sn
-
```

The effects of [dumpDatabase](#) are outside the state model. An update of [dumpDatabase](#) causes the contents of the DC's database to be written to a text file on the DC's disk. All the attributes specified in the [dumpDatabase](#) value are included in the dump, except that certain security-sensitive attributes are omitted from the dump even if requested. The dump may include attributes that were not explicitly requested.

### 3.1.1.3.3.10 fixupInheritance

The [fixupInheritance](#) attribute permits administrative tools to request that the DC recompute inherited security permissions on objects to ensure they conform to the security descriptor invariants (see section [7.1.3](#)), in case the current state of the permissions on the object is erroneous. This operation is not necessary on a correctly functioning DC. The requestor must have the "Recalculate-Security-Inheritance" **control access right** on the [nTDSDSA](#) object for the DC.

This operation is triggered by setting the [fixupInheritance](#) attribute to "1".

An LDIF sample that performs this operation is shown below:

```
dn:
changetype: modify
add: fixupInheritance
fixupInheritance: 1
```

-

In Windows Server 2003 and later setting the [fixupInheritance](#) attribute to the special values "forceupdate" and "downgrade" has effects outside the state model. Setting [fixupInheritance](#) to the value "forceupdate" enables storing values of the [nTSecurityDescriptor](#) attribute in a single-instance table, and setting [fixupInheritance](#) to the value "downgrade" disables the use of the single-instance table. For example:

```
dn:
changetype: modify
add: fixupInheritance
fixupInheritance: forceupdate
-
```

In Windows Server 2003 and later the [fixupInheritance](#) attribute can trigger security descriptor propagation under an object, specified using an identifier outside the state model, rather than throughout the directory. This is performed by setting the [fixupInheritance](#) attribute to the string "dnt:" followed by a 32-bit integer, expressed in decimal, that identifies the database row representing the object. For example:

```
dn:
changetype: modify
add: fixupInheritance
fixupInheritance: dnt:54758
-
```

### 3.1.1.3.3.11 invalidateRidPool

This operation causes the DC to discard its current pool of **relative identifiers (RIDs)**, used for allocating **security principals** in the directory. The DC requests a fresh pool of RIDs from the DC that owns the RID Master FSMO, per the procedure documented in [MS-DRSR] section [4.1.10.4.3](#) (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP\_FSMO\_REQ\_RID\_ALLOC).

The requestor must have the "Change-RID-Master" **control access right** on the RID Manager object, which is the object referenced by the [rIDManagerReference](#) attribute located on the root of the **domain** NC. The requestor must also have read permission on the aforementioned [rIDManagerReference](#) attribute. This operation cannot be performed on an RODC; an RODC returns **LDAP** error *unwillingToPerform*.

In order to perform this operation, the requestor provides the **domain's** Security ID (SID), in binary format (defined in [\[MS-DTYP\]](#) section **2.4.2**), as the value of the modify operation.

An LDIF sample that performs this operation is shown below. LDIF requires that binary values, like the **domain** SID, be base-64 encoded.

```
dn:
changetype: modify
add: invalidateRidPool
invalidateRidPool:: base-64 encoding of the binary-format domain SID
-
```

### 3.1.1.3.3.12 recalcHierarchy

The requestor must have the "Recalculate-Hierarchy" **control access right** on the [nTDSDSA](#) object for the DC. The type of modification and values specified in the **LDAP** Modify operation do not matter. A LDIF sample that performs this operation is shown below:

```
dn:  
changetype: modify  
add: recalcHierarchy  
recalcHierarchy: 1  
-
```

The effects of [recalcHierarchy](#) are outside the state model. An update of [recalcHierarchy](#) causes the hierarchy table used to support the MAPI address book to be recalculated immediately.

### 3.1.1.3.3.13 schemaUpdateNow

The requestor must have the "Update-Schema-Cache" **control access right** on the [nTDSDSA](#) object for the DC or on the root of the schema NC. After the completion of this operation, the [subSchema](#) (section [3.1.1.3.1.1.1](#)) exposed by the server reflects the current state of the schema as defined by the [attributeSchema](#) and [classSchema](#) objects in the schema NC. Requesting this operation on an RODC results in the **LDAP** error *unwillingToPerform*.

The type of modification and values specified in the **LDAP** modify operation do not matter. A LDIF sample that performs this operation is shown below:

```
dn:  
changetype: modify  
add: schemaUpdateNow  
schemaUpdateNow: 1  
-
```

The other effects of [schemaUpdateNow](#) are outside the state model. An update of [schemaUpdateNow](#) causes the in-memory cache of the schema to be recalculated from the copy of the schema stored in the schema NC.

### 3.1.1.3.3.14 removeLingeringObject

This operation causes the DC to expunge a lingering object. A DC that was offline for longer than the value of the tombstone lifetime can contain objects that have been deleted on other DCs and for which tombstones no longer exist. The result is that when that DC is brought back online, any such objects can continue to exist in its NC-replica even though the objects should have been deleted. Such objects are known as *lingering objects*.

Expunge was specified in section [3.1.1.1.6](#). Lingering object expunge can be performed on an object in a read-only NC. For more details on the lingering object expunge process, see IDL\_DRSReplicaVerifyObjects and IDL\_DRSGetObjectExistence in [MS-DRSR] sections [4.1.24](#) and [4.1.12](#).

The requestor must have the "DS-Replication-Synchronize" **control access right** on the root of the NC-replica which contains the lingering object, otherwise the result is *insufficientAccessRights*.



The value specified for this operation contains (1) the DN of the DSA object of a DC holding a writable replica of the NC containing the lingering object, and (2) the DN of the lingering object. These are encoded in the value string as two DNs separated by a colon: "**DSA Object DN:Lingering Object DN**". Each DN specified may be either a RFC 2253-style DN, or one of the alternative DN formats described in section [3.1.1.3.1.2.4](#). If the value is not in the specified format, the server rejects the request with the error *operationsError* ([\[RFC2251\]](#) section 4.1.10).

The DC performing the modify request first verifies that the lingering object specified in the request does not exist on the DC specified in the request. If this verification fails for any reason, the request returns the error *operationsError* ([\[RFC2251\]](#)). If the verification succeeds, the DC expunges the lingering object specified in the request.

An LDIF sample that performs this operation is shown below. The sample requests that the lingering object whose DN is "CN=TestObject, CN=Users, DC=Fabrikam, DC=com" be removed, and specifies that the server whose [nTDSDSA](#) object is "CN=NTDS Settings,CN=TESTDC-01,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=Fabrikam,DC=com" be used to verify the non-existence of the lingering object.

```
dn:
changetype: modify
replace: removeLingeringObject
removeLingeringObject: CN=NTDS Settings,
CN=TESTDC-01,CN=Servers,CN=Default-First-Site-Name,
CN=Sites,CN=Configuration,DC=Fabrikam,DC=com:CN=TestObject,
CN=Users, DC=Fabrikam, DC=com
-
```

### 3.1.1.3.3.15 doLinkCleanup

This operation causes the reference update task (section [3.1.1.6.2](#)) to run immediately. This task runs periodically; on a correctly functioning DC, there is no need to run it explicitly. The requestor must have the "Do-Garbage-Collection" **control access right** on the [nTDSDSA](#) object for the DC.

The type of modification and values specified in the **LDAP** modify operation do not matter. An LDIF sample that performs this operation is shown below:

```
dn:
changetype: modify
add: doLinkCleanup
doLinkCleanup: 1
-
```

### 3.1.1.3.3.16 doOnlineDefrag

This operation is triggered by setting the [doOnlineDefrag](#) attribute to a non-negative integer. The requestor must have the "Do-Garbage-Collection" **control access right** on the [nTDSDSA](#) object for the DC. An LDIF sample that performs this operation is shown below:

```
dn:
changetype: modify
replace: doOnlineDefrag
doOnlineDefrag: 60
-
```

The effects of [doOnlineDefrag](#) are outside the state model. An update of [doOnlineDefrag](#) causes an online de-fragmentation of the DC's directory database. If the [doOnlineDefrag](#) value is positive, it starts the de-fragmentation task, which runs until complete or until the specified number of seconds have elapsed. If the [doOnlineDefrag](#) value is zero, the de-fragmentation task is stopped if it is running.

### 3.1.1.3.3.17 replicateSingleObject

This operation causes the DC to request replication of a single object, specified in the modify request, from a source DC to the DC processing the request. The requestor must have the "DS-Replication-Synchronize" **control access right** on the root of the NC that contains the object to be replicated.

The value specified for the [replicateSingleObject](#) attribute in the modify request contains (1) the DN of the DSA object of the source DC, and (2) the DN of the object to be replicated. These are encoded in the value string as two DNs separated by a colon: "**DSA Object DN:Object To Be Replicated DN**". Each DN specified may be either a RFC 2253-style DN, or one of the alternative DN formats described in section [3.1.1.3.1.2.4](#). If the value is not in the specified format, the server rejects the request with the error *operationsError* (as specified in [\[RFC2251\]](#) section 4.1.10).

If the DC is an RODC, an additional colon may be added to the end of the value string, followed by the literal string "SECRETS\_ONLY". The presence of this additional parameter indicates that the RODC should request replication of the object's secret attributes instead of the other attributes. When this flag is specified, the "DS-Replication-Synchronize" **control access right** is not checked. Instead, the requestor must possess the "Read-Only-Replication-Secret-Synchronization" **control access right** on the root of the NC containing the object whose secrets are to be replicated.

An LDIF sample that performs the [replicateSingleObject](#) operation is shown below. This sample requests that the object whose DN is "CN=TestObject, CN=Users, DC=Fabrikam, DC=com" be replicated from the DC whose [nTDSDSA](#) object is "CN=NTDS Settings,CN=TESTDC-01,CN=Servers,CN=Default-First-Site-Name,CN= Sites,CN=Configuration,DC=Fabrikam,DC=com".

```
dn:
changetype: modify
replace: replicateSingleObject
replicateSingleObject: CN=NTDS Settings,
CN=TESTDC-01,CN=Servers,CN=Default-First-Site-Name,
CN= Sites,CN=Configuration,DC=Fabrikam,DC=com:CN=TestObject,
CN=Users, DC=Fabrikam, DC=com
-
```

### 3.1.1.3.3.18 updateCachedMemberships

The type of modification and values specified in the **LDAP** modify operation do not matter. The requestor must have the "Refresh-Group-Cache" **control access right** on the [nTDSDSA](#) object for the DC.

An LDIF sample that performs this operation is shown below:

```
dn:
changetype: modify
add: updateCachedMemberships
updateCachedMemberships: 1
-
```

The effects of [updateCachedMemberships](#) are outside the state model. An update of [updateCachedMemberships](#) causes the DC to refresh its cache of **universal group** memberships from a GC server.

#### 3.1.1.3.3.19 doGarbageCollectionPhantomsNow

This operation is triggered by setting the [doGarbageCollectionPhantomsNow](#) attribute to "1". The requestor must have the "Do-Garbage-Collection" **control access right** on the [nTDSDSA](#) object for the DC.

An LDIF sample that performs this operation is shown below:

```
dn:
changetype: modify
add: doGarbageCollectionPhantomsNow
doGarbageCollectionPhantomsNow: 1
-
```

The effects of [doGarbageCollectionPhantomsNow](#) are outside the state model. An update of [doGarbageCollectionPhantomsNow](#) causes a garbage-collector to run that reclaims storage used to implement referential integrity.

#### 3.1.1.3.3.20 invalidateGCCConnection

The type of modification to the [invalidateGCCConnection](#) attribute and the values specified in the **LDAP** Modify operation do not matter. The requestor must be a member of either the BUILTIN\Administrators group (section [7.1.1.4.11.2](#)) or the BUILTIN\Server Operators group (section [7.1.1.4.11.18](#)).

A LDIF sample that performs this operation is shown below:

```
dn:
changetype: modify
add: invalidateGCCConnection
invalidateGCCConnection: 1
-
```

The effects of [invalidateGCCConnection](#) are outside the state model. This operation causes the DC to rediscover the GC server that it uses in its implementation of referential integrity (section [3.1.1.1.6](#)).

#### 3.1.1.3.3.21 renewServerCertificate

The persistent state of a DC does not include the certificates that are necessary to authenticate the DC when a client makes an LDAPS (LDAP over SSL/TLS) connection. A DC obtains the certificates it needs by querying the operating system for them at startup. This operation provides a means for the requestor to request that the DC repeat the query to the operating system for the certificates, for example, if the available certificates have changed since startup. The requestor must have the "Reload-SSL-Certificate" **control access right** on the [nTDSDSA](#) object for the DC.

An **LDAP** Modify of the [renewServerCertificate](#) attribute causes the DC to query the operating system for certificates. When the operation returns, the DC has performed the query and the certificates it found are available for use in LDAPS connections.

The type of modification and values specified in the **LDAP** modify operation do not matter.

An LDIF sample that performs this operation is shown below:

```
dn:  
changetype: modify  
add: renewServerCertificate  
renewServerCertificate: 1  
-
```

### 3.1.1.3.3.22 rODCPurgeAccount

An **LDAP** Modify of the rODCPurgeAccount attribute causes the RODC to purge cached secrets of a specified **security principal**. The requestor must have the "Read-Only-Replication-Secret-Synchronization" **control access right** on the root of the default NC, otherwise the **LDAP** modify returns *insufficientAccessRights*. The Modify request must be directed to a RODC which hosts a NC-replica that contains the specified RODC object. If the RODC to which the operation is directed does not host such an NC, then the error *operationsError / ERROR* *ERROR\_DS\_CANT\_FIND\_EXPECTED\_NC* is returned. If the operation is sent to a DC that is not an RODC, then the error *operationsError / ERROR\_DS\_GENERIC\_ERROR* is returned.

The value specified for the rODCPurgeAccount attribute in the **LDAP** modify request must be the DN of the RODC object whose secrets are to be purged. The DN specified may be either a RFC 2253-style DN, or one of the alternative DN formats described in section 3.1.1.3.1.2.4. If the value is not in the specified format or the object does not exist, the server rejects the request with the error *operationsError / ERROR\_DS\_OBJ\_NOT\_FOUND*.

An LDIF sample that performs this operation is shown below. This sample purges the cached secrets of the user whose DN is "CN=TestUser, CN=Users, DC=Fabrikam, DC=com" from the RODC to which this operation is sent.

```
dn:  
changetype: modify  
replace: rODCPurgeAccount  
rODCPurgeAccount: CN=TestUser, CN=Users, DC=Fabrikam, DC=com  
-
```

### 3.1.1.3.3.23 runSamUpgradeTasks

An LDAP Modify of the runSamUpgradeTasks attribute causes the default groups and memberships (as specified in [MS-SAMR] section [3.2.4.2](#)) to be created in the domain if they are not already created. This operation is useful in a domain with different versions of domain controllers where the default groups and memberships are not yet created.

If a partial set of these modifications has already been performed in the domain through this task, the Modify operation of this attribute **MUST** cause the rest of the operations to be performed. If all such modifications have already been performed, the Modify operation of this attribute **MUST NOT** make any changes in the domain.

The requestor **MUST** be a member of the "Domain Admins" group in the domain to perform this operation; otherwise, the LDAP Modify returns *insufficientAccessRights*.

The DC, on receiving this request, **MUST** verify that the [otherWellKnownObjects](#) attribute on the object "CN=Server, CN=System, DC=<domain>" on the DC with PDC role contains "B:32:

6ACDD74F3F314AE396F62BBE6B2DB961:X", where <domain> is the domain NC DN and X is the DN of the nTDSDSA object of the DC receiving the request. If this condition is not satisfied, the LDAP Modify returns *operationsError* / *ERROR\_DS\_GENERIC\_ERROR*.

If these conditions are satisfied, the default groups and memberships (as specified in [MS-SAMR] section 3.2.4.2) are created in the domain.

The type of modification and values specified in the LDAP Modify operation do not matter. An LDIF sample that performs this operation is shown below. This sample triggers the default groups and memberships created on the target domain.

```
dn:  
changetype: modify  
add: runSamUpgradeTasks  
runSamUpgradeTasks: 1  
-
```

#### 3.1.1.3.3.24 sqmRunOnce

The type of modification and values specified in the **LDAP** modify operation do not matter. The requestor must have the SE\_DEBUG\_PRIVILEGE.

An LDIF sample that performs this operation is shown below:

```
dn:  
changetype: modify  
add: sqmRunOnce  
sqmRunOnce: 1  
-
```

The effects of sqmRunOnce are outside the state model. An update of sqmRunOnce causes the DC to report statistical data on the types and numbers of operations that the DC has performed using an implementation-defined reporting mechanism.

#### 3.1.1.3.4 LDAP Extensions

This section describes the extensions to **LDAP** supported by Microsoft Active Directory DCs in Windows 2000, Windows Server 2003, and Windows Server 2008. These extensions are:

- LDAP extended controls
- LDAP extended operations
- LDAP capabilities
- Matching rules
- SASL mechanisms
- Policies
- Configurable settings
- IP Deny list

### 3.1.1.3.4.1 LDAP Extended Controls

LDAP extended controls are an extensibility mechanism in version 3 of the **LDAP** protocol, as discussed in [RFC2251](#) section 4.1.12. The following sections describe the **LDAP** extended controls implemented by DCs in Windows 2000, Windows Server 2003, and Windows Server 2008 (both AD/DS and AD/LDS).

The **LDAP** extended controls supported by a DC are exposed as OIDs in the supportedControl attribute of the rootDSE. Each OID corresponds to a human-readable name, as shown in the following table.

Extended control name	OID
LDAP_PAGED_RESULT_OID_STRING	1.2.840.113556.1.4.319
LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID	1.2.840.113556.1.4.521
LDAP_SERVER_DIRSYNC_OID	1.2.840.113556.1.4.841
LDAP_SERVER_DOMAIN_SCOPE_OID	1.2.840.113556.1.4.1339
LDAP_SERVER_EXTENDED_DN_OID	1.2.840.113556.1.4.529
LDAP_SERVER_GET_STATS_OID	1.2.840.113556.1.4.970
LDAP_SERVER_LAZY_COMMIT_OID	1.2.840.113556.1.4.619
LDAP_SERVER_PERMISSIVE_MODIFY_OID	1.2.840.113556.1.4.1413
LDAP_SERVER_NOTIFICATION_OID	1.2.840.113556.1.4.528
LDAP_SERVER_RESP_SORT_OID	1.2.840.113556.1.4.474
LDAP_SERVER_SD_FLAGS_OID	1.2.840.113556.1.4.801
LDAP_SERVER_SEARCH_OPTIONS_OID	1.2.840.113556.1.4.1340
LDAP_SERVER_SORT_OID	1.2.840.113556.1.4.473
LDAP_SERVER_SHOW_DELETED_OID	1.2.840.113556.1.4.417
LDAP_SERVER_TREE_DELETE_OID	1.2.840.113556.1.4.805
LDAP_SERVER_VERIFY_NAME_OID	1.2.840.113556.1.4.1338
LDAP_CONTROL_VLVREQUEST	2.16.840.1.113730.3.4.9
LDAP_CONTROL_VLVRESPONSE	2.16.840.1.113730.3.4.10
LDAP_SERVER_ASQ_OID	1.2.840.113556.1.4.1504
LDAP_SERVER_QUOTA_CONTROL_OID	1.2.840.113556.1.4.1852
LDAP_SERVER_RANGE_OPTION_OID	1.2.840.113556.1.4.802
LDAP_SERVER_SHUTDOWN_NOTIFY_OID	1.2.840.113556.1.4.1907
LDAP_SERVER_FORCE_UPDATE_OID	1.2.840.113556.1.4.1948

Extended control name	OID
LDAP_SERVER_RANGE_RETRIEVAL_NOERR_OID	1.2.840.113556.1.4.1974
LDAP_SERVER_RODC_DCPROMO_OID	1.2.840.113556.1.4.1341
LDAP_SERVER_INPUT_DN_OID	1.2.840.113556.1.4.2026

The following table lists the set of **LDAP** extended controls supported in each Windows Server version. Any extended control supported in a given Windows Server version is supported by later versions.

Extended control name	Windows 2000	Windows Server 2003	Windows Server 2003 SP1	Windows Server 2008
LDAP_PAGED_RESULT_OID_STRING	X	X	X	X
LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID	X	X	X	X
LDAP_SERVER_DIRSYNC_OID	X	X	X	X
LDAP_SERVER_DOMAIN_SCOPE_OID	X	X	X	X
LDAP_SERVER_EXTENDED_DN_OID	X	X	X	X
LDAP_SERVER_GET_STATS_OID	X	X	X	X
LDAP_SERVER_LAZY_COMMIT_OID	X	X	X	X
LDAP_SERVER_PERMISSIVE_MODIFY_OID	X	X	X	X
LDAP_SERVER_NOTIFICATION_OID	X	X	X	X
LDAP_SERVER_RANGE_OPTION_OID*	X	X	X	X
LDAP_SERVER_RESP_SORT_OID	X	X	X	X
LDAP_SERVER_SD_FLAGS_OID	X	X	X	X
LDAP_SERVER_SEARCH_OPTIONS_OID	X	X	X	X
LDAP_SERVER_SORT_OID	X	X	X	X
LDAP_SERVER_SHOW_DELETED_OID	X	X	X	X
LDAP_SERVER_TREE_DELETE_OID	X	X	X	X
LDAP_SERVER_VERIFY_NAME_OID	X	X	X	X
LDAP_CONTROL_VLVREQUEST		X	X	X
LDAP_CONTROL_VLVRESPONSE		X	X	X

Extended control name	Windows 2000	Windows Server 2003	Windows Server 2003 SP1	Windows Server 2008
LDAP_SERVER_ASQ_OID		X	X	X
LDAP_SERVER_QUOTA_CONTROL_OID		X	X	X
LDAP_SERVER_SHUTDOWN_NOTIFY_OID**			X	X
LDAP_SERVER_FORCE_UPDATE_OID				X
LDAP_SERVER_RANGE_RETRIEVAL_NOERR_OID				X
LDAP_SERVER_RODC_DCPROMO_OID				X
LDAP_SERVER_INPUT_DN_OID				X

\* This OID does not identify an **LDAP** extended control. Its presence in the supportedControl attribute indicates that the DC is capable of range retrieval (see section [3.1.1.3.1.3.2](#)) of **LDAP** multivalued attributes. However, its absence does not indicate lack of support for range retrieval. This OID is not present in the supportedControl attribute of Windows 2000 DCs, but those DCs do support range retrieval.

\*\* Although exposed on the supportedControl attribute of Windows Server 2003 SP1 and later DCs, this control is only functional on DCs running the Small Business Server version of that operating system.

A client sends a control to the DC by attaching a Control structure (defined in [\[RFC2251\]](#) section 4.1.12) to an **LDAP** operation. The client sets the controlType field to the control's OID and the controlValue field as specified in the discussion for the control below. If the controlValue field contains data that is not in conformance with the specification of the control, including the case where the controlValue field contains data and the specification of the control states that the controlValue field is omitted, then if the control is marked critical the server returns the error *unavailableCriticalExtension* ([\[RFC2251\]](#) section 4.1.10). If the controlValue field is incorrect but the control is not marked critical, the server ignores the control.

A control sent by the client to a DC is known as a request control. In some cases, the server includes a corresponding Control structure attached to the response for the **LDAP** operation. These controls, known as response controls, are discussed below in conjunction with the request control that causes that response control to be returned.

A brief description of each **LDAP** control is given in the following table. Additionally, each control is discussed in more detail in the sections that follow. References to ASN.1 and BER encoding in the following section are references to [\[ITUX680\]](#) and [\[ITUX690\]](#), respectively.



Extended control name	Description
LDAP_PAGED_RESULT_OID_STRING	Splits the results of an <b>LDAP</b> search across multiple result sets.
LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID	Used with an <b>LDAP</b> ModifyDN operation to move an object from one <b>domain</b> to another <b>domain</b> .
LDAP_SERVER_DIRSYNC_OID	Used with an <b>LDAP</b> search operation to retrieve the changes made to objects since a previous LDAP_SERVER_DIRSYNC_OID search was performed.
LDAP_SERVER_DOMAIN_SCOPE_OID	Instructs the DC not to generate <b>LDAP</b> continuation references in response to a search operation.
LDAP_SERVER_EXTENDED_DN_OID	Used to request that an <b>LDAP</b> search operation return DN's in an extended format containing the values of the <a href="#">objectGUID</a> and <a href="#">objectSid</a> attributes.
LDAP_SERVER_GET_STATS_OID	Used with an <b>LDAP</b> search request to instruct the DC to return statistical data related to how the search was performed.
LDAP_SERVER_LAZY_COMMIT_OID	Instructs the DC that it MAY sacrifice durability guarantees on updates to improve performance.
LDAP_SERVER_PERMISSIVE_MODIFY_OID	Instructs the DC that an <b>LDAP</b> modify should succeed even if it attempts to add a value already present on the attribute or remove a value not present on the attribute.
LDAP_SERVER_NOTIFICATION_OID	Used with an <b>LDAP</b> search operation to register the client to be notified when changes are made to an object in the directory.
LDAP_SERVER_SD_FLAGS_OID	Instructs the DC which portions of a Windows Security Descriptor to retrieve during an <b>LDAP</b> search operation.
LDAP_SERVER_SEARCH_OPTIONS_OID	Used to pass flags to the DC to control search behaviors, specifically, to prevent <b>LDAP</b> continuation references from being generated and to search all NC-replicas that are subordinate to the search base, even if the search base is not instantiated on the DC.
LDAP_SERVER_SORT_OID and LDAP_SERVER_RESP_SORT_OID	Request and response controls, respectively, for instructing the DC to sort the search results.
LDAP_SERVER_SHOW_DELETED_OID	Used with an <b>LDAP</b> search operation to specify that the search results are to include any deleted objects that match the search filter.
LDAP_SERVER_TREE_DELETE_OID	Used with an <b>LDAP</b> delete operation to cause the server to recursively delete the entire subtree of objects located under the object specified in the search request (including the specified object).
LDAP_SERVER_VERIFY_NAME_OID	Permits the client to specify which GC the DC should use when processing an add or modify request to verify the existence of any objects pointed to by DN

Extended control name	Description
	attribute values.
LDAP_CONTROL_VLVREQUEST and LDAP_CONTROL_VLVRESPONSE	Request and response control, respectively, used with an <b>LDAP</b> search operation to retrieve a "sliding window" subset of the objects that satisfy the search requests.
LDAP_SERVER_ASQ_OID	Used to specify that an <b>LDAP</b> search operation should not be performed against the object specified as the base in the search, but rather against the set of objects named by a specified attribute of Object(DS-DN) syntax on the base object.
LDAP_SERVER_QUOTA_CONTROL_OID	Used with an <b>LDAP</b> search operation to retrieve the quota of a user.
LDAP_SERVER_RANGE_OPTION_OID	Indicates that the server is capable of range retrieval (see section <a href="#">3.1.1.3.1.3.2</a> ).
LDAP_SERVER_SHUTDOWN_NOTIFY_OID	Used with an <b>LDAP</b> search operation to cause the client to be notified when the DC is shutting down.
LDAP_SERVER_FORCE_UPDATE_OID	When attached to an <b>LDAP</b> update operation, causes the DC to perform the update even if that update would not affect the state of the DC.
LDAP_SERVER_RANGE_RETRIEVAL_NOERR_OID	Instructs the DC that, when performing a search using range retrieval (see section <a href="#">3.1.1.3.1.3.2</a> ) on an attribute whose values are forward link values or back link values and the value of low is greater than or equal to the number of values in the attribute, no error should be returned.
LDAP_SERVER_RODC_DCPROMO_OID	This control is used as part of the process of promoting a computer to be a RODC.
LDAP_SERVER_INPUT_DN_OID	This control is used to specify the DN of an object during an <b>LDAP</b> operation. Currently this control is used only while retrieving the <b>constructed attribute</b> <a href="#">msDS-IsUserCachableAtRodec</a> (see section <a href="#">3.1.1.3.4.1.24</a> ).

#### 3.1.1.3.4.1.1 LDAP\_PAGED\_RESULT\_OID\_STRING

This control, which is used as both a request control and a response control, is documented in [\[RFC2696\]](#).

DCs limit the number of objects that can be returned in a single search operation to the value specified by the MaxPageSize policy defined in section [3.1.1.3.4.6](#). The use of the LDAP\_PAGED\_RESULT\_OID\_STRING control permits clients to perform searches that return more objects than this limit by splitting the search into multiple searches, each of which returns no more objects than this limit.

#### 3.1.1.3.4.1.2 LDAP\_SERVER\_CROSSDOM\_MOVE\_TARGET\_OID

The LDAP\_SERVER\_CROSSDOM\_MOVE\_TARGET\_OID control is used with an **LDAP** ModifyDN operation to instruct the DC to move an object from one **domain** to another (see ModifyDN operation in section 3.1.1.5). When moving an object across **domains**, this control must be used by the client. The client sends the **LDAP** ModifyDN operation to which this control is attached to a DC in the **domain** containing the object to be moved.

When operating as AD/LDS, a DC rejects this control with error *operationsError* ([RFC2251] section 4.1.10).

When sending this control to the DC, the controlValue field is set to a UTF-8 string containing the fully qualified DNS name of a DC in the **domain** to which the object is to be moved. The string is not be BER encoded. Sending this control to the DC does not cause the server to include any controls in its response.

#### 3.1.1.3.4.1.3 LDAP\_SERVER\_DIRSYNC\_OID

The LDAP\_SERVER\_DIRSYNC\_OID control is used with an **LDAP** search operation to retrieve the changes made to objects since a previous search with an LDAP\_SERVER\_DIRSYNC\_OID control was performed. The LDAP\_SERVER\_DIRSYNC\_OID control can only be used to monitor for changes across an entire NC-replica, not a subtree within a NC-replica.

The server that the LDAP\_SERVER\_DIRSYNC\_OID control is executed on has two security modes that are differentiated by whether the client specifies the LDAP\_DIRSYNC\_OBJECT\_SECURITY flag.

If this flag is specified, the client can only view objects and attributes that are otherwise accessible to the client.

If the flag is not specified, the server performs an access check per the algorithm described in [MS-DRSR] section 5: **IsGetNCChangesPermissionGranted**, using as inputs the attributes requested in the search request. The algorithm is modified slightly in that, if the client requests any secret attributes, the request is ignored by the server and such attributes are not returned. If **IsGetNCChangesPermissionGranted** returns FALSE, the server returns the *insufficientAccessRights* error to the client.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    Flags                INTEGER
    maxAttributeCount    INTEGER
    Cookie               OCTET STRING
}
```

The **Flags** value has the following format presented in big-endian byte order. X denotes unused bits set to 0 by the client and ignored by the server.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	1	2	3	4	5	6	7	8	9	30	1
I V	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	P	X	A	X	X	X	X	X	X	X	X	X	X	O
																		D		F										S	
																		O		O											

The **Flags** value is a combination of zero or more bit flags from the following table, and is used to specify additional behaviors for the LDAP\_SERVER\_DIRSYNC\_OID control.

Bit flag name and value	Description
LDAP_SERVER_DIRSYNC_OBJECT_SECURITY (OS) 0x00000001	Windows Server 2003 and later: If this flag is present, the client can only view objects and attributes that are otherwise accessible to the client. If this flag is not present, the server must perform access checks as described earlier in this section. Windows 2000: Not supported.
LDAP_SERVER_DIRSYNC_ANCESTORS_FIRST_ORDER (AFO) 0x00000800	The server returns parent objects before child objects.
LDAP_SERVER_DIRSYNC_PUBLIC_DATA_ONLY (PDO) 0x00002000	Windows Server 2003 and later: This flag can optionally be passed to the DC, but it has no effect. Windows 2000: Not supported.
LDAP_SERVER_DIRSYNC_INCREMENTAL_VALUES (IV) 0x80000000	Windows Server 2003 and later: If this flag is not present, all of the values, up to a server-specified limit, in a multi-valued attribute are returned when any value changes. If this flag is present, only the changed values are returned, provided the attribute is a <b>forward link value</b> . Windows 2000: Not supported.

**maxAttributeCount** specifies the maximum number of attributes to return. **Cookie** is an opaque value that was returned by the DC on a previous search request that included the LDAP\_SERVER\_DIRSYNC\_OID control. The contents of **Cookie** are freely defined by the server and cannot be interpreted by the client. A search request with the LDAP\_SERVER\_DIRSYNC\_OID control attached will return the changes made to objects since the point in time when the previous search request which returned the value of **Cookie** being used in the current search request took place. If there was no previous LDAP\_SERVER\_DIRSYNC\_OID search request, **Cookie** is NULL, in which case the search will return all objects which satisfy the search request along with a value of **Cookie** to use for the next LDAP\_SERVER\_DIRSYNC\_OID search request.

The search request to which the LDAP\_SERVER\_DIRSYNC\_OID control is attached must satisfy the following criteria:

- The base of the search must be the root of an NC.
- The search scope must be subtree scope.
- Any valid **LDAP** search filter can be specified.

- Any attributes can be requested in the search. Only those objects for which these attributes have been created or modified since the time represented by **Cookie** will be considered for inclusion in the search.

If the base of the search is not the root of an NC, the server will return the error *unwillingToPerform* ([RFC2251] section 4.1.10). If the search scope is not subtree scope, the server will treat the search as if subtree scope was specified.

If **Cookie** is NULL, all objects in the NC-replica are considered for inclusion in the search. If **Cookie** is non-NULL, only those objects modified since that **Cookie** was returned are considered for inclusion in the search.

When the server receives a search request with the LDAP\_SERVER\_DIRSYNC\_OID control attached to it, it includes a response control in the search response. The controlType field of the returned Control structure is set to the OID of the LDAP\_SERVER\_DIRSYNC\_OID control, and the controlValue is the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    MoreResults      INTEGER
    unused           INTEGER
    CookieServer     OCTET STRING
}
```

The structure of the controlValue in the response control is the same as the structure of the controlValue in the request control, but the fields are interpreted differently. **MoreResults** is non-zero if there are more changes to retrieve, **unused** is not used, and **CookieServer** is the value to be used for **Cookie** in the next LDAP\_SERVER\_DIRSYNC\_OID control sent in a search request to the server.

#### 3.1.1.3.4.1.4 LDAP\_SERVER\_DOMAIN\_SCOPE\_OID

The LDAP\_SERVER\_DOMAIN\_SCOPE\_OID control is used to instruct the DC not to generate any **LDAP** continuation references when performing an **LDAP** operation. The effect of this is to limit any search using it to the single NC-replica in which the object that serves as the root of the search is located.

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

#### 3.1.1.3.4.1.5 LDAP\_SERVER\_EXTENDED\_DN\_OID

The LDAP\_SERVER\_EXTENDED\_DN\_OID control is used with an **LDAP** search request to cause the DC to return extended DNs. The extended form of an object's DN includes a string representation of the object's [objectGUID](#) attribute; for objects that have an [objectSid](#) attribute, the extended form also includes a string representation of that attribute. The DC uses this extended DN for all DNs in the **LDAP** search response.

The extended DN format is as follows:

<GUID=**guid\_value**>;<SID=**sid\_value**>;dn

where **guid\_value** is the value of the object's [objectGUID](#) attribute, **sid\_value** is the value of the object's [objectSid](#) attribute, and **dn** is the object's RFC 2253 DN. For objects that do not have an [objectSid](#) attribute, the format is instead as follows:

<GUID=**guid\_value**>;dn

When sending this control to a Windows 2000 DC, the controlValue field is omitted. When sending this control to a Windows Server 2003 or later DC, the controlValue field is either omitted, or is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {  
    Flag    INTEGER  
}
```

where **Flag** must be set to 0 or 1. A value of 0, which is equivalent to omitting the controlValue field, causes the DC to return the values of the [objectGUID](#) and [objectSid](#) attributes as a hexadecimal representation of their binary format. A value of 1 causes the DC to return the **GUID** in dashed-string format ([\[RFC4122\]](#) section 3) and to return the SID in SDDL SID string format ([\[MS-DTYP\]](#)). The returned SDDL SID string begins with "S-".

For example, setting **Flag** to 0 (or omitting the controlValue field) might return the following extended DN:

<GUID=b3d4bfb3c45ee4298e27b4a698a61b8>;<SID=01050000000000051500000061eb5b8c50ef705befda808bf4010000>;CN=Administrator, CN=Users,DC=Fabrikam,DC=com

while setting **Flag** to 1 would return the same object's extended DN in the following form:

<GUID=bdbfd4b3-453c-42ee-98e2-7b4a698a61b8>;<SID=S-1-5-21-2354834273-1534127952-2340477679-500>;CN=Administrator, CN=Users,DC=Fabrikam,DC=com

Sending this control to the DC does not cause the server to include any controls in its response.

#### 3.1.1.3.4.1.6 LDAP\_SERVER\_GET\_STATS\_OID

The LDAP\_SERVER\_GET\_STATS\_OID control is used with an **LDAP** Search operation. The client must have the SE\_DEBUG\_PRIVILEGE in order to specify this control.

When sending this control to a DC running Windows 2000, the client omits the controlValue field. When sending this control to a DC running Windows Server 2003 or later, the client either omits the controlValue field or sets the controlValue field to one of the 32-bit unsigned integer values in the following table. The values are not BER encoded.

Value Name	Value	Description
SO_NORMAL	0	Perform the search as if no LDAP_SERVER_GET_STATS_OID control were included in the search request.
SO_STATS	1	Perform the search and return data related to the resources consumed performing the search well as the actual search results.
SO_ONLY_OPTIMIZE	2	Return data to how the search would be performed, but do not actually return the search results.
SO_EXTENDED_FMT	4	Windows Server 2008: Returns the data in an alternative format documented below. Windows 2000 and Windows Server 2003: Not supported.

Omitting the controlValue field is equivalent to specifying the SO\_STATS value.

When the server receives a search request with the LDAP\_SERVER\_GET\_STATS\_OID control attached to it, it includes a response control in the search response. The controlType field of the returned Control structure is set to the OID of the LDAP\_SERVER\_GET\_STATS\_OID control. The controlValue field is included in the returned Control structure.

The response to this control contains information outside the state model. This control instructs the server to return internal data related to how the **LDAP** search was performed.

For Windows 2000 DCs, the returned controlValue is the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    threadCountTag      INTEGER
    threadCount          INTEGER
    coreTimeTag          INTEGER
    coreTime              INTEGER
    callTimeTag          INTEGER
    callTime              INTEGER
    searchSubOperationsTag INTEGER
    searchSubOperations  INTEGER
}
```

where **threadCountTag**, **coreTimeTag**, **callTimeTag**, and **searchSubOperationsTag** are equal to 1, 2, 3, and 4, respectively. **threadCount** is the number of threads that were processing **LDAP** requests on the DC at the time the search operation was performed, **coreTime** is the time in milliseconds which core logic in the DC spent processing the request, **callTime** is the overall time in milliseconds that the DC spent processing the request, and **searchSubOperations** is the number of individual operations which the DC performed in processing the request.

For Windows Server 2003 DCs, the controlValue of the response control is the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    threadCountTag      INTEGER
    threadCount          INTEGER
    callTimeTag          INTEGER
    callTime              INTEGER
    entriesReturnedTag  INTEGER
    entriesReturned      INTEGER
    entriesVisitedTag    INTEGER
    entriesVisited        INTEGER
    filterTag            INTEGER
    filter                OCTET STRING
    indexTag             INTEGER
    index                OCTET STRING
}
```

In this structure, **threadCountTag**, **threadCount**, **callTimeTag**, and **callTime** are as in the Windows 2000 structure. **entriesReturnedTag**, **entriesVisitedTag**, **filterTag**, and **indexTag** are 5, 6, 7, and 8, respectively. **entriesReturned** is the number of objects returned in the search result. **entriesVisited** is the number of objects that the DC considered for inclusion in the search result. **filter** is a UTF-8 string which represents the optimized form of the search filter used by the DC to perform a search. **index** is a ANSI string which indicates which database indexes were used by the DC to perform the search.

For Windows Server 2008 DCs, the controlValue of the response control is the BER encoding of the following ASN.1 structure if the SO\_EXTENDED\_FMT flag is not specified:

```
SEQUENCE {
    threadCountTag      INTEGER
    threadCount          INTEGER
    callTimeTag          INTEGER
    callTime             INTEGER
    entriesReturnedTag  INTEGER
    entriesReturned      INTEGER
    entriesVisitedTag   INTEGER
    entriesVisited       INTEGER
    filterTag            INTEGER
    filter               OCTET STRING
    indexTag             INTEGER
    index                OCTET STRING
    pagesReferencedTag  INTEGER
    pagesReferenced      INTEGER
    pagesReadTag         INTEGER
    pagesRead            INTEGER
    pagesPrereadTag     INTEGER
    pagesPreread         INTEGER
    pagesDirtiedTag     INTEGER
    pagesDirtied         INTEGER
    pagesRedirtiedTag   INTEGER
    pagesRedirtied       INTEGER
    logRecordCountTag   INTEGER
    logRecordCount       INTEGER
    logRecordBytesTag   INTEGER
    logRecordBytes       INTEGER
}
```

In this structure, fields with the same name as fields in the Windows Server 2003 structure are as in the Windows Server 2003 structure. **pagesReferencedTag**, **pagesReadTag**, **pagesPrereadTag**, **pagesDirtiedTag**, **pagesRedirtiedTag**, **logRecordCountTag**, and **logRecordCountBytesTag** are 9, 10, 11, 12, 13, 14, and 15, respectively. **pagesReferenced** is the number of database pages referenced by the DC in processing the search. **pagesRead** is the number of database pages read from disk and **pagesPreread** is the number of database pages preread from disk by the DC in processing the search. **pagesDirtied** is the number of clean database pages modified by the DC in processing the search, while **pagesRedirtied** is the number of previously modified database pages that were modified by the DC in processing the search. **logRecordCount** and **logRecordBytes** are the number and size in bytes, respectively, of database log records generated by the DC in processing the search.

For Windows Server 2008 DCs, if the SO\_EXTENDED\_FMT flag is specified, an alternative format is used for the controlValue of the response control instead of the format shown above. Unlike the previous formats in which each statistic is assigned a fixed position within the structure, in the alternative format the ordering of the statistics can change. Rather than relying on position, each statistic has an associated human-readable string that specifies what that statistic is. Additionally, the use of these associated strings alleviates the need to hard-code the positional information into the client-side parser of the response control, permitting the DC to be updated to return addition statistics without necessitating a corresponding client-side change.

When using the alternative format, the controlValue of the response control is the BER encoding of the following ASN.1 structure:



```

SEQUENCE OF SEQUENCE {
    statisticName      OCTET STRING
    CHOICE {
        [0] intStatistic      INTEGER
        [1] stringStatistic   OCTET STRING
    }
}

```

Effectively, this is an array of statistics, in which each statistic has a human-readable name (the **statisticName** field) and a value. If it is an integer-valued statistic, the value is stored in the **intStatistic** field. If it is a string-valued statistic, the value is stored in the **stringStatistic** field.

When the SO\_EXTENDED\_FMT flag is specified, Windows Server 2008 DCs return the same statistics as if the flag was not specified. The only difference is the format used to return the statistics. The wording of the **statisticName** field is implementation-defined. Currently, the wording as it maps to each statistic as specified in the non-SO\_EXTENDED\_FMT version of the structure is as follows:

threadCount	"Thread count"
callTime	"Call time (in ms)"
entriesReturned	"Entries Returned"
entriesVisited	"Entries Visited"
filter	"Used Filter"
index	"Used Indexes"
pagesReferenced	"Pages Referenced"
pagesRead	"Pages Read From Disk"
pagesPreread	"Pages Pre-read From Disk"
pagesDirtied	"Clean Pages Modified"
pagesRedirtied	"Dirty Pages Modified"
logRecordCount	"Log Records Generated"
logRecordBytes	"Log Record Bytes Generated"

#### 3.1.1.3.4.1.7 LDAP\_SERVER\_LAZY\_COMMIT\_OID

The LDAP\_SERVER\_LAZY\_COMMIT\_OID control is used to modify the behavior of any **LDAP** operation. The presence of this control instructs the DC that it may sacrifice durability guarantees on updates to improve performance.

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

#### 3.1.1.3.4.1.8 LDAP\_SERVER\_PERMISSIVE\_MODIFY\_OID

The LDAP\_SERVER\_PERMISSIVE\_MODIFY\_OID control is used to modify the behavior of an **LDAP** modify operation. An **LDAP** modify operation normally returns an error if it attempts to add an attribute that already exists on an object to that object (or, in the case of multi-valued attributes, it attempts to add a value that is already present in the attribute). An **LDAP** modify operation will also normally fail if it attempts to delete an attribute that does not exist on the specified object or that does not contain the value specified in the deletion request. With this control, adding a value to an attribute that already exists and already contains the value to be added will cause the server to return *success* even though no modification was actually performed by the server. Similarly, deletion of an attribute that does not exist or does not contain the specified value will return *success*.

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

#### 3.1.1.3.4.1.9 LDAP\_SERVER\_NOTIFICATION\_OID

The LDAP\_SERVER\_NOTIFICATION\_OID control is used with an **LDAP** search operation to register the client to be notified when changes are made to an object in the directory.

Notifications are asynchronous operations. When the DC receives a search request with this control attached, it does not immediately send a response to the request. Instead, when an object is modified, if that object falls within the scope of the search request to which the LDAP\_SERVER\_NOTIFICATION\_OID control was attached, the DC sends a SearchEntry response that contains the modified object to the client using the messageID from the original search request (SearchEntry and messageID are defined in [\[RFC2251\]](#) section 4.1.1). The SearchEntry response will contain those attributes of the object that were requested in the original request. These attributes are not necessarily the attributes that were modified. A client indicates that it no longer wants notifications by sending an **LDAP** abandon operation, specifying the messageID of the original search request.

LDAP search requests that include this control are subject to the following restrictions:

- The only filter permitted in the search request is "(objectclass = \*)". The server will return the error *unwillingToPerform* ([\[RFC2251\]](#) section 4.1.10) if this is not the case.
- Base, one-level, and subtree search scopes are permitted. For Windows 2000 DCs, the base DN specified in a subtree search must be the root of a NC, and the server will return the error *unwillingToPerform* if this is not the case. Windows Server 2003 and later DCs do not have this restriction.

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its eventual responses.

#### 3.1.1.3.4.1.10 LDAP\_SERVER\_RANGE\_OPTION\_OID

LDAP\_SERVER\_RANGE\_OPTION\_OID, unlike the other controls discussed in this section, does not actually designate an **LDAP** extended control. Nonetheless, it is included in this discussion because its OID is found in the *supportedControl* attribute of the DC's rootDSE. The presence of this OID indicates that the DC supports range retrieval of multivalued attributes. Range retrieval is a mechanism that permits attributes that have too many values to be retrieved in a single **LDAP** search request to be retrieved via multiple **LDAP** search requests. Range retrieval is documented in section [3.1.1.3.1.3.2](#).

Note that although this OID is not present in the *supportedControl* attribute of Windows 2000 DCs, such DCs nonetheless support range retrieval.

#### 3.1.1.3.4.1.11 LDAP\_SERVER\_SD\_FLAGS\_OID

The LDAP\_SERVER\_SD\_FLAGS\_OID control is used with an **LDAP** Search request to control the portion of a Windows Security Descriptor to retrieve. The DC returns only the specified portion of the security descriptors. It is also used with **LDAP** Add and Modify requests to control the portion of a Windows Security Descriptor to modify. The DC modifies only the specified portion of the security descriptor.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {  
    Flags      INTEGER
```

}

The **Flags** value has the following format presented in big-endian byte order. X denotes unused bits that must be set to 0 by the client and that must be ignored by the server.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	S	D	G	O
																													S	S	S	S
																													I	I	I	I

The **Flags** value is a combination of zero or more bit flags from the following table:

Bit flag name and value	Portion of security descriptor to retrieve/update
OWNER_SECURITY_INFORMATION (OSI) 0x1	Owner identifier of the object.
GROUP_SECURITY_INFORMATION (GSI) 0x2	<b>Primary group</b> identifier.
DACL_SECURITY_INFORMATION (DSI) 0x4	<b>Discretionary ACL</b> of the object.
SACL_SECURITY_INFORMATION (SSI) 0x8	System <b>ACL</b> of the object.

Specifying **Flags** with no bits set, or not using the LDAP\_SERVER\_SD\_FLAGS\_OID control, is equivalent to setting **Flags** to (OWNER\_SECURITY\_INFORMATION | GROUP\_SECURITY\_INFORMATION | DACL\_SECURITY\_INFORMATION | SACL\_SECURITY\_INFORMATION). Sending this control to the DC does not cause the server to include any controls in its response.

### 3.1.1.3.4.1.12 LDAP\_SERVER\_SEARCH\_OPTIONS\_OID

The LDAP\_SERVER\_SEARCH\_OPTIONS\_OID control is used with an **LDAP** Search request to control various behaviors.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    Flags      INTEGER
}
```

The **Flags** value has the following format presented in big-endian byte order. X denotes unused bits that must be set to 0 by the client and that must be ignored by the server:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	S	S
																														S	S
																														F	F
																														P	P
																														R	R

The **Flags** value is a combination of zero or more bit flags from the following table:

Bit flag name and value	Description
SERVER_SEARCH_FLAG_DOMAIN_SCOPE (SSFDS) 1	Prevents continuation references from being generated when the search results are returned. This performs the same function as the LDAP_SERVER_DOMAIN_SCOPE_OID control.
SERVER_SEARCH_FLAG_PHANTOM_ROOT (SSFPR) 2	Instructs the server to search all NC-replicas that are subordinate to the search base, even if the search base is not instantiated on the server. This will cause the search to be executed over all NC-replicas held on the DC that are subordinate to the search base. This enables search bases such as the empty string, which would cause the server to search all of the NC-replicas that it holds.

Sending this control to the DC does not cause the server to include any controls in its response.

### 3.1.1.3.4.1.13 LDAP\_SERVER\_SORT\_OID and LDAP\_SERVER\_RESP\_SORT\_OID

This request control and its corresponding response control, LDAP\_SERVER\_RESP\_SORT\_OID, are documented in [\[RFC2891\]](#).

DCs only support sorting on a single attribute at a time. Therefore, the client constructs a SortKeyList containing only one sequence. DCs running Windows 2000 do not support ordering rules when sorting, so the client omits the orderingRule field of the SortKeyList when sending this control to a DC running Windows 2000; sorting uses the *English: United States* sort order. DCs running Windows Server 2003 and later support ordering rules for the sort orders specified in the following table; if no ordering rule is specified, the DC uses the *English: United States* sort order. Section [7.5](#) specifies the effect of each sort order. Section [2.2.1](#) specifies the mapping between the sort orders below and the LCIDs used in section [7.5](#).

Ordering rule OID	Sort Order
1.2.840.113556.1.4.1461	<i>Afrikaans</i>
1.2.840.113556.1.4.1462	<i>Albanian</i>
1.2.840.113556.1.4.1463	<i>Arabic: Saudi Arabia</i>
1.2.840.113556.1.4.1464	<i>Arabic: Iraq</i>
1.2.840.113556.1.4.1465	<i>Arabic: Egypt</i>

Ordering rule OID	Sort Order
1.2.840.113556.1.4.1466	<i>Arabic: Libya</i>
1.2.840.113556.1.4.1467	<i>Arabic: Algeria</i>
1.2.840.113556.1.4.1468	<i>Arabic: Morocco</i>
1.2.840.113556.1.4.1469	<i>Arabic: Tunisia</i>
1.2.840.113556.1.4.1470	<i>Arabic: Oman</i>
1.2.840.113556.1.4.1471	<i>Arabic: Yemen</i>
1.2.840.113556.1.4.1472	<i>Arabic: Syria</i>
1.2.840.113556.1.4.1473	<i>Arabic: Jordan</i>
1.2.840.113556.1.4.1474	<i>Arabic: Lebanon</i>
1.2.840.113556.1.4.1475	<i>Arabic: Kuwait</i>
1.2.840.113556.1.4.1476	<i>Arabic: UAE</i>
1.2.840.113556.1.4.1477	<i>Arabic: Bahrain</i>
1.2.840.113556.1.4.1478	<i>Arabic: Qatar</i>
1.2.840.113556.1.4.1479	<i>Armenian</i>
1.2.840.113556.1.4.1480	<i>Assamese</i>
1.2.840.113556.1.4.1481	<i>Azeri: Latin</i>
1.2.840.113556.1.4.1482	<i>Azeri: Cyrillic</i>
1.2.840.113556.1.4.1483	<i>Basque</i>
1.2.840.113556.1.4.1484	<i>Belarussian</i>
1.2.840.113556.1.4.1485	<i>Bengali</i>
1.2.840.113556.1.4.1486	<i>Bulgarian</i>
1.2.840.113556.1.4.1487	<i>Burmese</i>
1.2.840.113556.1.4.1488	<i>Catalan</i>
1.2.840.113556.1.4.1489	<i>Chinese: Taiwan</i>
1.2.840.113556.1.4.1490	<i>Chinese: PRC</i>
1.2.840.113556.1.4.1491	<i>Chinese: Hong Kong SAR</i>
1.2.840.113556.1.4.1492	<i>Chinese: Singapore</i>
1.2.840.113556.1.4.1493	<i>Chinese: Macau SAR</i>
1.2.840.113556.1.4.1494	<i>Croatian</i>

Ordering rule OID	Sort Order
1.2.840.113556.1.4.1495	<i>Czech</i>
1.2.840.113556.1.4.1496	<i>Danish</i>
1.2.840.113556.1.4.1497	<i>Dutch</i>
1.2.840.113556.1.4.1498	<i>Dutch:Belgium</i>
1.2.840.113556.1.4.1499	<i>English: United States</i>
1.2.840.113556.1.4.1500	<i>English: United Kingdom</i>
1.2.840.113556.1.4.1665	<i>English: Australia</i>
1.2.840.113556.1.4.1666	<i>English: Canada</i>
1.2.840.113556.1.4.1667	<i>English: New Zealand</i>
1.2.840.113556.1.4.1668	<i>English: Ireland</i>
1.2.840.113556.1.4.1505	<i>English: South Africa</i>
1.2.840.113556.1.4.1506	<i>English: Jamaica</i>
1.2.840.113556.1.4.1507	<i>English: Caribbean</i>
1.2.840.113556.1.4.1508	<i>English: Belize</i>
1.2.840.113556.1.4.1509	<i>English:Trinidad</i>
1.2.840.113556.1.4.1510	<i>English: Zimbabwe</i>
1.2.840.113556.1.4.1511	<i>English: Philippines</i>
1.2.840.113556.1.4.1512	<i>Estonian</i>
1.2.840.113556.1.4.1513	<i>Faeroese</i>
1.2.840.113556.1.4.1514	<i>Persian</i>
1.2.840.113556.1.4.1515	<i>Finnish</i>
1.2.840.113556.1.4.1516	<i>French: France</i>
1.2.840.113556.1.4.1517	<i>French: Belgium</i>
1.2.840.113556.1.4.1518	<i>French: Canada</i>
1.2.840.113556.1.4.1519	<i>French: Switzerland</i>
1.2.840.113556.1.4.1520	<i>French: Luxembourg</i>
1.2.840.113556.1.4.1521	<i>French: Monaco</i>
1.2.840.113556.1.4.1522	<i>Georgian</i>
1.2.840.113556.1.4.1523	<i>German: Germany</i>

Ordering rule OID	Sort Order
1.2.840.113556.1.4.1524	<i>German: Switzerland</i>
1.2.840.113556.1.4.1525	<i>German: Austria</i>
1.2.840.113556.1.4.1526	<i>German: Luxembourg</i>
1.2.840.113556.1.4.1527	<i>German: Liechtenstein</i>
1.2.840.113556.1.4.1528	<i>Greek</i>
1.2.840.113556.1.4.1529	<i>Gujarati</i>
1.2.840.113556.1.4.1530	<i>Hebrew</i>
1.2.840.113556.1.4.1531	<i>Hindi</i>
1.2.840.113556.1.4.1532	<i>Hungarian</i>
1.2.840.113556.1.4.1533	<i>Icelandic</i>
1.2.840.113556.1.4.1534	<i>Indonesian</i>
1.2.840.113556.1.4.1535	<i>Inukitut</i>
1.2.840.113556.1.4.1536	<i>Italian:Italy</i>
1.2.840.113556.1.4.1537	<i>Italian:Switzerland</i>
1.2.840.113556.1.4.1538	<i>Japanese</i>
1.2.840.113556.1.4.1539	<i>Kannada</i>
1.2.840.113556.1.4.1540	<i>Kashmiri Arabic</i>
1.2.840.113556.1.4.1541	<i>Kashmiri</i>
1.2.840.113556.1.4.1542	<i>Kazakh</i>
1.2.840.113556.1.4.1543	<i>Khmer</i>
1.2.840.113556.1.4.1544	<i>Kirghiz</i>
1.2.840.113556.1.4.1545	<i>Konkani</i>
1.2.840.113556.1.4.1546	<i>Korean</i>
1.2.840.113556.1.4.1547	<i>Korean:Johab</i>
1.2.840.113556.1.4.1548	<i>Latvian</i>
1.2.840.113556.1.4.1549	<i>Lithuanian</i>
1.2.840.113556.1.4.1550	<i>Macedonian FYROM</i>
1.2.840.113556.1.4.1551	<i>Malaysian</i>
1.2.840.113556.1.4.1552	<i>Malay Brunei Darussalam</i>

Ordering rule OID	Sort Order
1.2.840.113556.1.4.1553	<i>Malayalam</i>
1.2.840.113556.1.4.1554	<i>Maltese</i>
1.2.840.113556.1.4.1555	<i>Manipuri</i>
1.2.840.113556.1.4.1556	<i>Marathi</i>
1.2.840.113556.1.4.1557	<i>Nepali:Nepal</i>
1.2.840.113556.1.4.1558	<i>Norwegian:Bokmal</i>
1.2.840.113556.1.4.1559	<i>Norwegian:Nynorsk</i>
1.2.840.113556.1.4.1560	<i>Oriya</i>
1.2.840.113556.1.4.1561	<i>Polish</i>
1.2.840.113556.1.4.1562	<i>Portuguese:Brazil</i>
1.2.840.113556.1.4.1563	<i>Portuguese:Portugal</i>
1.2.840.113556.1.4.1564	<i>Punjabi</i>
1.2.840.113556.1.4.1565	<i>Romanian</i>
1.2.840.113556.1.4.1566	<i>Russian</i>
1.2.840.113556.1.4.1567	<i>Sanskrit</i>
1.2.840.113556.1.4.1568	<i>Serbian:Cyrillic</i>
1.2.840.113556.1.4.1569	<i>Serbian:Latin</i>
1.2.840.113556.1.4.1570	<i>Sindhi:India</i>
1.2.840.113556.1.4.1571	<i>Slovak</i>
1.2.840.113556.1.4.1572	<i>Slovenian</i>
1.2.840.113556.1.4.1573	<i>Spanish: SpainTraditional Sort</i>
1.2.840.113556.1.4.1574	<i>Spanish: Mexico</i>
1.2.840.113556.1.4.1575	<i>Spanish: SpainModern Sort</i>
1.2.840.113556.1.4.1576	<i>Spanish: Guatemala</i>
1.2.840.113556.1.4.1577	<i>Spanish: Costa Rica</i>
1.2.840.113556.1.4.1578	<i>Spanish: Panama</i>
1.2.840.113556.1.4.1579	<i>Spanish: Dominican Republic</i>
1.2.840.113556.1.4.1580	<i>Spanish: Venezuela</i>
1.2.840.113556.1.4.1581	<i>Spanish: Colombia</i>



Ordering rule OID	Sort Order
1.2.840.113556.1.4.1582	<i>Spanish: Peru</i>
1.2.840.113556.1.4.1583	<i>Spanish: Argentina</i>
1.2.840.113556.1.4.1584	<i>Spanish: Ecuador</i>
1.2.840.113556.1.4.1585	<i>Spanish: Chile</i>
1.2.840.113556.1.4.1586	<i>Spanish: Uruguay</i>
1.2.840.113556.1.4.1587	<i>Spanish: Paraguay</i>
1.2.840.113556.1.4.1588	<i>Spanish: Bolivia</i>
1.2.840.113556.1.4.1589	<i>Spanish: El Salvador</i>
1.2.840.113556.1.4.1590	<i>Spanish: Honduras</i>
1.2.840.113556.1.4.1591	<i>Spanish: Nicaragua</i>
1.2.840.113556.1.4.1592	<i>Spanish: Puerto Rico</i>
1.2.840.113556.1.4.1593	<i>Swahili: Kenya</i>
1.2.840.113556.1.4.1594	<i>Swedish</i>
1.2.840.113556.1.4.1595	<i>Swedish: Finland</i>
1.2.840.113556.1.4.1596	<i>Tamil</i>
1.2.840.113556.1.4.1597	<i>Tatar: Tatarstan</i>
1.2.840.113556.1.4.1598	<i>Telugu</i>
1.2.840.113556.1.4.1599	<i>Thai</i>
1.2.840.113556.1.4.1600	<i>Turkish</i>
1.2.840.113556.1.4.1601	<i>Ukrainian</i>
1.2.840.113556.1.4.1602	<i>Urdu: Pakistan</i>
1.2.840.113556.1.4.1603	<i>Urdu: India</i>
1.2.840.113556.1.4.1604	<i>Uzbek: Latin</i>
1.2.840.113556.1.4.1605	<i>Uzbek: Cyrillic</i>
1.2.840.113556.1.4.1606	<i>Vietnamese</i>
1.2.840.113556.1.4.1607	<i>Japanese: XJIS</i>
1.2.840.113556.1.4.1608	<i>Japanese: Unicode</i>
1.2.840.113556.1.4.1609	<i>Chinese: Big5</i>
1.2.840.113556.1.4.1610	<i>Chinese: PRCP</i>

Ordering rule OID	Sort Order
1.2.840.113556.1.4.1611	<i>Chinese: Unicode</i>
1.2.840.113556.1.4.1612	<i>Chinese: PRC</i>
1.2.840.113556.1.4.1613	<i>Chinese: BOPOMOFO</i>
1.2.840.113556.1.4.1614	<i>Korean: KSC</i>
1.2.840.113556.1.4.1615	<i>Korean: Unicode</i>
1.2.840.113556.1.4.1616	<i>German Phone Book</i>
1.2.840.113556.1.4.1617	<i>Hungarian: Default</i>
1.2.840.113556.1.4.1618	<i>Hungarian: Technical</i>
1.2.840.113556.1.4.1619	<i>Georgian: Traditional</i>
1.2.840.113556.1.4.1620	<i>Georgian: Modern</i>

Windows Server 2008 supports an additional sort behavior called "phonetic display name sort". This behavior is triggered by specifying "msDS-PhoneticDisplayName;extended" as the attributeType in the SortKeyList ([RFC2891](#) section 1.1). When this option is present, the DC checks that the **LDAP** request satisfies the following requirements:

- The operation is an **LDAP** search request.
- The orderingRule field specifies the Japanese sort order (namely, "1.2.840.113556.1.4.1538")
- The LDAP\_CONTROL\_VLVREQUEST control is attached to the search.
- The search request has been sent to a global catalog port (port 3268 or 3269).
- The scope of the search request is wholeSubtree.
- The base object of the search request specifies the DN "".

If all of the above criteria are satisfied, the DC performs a phonetic display name sort. In this sort, the search results are sorted on the [msDS-PhoneticDisplayName](#) attribute, using the Japanese sort order, in the normal fashion, except that if an object O does not have a value for the [msDS-PhoneticDisplayName](#) attribute but does have a value V for the [displayName](#) attribute, the server treats V as the value of O! [msDS-PhoneticDisplayName](#) for the purposes of the sort.

For example, consider an unsorted search result set consisting of four objects, as shown in the following table. Note that object #2 does not have a value for [msDS-PhoneticDisplayName](#).

Object #	<a href="#">msDS-PhoneticDisplayName</a> value	<a href="#">displayName</a> value
1	A	C
2		D
3	B	E
4	F	C

Assuming for the purpose of this example that the letters A...Z sort in the order {A, ..., Z}, the results of performing a phonetic display name sort on the above is the following:

Object #	<a href="#">msDS-PhoneticDisplayName</a> value	<a href="#">displayName</a> value
1	A	C
3	B	E
2		D
4	F	C

In particular, object #2 was placed before object #4 because the sort treated it as if it had the value "D" for its [msDS-PhoneticDisplayName](#) attribute.

#### 3.1.1.3.4.1.14 LDAP\_SERVER\_SHOW\_DELETED\_OID

The LDAP\_SERVER\_SHOW\_DELETED\_OID control is used with an **LDAP** search operation to specify that the search results are to include any deleted objects that match the search filter. These deleted objects (tombstones) are objects that have been removed from the directory via a deletion operation, but whose tombstone lifetime has not yet expired (and thus the objects have not yet been permanently removed from the directory).

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

#### 3.1.1.3.4.1.15 LDAP\_SERVER\_TREE\_DELETE\_OID

The LDAP\_SERVER\_TREE\_DELETE\_OID control is used with an **LDAP** delete operation to cause the server to recursively delete the entire subtree of objects located underneath the object specified in the delete operation. The object specified in the delete operation is also deleted.

If the server is unable to delete the entire tree in a single **LDAP** delete request, it deletes as much of the tree as it can, but does not delete the root of the tree (the object specified in the delete operation) and returns the error code *adminLimitExceeded* ([\[RFC2251\]](#) section 4.1.10).

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

#### 3.1.1.3.4.1.16 LDAP\_SERVER\_VERIFY\_NAME\_OID

The LDAP\_SERVER\_VERIFY\_NAME\_OID control is used with **LDAP** Add and Modify requests to identify the GC server used to verify the existence of any objects pointed to by DN attribute values (as specified in section [3.1.1.1.6](#)). If the DC needs to call a GC server while processing the Add or Modify request, it calls the GC server specified in this control. If this control is not used, the DC is free to call any GC server in the forest.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    Flags          INTEGER
    ServerName     OCTET STRING
```

}

where **Flags** is ignored and **ServerName** is a UTF-16 encoded **Unicode** string containing the fully qualified DNS name of the global catalog server to contact for verification. Sending this control to the DC does not cause the server to include any controls in its response.

If the **LDAP** Add or Modify request needs to call a GC server and the server designated by this control in the request is not available or is not a GC server, the Add or Modify request fails with the **LDAP** error *unavailable*.

### 3.1.1.3.4.1.17 LDAP\_CONTROL\_VLVREQUEST and LDAP\_CONTROL\_VLVRESPONSE

The LDAP\_CONTROL\_VLVREQUEST control is used with an **LDAP** search operation to retrieve a subset of the objects that satisfy the search request. This control permits the client to specify a particular object (the "target object") in a sorted set of search results, and to request that the server return a specified number of objects before and after the target object in addition to the target object itself. "Before" and "after" the target object are relative to the sort order of the search result set. When using this control, the LDAP\_SERVER\_SORT\_OID control must also be used to specify a sort order for the search result set.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure (maxInt is defined in [RFC2251](#) section 4.1.1):

```
SEQUENCE {
  beforeCount      INTEGER (0..maxInt),
  afterCount       INTEGER (0..maxInt),
  CHOICE {
    byoffset        [0] SEQUENCE {
      offset        INTEGER (0 .. maxInt),
      contentCount  INTEGER (0 .. maxInt)
    },
    greaterThanOrEqual [1] AssertionValue
  },
  contextID        OCTET STRING OPTIONAL
}
```

where **beforeCount** indicates how many objects before the target object are to be included in the search results, and **afterCount** indicates how many objects after the target object are to be included in the search results.

**byoffset** and **greaterThanOrEqual** provide two mutually-exclusive ways of specifying the target object. These will now be discussed in turn.

First, the target object can be specified by its position relative to the first object in the sorted set of objects that satisfy the search request, in which case the **byoffset** choice is used. In this case, **contentCount** contains the client's estimation of the total number of objects that satisfy the search criteria. If the client specifies 0 for contentCount, it is as if the client had specified a number identical to the server's estimate of the total number of objects that satisfy the search criteria - the quantity **serverContentCount** below. **offset** is used with **contentCount** to specify the position (relative to the first object in the sorted set of search results) of the object to use as the target object according to the following formula:

$$p = \text{serverContentCount} * (\text{offset} / \text{contentCount})$$

where **serverContentCount** is the DC's estimate of the total number of objects that satisfy the search criteria. The object located at position **p** in the sorted list of search results is used as the target object.

A value of **offset** equal to 1 means that the target object is the first object in the search result set, while a value of **offset** equal to **contentCount** means the target object is the last object in the search result set. If the client specified 0 for **contentCount**, then **p = offset** in the formula above so the target object is **offset-1** objects beyond the first object in the search result set.

The second means of specifying the target object is by the **greaterThanOrEqual** choice, instead of the **byoffset** choice. In this case, **greaterThanOrEqual** is an AssertionValue as defined in [\[RFC2251\]](#) section 4.1.7. The target object is the first object in the sorted result set for which the value of the attribute on which it is sorted (that is, the attribute specified by **attributeType** in the LDAP\_SERVER\_SORT\_OID control) is greater than or equal to the value specified by **greaterThanOrEqual**. However, if the sort order is reversed (by specifying that the **reverseOrder** field of the LDAP\_SERVER\_SORT\_OID control is true), then the target object is the first object for which the sort attribute value is less than or equal to the **greaterThanOrEqual** value.

If the **contextID** field is present, it is the opaque value returned by the DC as the **contextIDServer** field of the LDAP\_CONTROL\_VLVRESPONSE control returned with the search response to the previous search over the same "list" as this search. A "list" is a sorted set of search results, defined by a search request value sent to a particular DC over a particular **LDAP** connection. The client omits this field if this is the first search request that included the LDAP\_CONTROL\_VLVREQUEST control for the "list", or if the client did not retain the **contextIDServer** field of the previous LDAP\_CONTROL\_VLVRESPONSE for the "list". The presence or absence of the **contextID** field in the request only affects performance.

When the server receives a search request with the LDAP\_CONTROL\_VLVREQUEST control attached to it, it includes a response control in the search response. The controlType field of the returned Control structure is set to the OID of the LDAP\_CONTROL\_VLVRESPONSE control, and the controlValue is the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    targetPosition      INTEGER (0 .. maxInt),
    contentCount        INTEGER (0 .. maxInt),
    virtualListViewResult ENUMERATED {
        success                (0),
        operationsError        (1),
        unwillingToPerform     (53),
        insufficientAccessRights (50),
        busy                   (51),
        timeLimitExceeded      (3),
        adminLimitExceeded     (11),
        sortControlMissing     (60),
        offsetRangeError       (61),
        other                   (80)
    },
    contextIDServer      OCTET STRING OPTIONAL
}
```

where **targetPosition** is the position of the target object relative to the beginning of the sorted set of search results, **contentCount** is the server's estimate of the total number of objects that satisfy the search request, and **virtualListViewResult** is an **LDAP** error code that indicates the success or failure of the DC in processing the LDAP\_CONTROL\_VLVREQUEST control. These codes have the same meanings as defined for **LDAP** in [\[RFC2251\]](#), but they pertain specifically to the processing of

the control. **contextIDServer** is the opaque value described in the specification of the **contextID** field above.

The Active Directory implementation of VLV is based on that described in [\[VLVDRAFT\]](#). Although implementors may consult that document as an informative reference, the preceding description documents the protocol as implemented by Active Directory. No claim is made with regards to Active Directory's conformance or nonconformance with the protocol as specified in [\[VLVDRAFT\]](#).

### 3.1.1.3.4.1.18 LDAP\_SERVER\_ASQ\_OID

The LDAP\_SERVER\_ASQ\_OID control is used with an **LDAP** search operation. When this control is used, the search is not performed against the object specified in the search, or the objects located underneath that object, but rather against the set of objects named by an attribute of Object(DS-DN) syntax located on the object specified by the base DN of the search request. The specific attribute to use to scope the search is named in the control. Only searches of base object scope can be used with the LDAP\_SERVER\_ASQ\_OID control.

For example, suppose there is an object *o* and a multivalued attribute *A* of Object(DS-DN) syntax such that *o.A* contains the DNs of objects *o1*, *o2*, and *o3*. An **LDAP** base-scope search operation that targets object *o*, with the LDAP\_SERVER\_ASQ\_OID control attached and specifying the *A* attribute, will cause the server to perform the search not against object *o* but against objects *o1*, *o2*, and *o3*.

When sending this control to the DC, the *controlValue* field is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    sourceAttribute    OCTET STRING
}
```

where **sourceAttribute** is a UTF-8 string that specifies the **LDAP** display name of the attribute to use to scope the search (for example, attribute *A* in the previous example).

When the server receives a search request with the LDAP\_SERVER\_ASQ\_OID control attached to it, it includes a response control in the search response. The *controlType* field of the returned Control structure is set to the OID of the LDAP\_SERVER\_ASQ\_OID control, and the *controlValue* is the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    searchResults      ENUMERATED {
        success                (0),
        invalidAttributeSyntax (21),
        unwillingToPerform    (53),
        affectsMultipleDSAs    (71)
    },
}
```

where the meaning of **searchResults** is as indicated in the following table:

searchResult name	searchResult value	Description
<i>success</i>	0	Search results are returned for all objects referenced by

searchResult name	searchResult value	Description
		sourceAttribute.
<i>invalidAttributeSyntax</i>	21	sourceAttribute is not of Object(DS-DN) syntax.
<i>unwillingToPerform</i>	53	The search scope was not set to base object scope.
<i>affectsMultipleDSAs</i>	71	Partial results were returned, but not all the objects were available on the DC.

The search results consist of each object that is specified by the **sourceAttribute** attribute and that matches the search filter returned as a SearchResultEntry (defined in [RFC2251](#) section 4.5.2) containing the attributes specified in the attribute list of the search request. If any of the objects specified by **sourceAttribute** are not available on the DC, the search results include all of the objects that are available on the DC, and the **searchResults** return value is set to the *affectsMultipleDSAs* error code to indicate that some data that might be otherwise available is not present in the results.

#### 3.1.1.3.4.1.19 LDAP\_SERVER\_QUOTA\_CONTROL\_OID

This control is used with an **LDAP** search operation to retrieve the quota of a user. When used with an **LDAP** search operation which queries the **constructed attributes** [msDS-QuotaEffective](#) and [msDS-QuotaUsed](#) on the [msDS-QuotaContainer](#) object, the server will return the quota of the user who is specified by the control, rather than the quota of the user that the connection is authenticated as.

The caller must have the RIGHT\_DS\_READ\_PROPERTY right on the Quotas container (described in section [7.1.1.4.3](#)) to retrieve the quota of a user other than the user whom the caller is authenticated as. To retrieve their own quota, the caller must have either the RIGHT\_DS\_READ\_PROPERTY right or the DS-Query-Self-Quota **control access right** on the Quotas container. These access checks are also specified in section [3.1.1.4.4](#).

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    querySID    OCTET STRING
}
```

where **querySID** is the SID, in binary form, of the user whose quota is to be retrieved (the binary form of SIDs is documented in [\[MS-DTYP\]](#) section **2.4.2**). Sending this control to the DC does not cause the server to include any controls in its response.

#### 3.1.1.3.4.1.20 LDAP\_SERVER\_SHUTDOWN\_NOTIFY\_OID

This control is used with an **LDAP** Search request. The Search request has base object scope. The base DN of the search is the DN of the DC's [nTDSDSA](#) object, and the search filter is "(objectClass=\*)". If the application sending the search request is not running on the same computer as the DC, the result is **LDAP** error *unwillingToPerform*.

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

This control is only supported on the Small Business Server version of the Windows operating system.

Because this control only has an effect for applications running on the same machine as the DC, the effects of this control are not observable on the network. This control causes DC to notify the client when the DC is shutting down. When the DC receives a search request with this control attached, it does not immediately send a response to the request. Instead, it sends the SearchResultDone response (see [RFC2251](#) section 4.5.2) to the request when the DC is shutting down.

#### 3.1.1.3.4.1.21 LDAP\_SERVER\_FORCE\_UPDATE\_OID

A DC does not perform originating updates that do not affect the state of the DC. For example, given an **LDAP** Modify operation that sets the value of an attribute A to a value V, if the value of A is already V prior to the Modify operation, the DC skips the update and returns success. The **stamp** associated with A is not changed and the Modify operation does not cause **replication traffic**.

When the LDAP\_SERVER\_FORCE\_UPDATE\_OID control is attached to an update operation, the DC does not perform the optimization described in the previous paragraph. The update always generates a new **stamp** for the attribute or link value, and always replicates.

When sending this control to a DC, the controlValue field of the Control structure is omitted. Sending this control to a DC does not cause the DC to include any controls in its response.

#### 3.1.1.3.4.1.22 LDAP\_SERVER\_RANGE\_RETRIEVAL\_NOERR\_OID

This control is used to modify the behavior of a range retrieval operation (see section [3.1.1.3.1.3.2](#)). When this control is not specified, if range retrieval is being performed on an attribute whose values are forward link values or back link values and the value of **low** is greater than or equal to the number of values in the attribute, the DC will return the error *operationsError*. If this control is specified, no error is returned in this case (and no values are returned). For example, if an object has a [member](#) attribute with 500 values, performing the range retrieval "member;range=500-\*" will return *operationsError* without this control and *success* with this control.

When sending this control to a DC, the controlValue field of the Control structure is omitted. Sending this control to a DC does not cause the DC to include any controls in its response.

#### 3.1.1.3.4.1.23 LDAP\_SERVER\_RODC\_DCPROMO\_OID

If this control is specified and the caller does not have the DS-Install-Replica **control access right** on the root of the default NC, the result is **LDAP** error *insufficientAccessRights* / *ERROR\_ACCESS\_DENIED*.

If the request is an Add of an object of class [user](#) or a subclass of [user](#), the presence of this control has the following effects:

- The DC generates a value in the range [1 .. 65535] that is not used as a value of the [msDS-SecondaryKrbTgtNumber](#) attribute on an object in this **domain**, and assigns the generated value to the [msDS-SecondaryKrbTgtNumber](#) attribute of the created object. If no such value exists, the result is **LDAP** error *other* / *ERROR\_NO\_SYSTEM\_RESOURCES*.

**Note** In Windows Server 2008, the DC servicing the request need not be the PDC FSMO role owner.

If the request is an Add of an object of class [nTDSDSA](#), the presence of this control has the following effects:



- The DC creates the [nTDSDSA](#) object using the information provided in the Add request. The only special effect of the control is to perform the checking of the DS-Install-Replica **control access right** (specified above) to authorize the [nTDSDSA](#) object creation. Without this control, an Add that attempts to create an [nTDSDSA](#) object will fail because the class is system-only (section [3.1.1.2.4.8](#)).

When sending this control to a DC, the controlValue field of the Control structure is omitted. Sending this control to a DC does not cause the DC to include any controls in its response.

#### 3.1.1.3.4.1.24 LDAP\_SERVER\_INPUT\_DN\_OID

This control is used to specify the DN of an object during certain **LDAP** operations.

When used with an **LDAP** search operation that queries the **constructed attribute** [msDS-IsUserCachableAtRode](#) on a computer object representing an RODC, the server will return the administrative policy regarding whether the secrets of the security principal represented by the DN specified in the control can be cached on the RODC. The caller must have the DS-Replication-Secrets-Synchronize control access right on the root of the default NC, otherwise *operationsError / ERROR\_DS\_DRA\_ACCESS\_DENIED* is returned. This access check is also specified in section [3.1.1.4.4](#).

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    InputDN    OCTET STRING
}
```

where **InputDN** is a UTF-8 encoding of the DN of a security principal. The DN may be either a RFC 2253-style DN, or one of the alternative DN formats described in section [3.1.1.3.1.2.4](#).

#### 3.1.1.3.4.2 LDAP Extended Operations

LDAP extended operations are an extensibility mechanism in version 3 of the **LDAP** protocol, as discussed in [\[RFC2251\]](#) section 4.12. The following sections describe the **LDAP** extended operations implemented by DCs in Windows Server 2008 and previous versions.

The **LDAP** extended operations supported by a DC are exposed as OIDs in the [supportedExtension](#) attribute of the rootDSE. Each OID is mapped to a human-readable name as shown in the table below.

Extended operation name	OID
LDAP_SERVER_FAST_BIND_OID	1.2.840.113556.1.4.1781
LDAP_SERVER_START_TLS_OID	1.3.6.1.4.1.1466.20037
LDAP_TTL_REFRESH_OID	1.3.6.1.4.1.1466.101.119.1
LDAP_SERVER_WHO_AM_I_OID	1.3.6.1.4.1.4203.1.11.3

Only Windows Server 2003 and later DCs support extended operations. The following table specifies the set of **LDAP** extended operations supported in each Windows Server version that supports extended operations.

Extended operation name	Windows Server 2003	Windows Server 2008
LDAP_SERVER_FAST_BIND_OID	X	X
LDAP_SERVER_START_TLS_OID	X	X
LDAP_TTL_REFRESH_OID	X	X
LDAP_SERVER_WHO_AM_I_OID		X

Each of these operations is executed by performing an **LDAP** ExtendedRequest operation, specifying the OID of the extended operation as the requestName field in the ExtendedRequest (see [\[RFC2251\]](#) section 4.12). The server responds to an ExtendedRequest by returning an ExtendedResponse, the fields of which are also documented in section 4.12 of the RFC.

### 3.1.1.3.4.2.1 LDAP\_SERVER\_FAST\_BIND\_OID

The presence of this OID in the supportedExtension attribute indicates that the DC provides support for fast bind mode. Fast bind mode causes the server to validate the credentials of **LDAP** bind requests that are sent on the connection, but unlike a regular (non-fast bind mode) bind, the DC performs **Authentication** only; it does not perform **authorization** steps such as computing the group memberships of the authenticated security principal. The LDAP\_SERVER\_FAST\_BIND\_OID operation puts the **LDAP** connection on which it was sent into fast bind mode on the DC. The server will reject this operation with the error *unwillingToPerform* ([\[RFC2251\]](#) section 4.1.10) if a successful bind has already been performed on the connection.

Note that a client can retrieve the supportedExtension attribute from the root DSE without having first performed a bind (since the supportedExtension attribute is anonymously accessible, and **LDAP** v3 does not require a bind to be performed for anonymous access). Thus, a client can determine if the server supports fast bind mode without first having to bind to the server.

Only simple binds are accepted on a connection in this mode. All other types of bind operations are rejected with the error *unwillingToPerform*. The connection is always treated as if no bind had occurred for the purposes of all other **LDAP** operations, i.e., the connection is treated as the anonymous user (that is, an anonymous bind).

To send this extended operation to the DC, the client sends an **LDAP** ExtendedRequest with the requestName field containing the operation's OID. The requestValue field is omitted. The server will return an ExtendedResponse with the responseName field containing the operation's OID and the response field omitted.

The following shows a typical sequence of operations in fast bind:

- The client establishes an **LDAP** connection with the DC.
- (Optional) The client checks the supportedExtension attribute on the root DSE to confirm that the DC supports fast bind mode.
- The client sends the LDAP\_SERVER\_FAST\_BIND\_OID extended operation to the DC to put the **LDAP** connection into fast bind mode.
- The client performs one or more simple binds on the connection.

### 3.1.1.3.4.2.2 LDAP\_SERVER\_START\_TLS\_OID

This presence of this OID in the supportedExtension attribute indicates that the DC provides support for the **LDAP** StartStopTLS protocol as described in [\[RFC2830\]](#).

A connection cannot be put into TLS mode if it is using an integrity validation or encryption mechanism that was negotiated as part of a bind request (for example, a SASL-layer encryption mechanism). Such an attempt will be rejected with the error *operationsError* ([\[RFC2251\]](#) section 4.2.1).

### 3.1.1.3.4.2.3 LDAP\_TTL\_REFRESH\_OID

The presence of this OID in the supportedExtension attribute indicates that the DC provides support for dynamic objects as defined in [\[RFC2589\]](#). This extended operation is sent to the DC to refresh a specific dynamic object that has already been created. The extended operation is documented in [\[RFC2589\]](#).

### 3.1.1.3.4.2.4 LDAP\_SERVER\_WHO\_AM\_I\_OID

The presence of this OID in the supportedExtension attribute indicates that the DC provides support for the "Who Am I?" **LDAP** extended operation described in [\[RFC4532\]](#). Active Directory implements this operation in conformance with that RFC.

If the client is authenticated as a Windows security principal, the *authzId* returned in the response will contain the string "u:" followed by either (1) the NetBIOS domain name, followed by a backslash ('\'), followed by the [sAMAccountName](#) of the security principal, or (2) the SID of the security principal, in SDDL SID string format ([\[MS-DTYP\]](#)). If the client is authenticated as an AD/LDS security principal, the returned *authzId* will contain the string "dn:" followed by the DN of the security principal. If the client has not authenticated, the returned *authzId* will be the empty string.

Active Directory does not implement Proxied Authentication Control of [\[RFC4370\]](#), so section 4.1 of [\[RFC4532\]](#) is not applicable to Active Directory.

### 3.1.1.3.4.3 LDAP Capabilities

The following sections specify the capabilities exposed by DCs on the supportedCapabilities attribute of the rootDSE. Capabilities are exposed in that attribute as OIDs, each of which is mapped to a human-readable name as shown in the table below.

Capability name	OID
LDAP_CAP_ACTIVE_DIRECTORY_OID	1.2.840.113556.1.4.800
LDAP_CAP_ACTIVE_DIRECTORY_LDAP_INTEG_OID	1.2.840.113556.1.4.1791
LDAP_CAP_ACTIVE_DIRECTORY_V51_OID	1.2.840.113556.1.4.1670
LDAP_CAP_ACTIVE_DIRECTORY_ADAM_DIGEST	1.2.840.113556.1.4.1880
LDAP_CAP_ACTIVE_DIRECTORY_ADAM_OID	1.2.840.113556.1.4.1851
LDAP_CAP_ACTIVE_DIRECTORY_PARTIAL_SECRETS_OID	1.2.840.113556.1.4.1920
LDAP_CAP_ACTIVE_DIRECTORY_V61_OID	1.2.840.113556.1.4.1935

Not all versions of Windows Server support all the **LDAP** capabilities. The following tables indicates which capabilities are supported in which version.

Capability Name	Windows 2000	Windows 2000 Server SP3	Windows Server 2003	Windows Server 2008 AD/DS	Windows Server 2008 AD/LDS
LDAP_CAP_ACTIVE_DIRECTORY_OID	X	X	X	X	
LDAP_CAP_ACTIVE_DIRECTORY_LDAP_INTEG_OID		X	X	X	X
LDAP_CAP_ACTIVE_DIRECTORY_V51_OID			X	X	X
LDAP_CAP_ACTIVE_DIRECTORY_ADAM_DIGEST					X*
LDAP_CAP_ACTIVE_DIRECTORY_ADAM_OID					X
LDAP_CAP_ACTIVE_DIRECTORY_PARTIAL_SECRETS_OID				X*	
LDAP_CAP_ACTIVE_DIRECTORY_V61_OID				X	X

\* These capabilities are only exposed by the server in certain conditions. For each of these conditional capabilities, the section describing the capability describes the conditions that apply.

#### 3.1.1.3.4.3.1 LDAP\_CAP\_ACTIVE\_DIRECTORY\_OID

The presence of this capability indicates that the **LDAP** server is running Microsoft Active Directory and is running as AD/DS.

#### 3.1.1.3.4.3.2 LDAP\_CAP\_ACTIVE\_DIRECTORY\_LDAP\_INTEG\_OID

The presence of this capability indicates that the **LDAP** server on the DC is capable of signing and sealing on an NTLM authenticated connection, and that the server is capable of performing subsequent binds on a signed or sealed connection.

#### 3.1.1.3.4.3.3 LDAP\_CAP\_ACTIVE\_DIRECTORY\_V51\_OID

The presence of this capability indicates that the **LDAP** server is running at least the Windows Server 2003 version of Microsoft Active Directory and is running as AD/DS.

#### 3.1.1.3.4.3.4 LDAP\_CAP\_ACTIVE\_DIRECTORY\_ADAM\_DIGEST

On a DC operating as AD/LDS, the presence of this capability indicates that the DC accepts DIGEST-MD5 binds. An AD/LDS DC's DIGEST-MD5 bind functionality depends upon the value of the ADAMDisableSSI configurable setting as specified in section [3.1.1.3.4.7](#).

### 3.1.1.3.4.3.5 LDAP\_CAP\_ACTIVE\_DIRECTORY\_ADAM\_OID

The presence of this capability indicates that the **LDAP** server is running Microsoft Active Directory as AD/LDS.

### 3.1.1.3.4.3.6 LDAP\_CAP\_ACTIVE\_DIRECTORY\_PARTIAL\_SECRETS\_OID

On an Active Directory DC operating as AD/DS, the presence of this capability indicates that the DC is a RODC.

### 3.1.1.3.4.3.7 LDAP\_CAP\_ACTIVE\_DIRECTORY\_V61\_OID

The presence of this capability indicates that the **LDAP** server is running at least the Windows Server 2008 version of Microsoft Active Directory and is running as AD/DS.

### 3.1.1.3.4.4 LDAP Matching Rules (extensibleMatch)

The following sections describe the matching rules supported by DCs when performing **LDAP** search requests. Unlike, for example, extended controls and extended operations, there is no attribute exposed by the DC that specifies which matching rules it supports. The identifiers for these matching rules are used in an extensibleMatch clause in the Filter portion of a SearchRequest, as described in [RFC2251](#) section 4.5.1. Matching rules are identified by an OID that corresponds to a human-readable name as shown in the table below.

Capability Name	OID
LDAP_MATCHING_RULE_BIT_AND	1.2.840.113556.1.4.803
LDAP_MATCHING_RULE_BIT_OR	1.2.840.113556.1.4.804
LDAP_MATCHING_RULE_TRANSITIVE_EVAL	1.2.840.113556.1.4.1941

Windows 2000 and Windows Server 2003 support the LDAP\_MATCHING\_RULE\_BIT\_AND and LDAP\_MATCHING\_RULE\_BIT\_OR matching rules. Windows Server 2008 supports those two rules and the LDAP\_MATCHING\_RULE\_TRANSITIVE\_EVAL rule, in both AD/DS and AD/LDS.

#### 3.1.1.3.4.4.1 LDAP\_MATCHING\_RULE\_BIT\_AND

This rule is equivalent to a bitwise "and" operation. When this matching rule is used as a clause in a query filter, the clause is satisfied only if all the bits set to '1' in the value included in the clause correspond to bits set to '1' in the value stored in the directory.

#### 3.1.1.3.4.4.2 LDAP\_MATCHING\_RULE\_BIT\_OR

This rule is equivalent to a bitwise "or" operation. When this matching rule is used as a clause in a query filter, the clause is satisfied only if at least one of the bits set to '1' in the value included in the clause corresponds to a bit set to '1' in the value stored in the directory.

#### 3.1.1.3.4.4.3 LDAP\_MATCHING\_RULE\_TRANSITIVE\_EVAL

This rule provides recursive search of a **link attribute**. A filter F of the form "(A: 1.2.840.113556.1.4.1941:=V)", where A is a link attribute and V is a value, evaluates to True for an object whose DN is D if the following method EvalTransitiveFilter(A, V, D) returns true, and False if the method returns false. If A is not a link attribute, the filter F evaluates to Undefined.

EvalTransitiveFilter(A: attribute, V: value, D: DN)

- If A is of Object(DN-String), Object(DN-Binary), Object(OR-Name), or Object(Access-Point) syntax, let V' equal the **object\_DN** portion of V. Otherwise, let V' equal V.
- Return the value of EvalTransitiveFilterHelper(A, V', D, {})

EvalTransitiveFilterHelper(A: attribute, V': value, ToVisit: DN, Visited: SET OF DN)

- If A is of Object(DN-String), Object(DN-Binary), Object(OR-Name), or Object(Access-Point) syntax, let C be the set of the **object\_DN** components of the values of ToVisit.A. Otherwise, let C be the set of the values of ToVisit.A. Note that C is a set of DNs.
- If V' is in C, return true.
- Let Visited' equal the Visited set plus {ToVisit}.
- For each DN NextToVisit in C
  - If NextToVisit is in Visited, do nothing.
  - Let Result = EvalTransitiveFilterHelper(A, V', NextToVisit, Visited')
  - If Result is true, return true
- Return false

#### 3.1.1.3.4.5 LDAP SASL Mechanisms

The following sections describe the **Simple Authentication and Security Layer (SASL)** mechanisms that are implemented by DCs in Windows 2000 and later. SASL is described in [\[RFC2222\]](#), and the usage of SASL and other **Authentication** methods in **LDAP** is described in [\[RFC2829\]](#). The SASL mechanisms supported by a DC are exposed as strings in the *supportedSASLMechanisms* attribute of the rootDSE.

Not all versions of Windows Server support all the **LDAP** SASL mechanisms. The following tables indicates which SASL mechanisms are supported in which version.

Mechanism Name	Windows 2000 RTM	Windows Server 2003 RTM and later
GSSAPI	<b>X</b>	<b>X</b>
GSS-SPNEGO	<b>X</b>	<b>X</b>
EXTERNAL		<b>X</b>
DIGEST-MD5		<b>X</b>

Additional details of **LDAP** authentication in Active Directory are in section [5.1](#).

##### 3.1.1.3.4.5.1 GSSAPI

The presence of the "GSSAPI" string value in the *supportedSASLMechanisms* attribute indicates that the DC accepts the GSSAPI security mechanism for **LDAP** bind requests. The GSSAPI mechanism for SASL is described in [\[RFC2222\]](#) section 7.2, and GSSAPI is described in more detail in [\[RFC2078\]](#).

Active Directory supports Kerberos when using GSSAPI, see [\[MS-KILE\]](#) and [\[RFC1964\]](#) for details of Kerberos.

#### 3.1.1.3.4.5.2 GSS-SPNEGO

The presence of the "GSS-SPNEGO" string value in the *supportedSASLMechanisms* attribute indicates that the DC accepts the GSS-SPNEGO security mechanism for **LDAP** bind requests. This mechanism is documented in [\[RFC4178\]](#). Active Directory supports Kerberos (see [\[MS-KILE\]](#)) and NTLM (see [\[MS-NLMP\]](#)) when using GSS-SPNEGO.

#### 3.1.1.3.4.5.3 EXTERNAL

The presence of the "EXTERNAL" string value in the *supportedSASLMechanisms* attribute indicates that the DC accepts external security mechanisms for **LDAP** bind requests. The EXTERNAL SASL mechanism is described in [\[RFC2222\]](#) section 7.4, and [\[RFC2829\]](#). In the case of DCs, the external **Authentication** information used to validate the identity of the client making the bind request comes from the client certificate presented by the client during the **SSL/TLS handshake** that occurs in response to the client sending an LDAP\_SERVER\_START\_TLS\_OID extended operation. When the server receives an EXTERNAL SASL bind following a successful LDAP\_SERVER\_START\_TLS\_OID extended operation in which a valid certificate was presented by the client, the server causes the connection to be bound as the identity represented by that certificate.

#### 3.1.1.3.4.5.4 DIGEST-MD5

The presence of the "DIGEST-MD5" string value in the *supportedSASLMechanisms* attribute indicates that the DC accepts the **digest** security mechanism for **LDAP** bind requests. The usage of **digest** authentication with **LDAP** is documented in [\[RFC2829\]](#) section 6.1, and in [\[RFC2831\]](#).

#### 3.1.1.3.4.6 LDAP Policies

The DC's **LDAP** interface supports various policies that can be configured by an administrator. The names of these policies are listed on the *supportedLDAPPolicies* attribute on the rootDSE. These policies are listed in the table below, which also lists which versions of Windows support which policies.

Policy Name	Windows 2000	Windows Server 2003 and later
MaxActiveQueries	X*	
InitRecvTimeout	X	X
MaxConnections	X	X
MaxConnIdleTime	X	X
MaxDatagramRecv	X	X
MaxNotificationPerConn	X	X
MaxPoolThreads	X	X
MaxReceiveBuffer	X	X
MaxPageSize	X	X

Policy Name	Windows 2000	Windows Server 2003 and later
MaxQueryDuration	X	X
MaxResultSetSize	X	X
MaxTempTableSize	X	X
MaxValRange		X

\* Support for this policy was removed from Windows Server 2003 and later.

Windows Server 2008 supports the same set of **LDAP** policies as Windows Server 2003.

LDAP policies are specified using the [LDAPAdminLimits](#) attribute. The [LDAPAdminLimits](#) attribute of a [queryPolicy](#) object is a multivalued string where each string value encodes a name-value pair. In the encoding, the name and value are separated by an "=". For example, the encoding of the name "MaxActiveQueries" with value "0" is "MaxActiveQueries=0". Each name is the name of an **LDAP** policy, and the value is a value of that policy.

There can be multiple [queryPolicy](#) objects in a forest. A DC determines the [queryPolicy](#) object that contains its policies according to the following logic:

- If the [queryPolicyObject](#) attribute is present on the DC's [nTDSDSA](#) object, the DC uses the [queryPolicy](#) object referenced by it.
- Otherwise, if the [queryPolicyObject](#) attribute is present on the [nTDSsiteSettings](#) object for the **site** to which the DC belongs, the DC uses the [queryPolicy](#) object referenced by it.
- Otherwise, the DC uses the [queryPolicy](#) object whose DN is "CN=Default Query Policy,CN=Query-Policies" relative to the [nTDSservice](#) object (for example, "CN=Default Query Policy, CN=Query-Policies, CN=Directory Service, CN=Windows NT, CN=Services" relative to the root of the config NC).

The effect of setting an **LDAP** policy is outside the state model. The effect of each policy, as well as the default value used if the policy's value is not specified in an [LDAPAdminLimits](#) attribute, is shown in the following table.

The meaning of each policy, as well as the default value used if the policy's value is not specified, is shown in the following table.

Policy name	Default value	Description
MaxActiveQueries	20	The maximum number of concurrent <b>LDAP</b> search operations that are permitted to run at the same time on a DC. When this limit is reached, the DC returns a <i>busy</i> error ( <a href="#">[RFC2251]</a> section 4.1.10).
InitRecvTimeout	120	The maximum time in seconds that a DC waits for the client to send the first request after the DC receives a new connection. If the client does not send the first request in this amount of time, the server disconnects the client.
MaxConnections	5000	The maximum number of simultaneous <b>LDAP</b> connections that a DC will accept. If a connection comes in after the DC reaches



Policy name	Default value	Description
		this limit, the DC will drop another connection.
MaxConnIdleTime	900	The maximum time in seconds that the client can be idle before the DC closes the connection. If a connection is idle for more than this time, the DC disconnects the client.
MaxDatagramRecv	4096	The maximum size in bytes of a UDP datagram request that a DC will process. Requests that are larger than this value are ignored by the DC.
MaxNotificationPerConn	5	The maximum number of outstanding notification search requests (using the LDAP_SERVER_NOTIFICATION_OID control) that the DC permits on a single connection. When this limit is reached the server returns a <i>busy</i> error ( <a href="#">[RFC2251]</a> section 4.1.10) to any new notification searches that are requested on that connection.
MaxPoolThreads	4	The maximum number of threads per processor that a DC dedicates to listening for network input or output. This value also determines the maximum number of threads per processor that can work on <b>LDAP</b> requests at the same time.
MaxReceiveBuffer	10,485,760	The maximum size, in bytes, of a request that the server will accept. If the server receives a request that is larger than this, it will drop the connection.
MaxPageSize	1000	The maximum number of objects that are returned in a single search result, independent of how large each returned object is. To perform a search where the result might exceed this number of objects, the client must specify the paged search control.
MaxQueryDuration	120	The maximum time in seconds that a DC will spend on a single search. When this limit is reached, the DC returns a <i>timeLimitExceeded</i> error ( <a href="#">[RFC2251]</a> section 4.1.10).
MaxResultSetSize	262,144	The maximum number of bytes that a DC stores to optimize the individual searches that make up a paged search.
MaxTempTableSize	10,000	The maximum number of rows that a DC will create in a temporary database table to hold intermediate results during query processing.
MaxValRange	1500	The maximum number of values that can be retrieved from a multivalued attribute in a single search request. Windows 2000 DCs do not support this policy and instead always use a setting of 1000 values.

#### 3.1.1.3.4.7 LDAP Configurable Settings

A forest supports several administrator-controlled settings that affect **LDAP**. The name of each setting is included in the [supportedConfigurableSettings](#) attribute on the rootDSE. These settings are listed in the table below. The table also lists which versions of Windows Server support which settings. The settings are stored on the [msDS-Other-Settings](#) attribute of the directory service object, as specified in section [7.1.1.2.4.1.1](#). For more information see [\[ADDLG\]](#).

Setting name	Windows 2000	Windows Server 2003	Windows Server 2003 SP1	Windows Server 2008 AD/DS	Windows Server 2008 AD/LDS
DynamicObjectDefaultTTL		X	X	X	X
DynamicObjectMinTTL		X	X	X	X
DisableVLVSupport			X	X	X
ADAMAllowADAMSecurityPrincipalsInConfigPartition					X
ADAMDisableLogonAuditing					X
ADAMDisablePasswordPolicies					X
ADAMDisableSPNRegistration					X
ADAMDisableSSI					X
ADAMLastLogonTimestampWindow					X
MaxReferrals				X	X
ReferralRefreshInterval				X	X
RequireSecureProxyBind					X
RequireSecureSimpleBind					X
SelfReferralsOnly				X	X

The DynamicObjectDefaultTTL is the default [entryTTL](#) value for a new dynamic object. The value is in seconds, and defaults to 86400. The minimum value is 1 and the maximum value is 31557600 (one year).

The DynamicObjectMinTTL is the minimum valid [entryTTL](#) value for a new dynamic object. The value is in seconds, and defaults to 900. The minimum value is 1 and the maximum value is 31557600 (one year).

When the DisableVLVSupport setting is set to 1, the DC excludes the OIDs for the LDAP\_CONTROL\_VLVREQUEST and LDAP\_CONTROL\_VLVRESPONSE controls from the [supportedControl](#) attribute of the rootDSE. Additionally, if the LDAP\_CONTROL\_VLVREQUEST control is attached to an incoming **LDAP** request and is not marked as critical, the DC ignores the control. If the control is attached to an incoming **LDAP** request and is marked critical, the DC fails the request with the error *unavailableCriticalExtension* ([\[RFC2251\]](#) section 4.1.10). If the DisableVLVSupport setting is not specified, it defaults to 0.

When ADAMAllowADAMSecurityPrincipalsInConfigPartition equals 1, **security principals** (that is, objects that have an [objectSid](#) attribute) may be created in the Config NC. When equal to 0, attempts to create a security principal in the Config NC are rejected with the error *unwillingToPerform* ([\[RFC2251\]](#) section 4.1.10). If ADAMAllowADAMSecurityPrincipalsInConfigPartition is not specified, it defaults to 0.

The effect of ADAMDisableLogonAuditing is outside the state model. When ADAMDisableLogonAuditing equals 1, the DC does not generate audit events when an AD/LDS security principal (section [5.1.1.5](#)) authenticates to the server. If set to 0, the DC attempts to generate audit events when a AD/LDS security principal authenticates to the server; policy on the computer running the DC determines whether audit events are actually generated. If ADAMDisableLogonAuditing is not specified, it defaults to 0.

When ADAMDisablePasswordPolicies does not equal 1 and an **LDAP** bind is performed or a password is changed on an AD/LDS security principal, the DC enforces the current password policy in effect on the AD/LDS server as reported by SamrValidatePassword ([MS-SAMR] section [3.2.5.13.8](#)). When ADAMDisablePasswordPolicies is set to 1, the DC does not enforce any such policies. If ADAMDisablePasswordPolicies is not explicitly specified, it defaults to 0.

When ADAMDisableSPNRegistration equals 1, a DC running as AD/LDS does not register its SPNs on the [servicePrincipalName](#) of the [computer](#) object as described in [MS-DRSR] section [2.2.2](#). When ADAMDisableSPNRegistration equals 0, a DC running as AD/LDS performs **SPN** registration as described in that document. If ADAMDisableSPNRegistration is not explicitly specified, it defaults to 0.

When ADAMDisableSSI equals 1, a DC running as AD/LDS does not support DIGEST-MD5 authentication for AD/LDS **security principals**. If ADAMDisableSSI equals 0, a DC running as AD/LDS supports DIGEST-MD5 for AD/LDS **security principals**. ADAMDisableSSI has no effect on a DC running as AD/DS. If ADAMDisableSSI is not explicitly specified, it defaults to 0.

ADAMLastLogonTimestampWindow specifies how frequently, in days, AD/LDS updates the [lastLogonTimestamp](#) attribute when an AD/LDS **security principal** (see section [5.1.1.5](#)) authenticates to the server. For an AD/LDS security principal O, if a successful **LDAP** bind as that security principal is performed at time T, and the difference between O![lastLogonTimestamp](#) and T is greater than ADAMLastLogonTimestampWindow days, then the AD/LDS DC sets O![lastLogonTimestamp](#) to T. Otherwise, the AD/LDS DC leaves O![lastLogonTimestamp](#) unchanged. If ADAMLastLogonTimestampWindow is not explicitly specified, it defaults to 7.

MaxReferrals specifies the maximum number of **LDAP** URLs that the DC will include in a referral or continuation reference. The default value is 3.

The effect of ReferralRefreshInterval is outside the state model. A Windows DC maintains an in-memory cache of referral information so that it can return referrals and continuation references without consulting the directory state. ReferralRefreshInterval specifies how frequently, in minutes, a DC refreshes the in-memory cache from the directory state. The default value is 5.

When RequireSecureProxyBind is set to 1, AD/LDS will reject (with the error *confidentialityRequired* [\[RFC2251\]](#) section 4.1.10) an **LDAP** simple bind request that requests **Authentication** as an

AD/LDS bind proxy (section [5.1.1.5](#)) if that request is not performed on a SSL/TLS-encrypted or SASL-protected connection with a cipher strength of at least 128 bits. If RequireSecureProxyBind is set to 0, no such restriction is imposed. If RequireSecureProxyBind is not explicitly specified, it defaults to 1.

When RequireSecureSimpleBind is set to 1, AD/LDS will reject (with the error *confidentialityRequired* [\[RFC2251\]](#) section 4.1.10) an **LDAP** simple bind request that requests **Authentication** as an AD/LDS security principal (section [5.1.1.5](#)) if that request is not performed on a SSL/TLS-encrypted or SASL-protected connection with a cipher strength of at least 128 bits. If RequireSecureSimpleBind is set to 0, no such restriction is imposed. If RequireSecureProxyBind is not explicitly specified, it defaults to 0.

If SelfReferralsOnly is set to 1, then the DC will only return referrals and continuation references that refer to itself. It will not return referrals and continuation references to NCs of which it does not have an NC replica. Referrals and continuation references to NCs of which it does have an NC replica will name itself as the referred-to server.

### 3.1.1.3.4.8 LDAP IP-Deny List

The IP Deny list specifies a set of IP addresses from which the DC will reject incoming **LDAP** connection requests. The IP Deny list is stored in the [IDAPIPDenyList](#) attribute on the [queryPolicy](#) object. The DC retrieves the [IDAPIPDenyList](#) attribute from the same [queryPolicy](#) object that it retrieves the [IDAPAdminLimits](#) attribute from in section [3.1.1.3.4.6](#)

The [IDAPIPDenyList](#) attribute is a multivalued attribute. Each value of the attribute is a string in the following form:

**X.X.X.X M.M.M.M**

where **X.X.X.X** is an IP address and **M.M.M.M** is a network mask. A connection from an IP address Y.Y.Y.Y will be rejected if the bitwise-and of Y.Y.Y.Y and **M.M.M.M** equals **X.X.X.X**.

For example, the value "157.59.132.0 255.255.255.0" would cause requests from IP addresses 157.59.132.0 through 157.59.132.255 to be rejected. The value "157.59.132.245 255.255.255.255" would reject only IP address 157.59.132.245.

The IP Deny list is only supported on IP v4 connections. Active Directory does not support this mechanism on IP v6 connections.

### 3.1.1.4 Reads

#### References:

- Glossary: AD, **Access Check**, Attribute, Auxiliary class, Auxiliary object class, Canonical name, Child object, Children, **constructed attribute**, **Container**, [crossRef](#) object, DC, Distinguished name, DN, **Domain** controller, **Domain NC**, **Global group**, [governsID](#), [group](#) object, **LDAP**, **Most Specific object class**, Naming context, NC, NC, replica, [nTDSDSA](#) object, Object, Object class, **Object class name**, **Object of Class x (or x Object)**, Object reference, [objectClass](#), [objectGUID](#), [objectSid](#), OID, Parent object, Property set, Read permission, Replica, RID, rootDSE, Schema, Schema container, Schema NC, **SD**, Security identifier, Security Context, Security Descriptor, Security principal object, SID, Structural class, Structural object class, Tombstone, Universal group
- [\[RFC2251\]](#)
- Special Objects and Forest Invariants: section [7.1](#)

- [\[MS-DRSR\]](#)
- [\[XMLSCHEMA2/2\]](#)
- Quota invariants: section [3.1.1.5](#)
- Range Retrieval of Attribute Values: section [3.1.1.3](#)
- Referrals in **LDAP** v2 and v3: section [3.1.1.3](#)
- [\[MS-ADSC\]](#)
- [\[MS-ADA1\]](#)
- [\[MS-ADA2\]](#)
- [\[MS-ADA3\]](#)
- Function GetWellknownObject: section [3.1.1.1](#)

#### 3.1.1.4.1 Introduction

LDAP reads are specified in [\[RFC2251\]](#) section 4.5. Generally and imprecisely, reads are searches starting at some object in AD and restricted by the requestor to either the object, the object's children, or the tree of objects rooted by object. After applying that restriction, the search is then restricted to the objects and the values for attributes on those objects to which the requestor has access. The search is finally restricted to the objects that match the search filter. The requested attributes and their values for those matching objects are then returned to the requestor. The RFC specifies the details for **LDAP** reads. This section covers access checks for **LDAP** reads, extended access checks for reading the specified attributes, the attributes used to construct the specified **constructed attributes**, and the effect of defunct attributes and classes on reads.

This section does not provide details on the classes and attributes mentioned here. For details, see [\[MS-ADSC\]](#), [\[MS-ADA1\]](#), [\[MS-ADA2\]](#), and [\[MS-ADA3\]](#).

#### 3.1.1.4.2 Definitions

The following functions are used to specify the behavior of several of the **constructed attributes**. They are collected together here in one place because of the dependencies they have on each other.

Let SUPCLASSES ([top](#)) be the empty set. For other classes O, let SUPCLASSES(O) be the union of O![subClassOf](#) and SUPCLASSES(O![subClassOf](#)).

Let AUXCLASSES(O) be the union of

O![systemAuxiliaryClass](#)

and O![auxiliaryClass](#)

and AUXCLASSES(O![systemAuxiliaryClass](#))

and AUXCLASSES(O![auxiliaryClass](#))

and AUXCLASSES(SUPCLASSES(O))

Let POSSSUPERIORS(O) be the union of

O![systemPossSuperiors](#)

and O![possSuperiors](#)

and POSSSUPERIORS (SUPCLASSES(O))

and POSSSUPERIORS (AUXCLASSES(O))

Let CLASSATTS(O) be the union of

O![mustContain](#)

and O![systemMustContain](#)

and O![mayContain](#)

and O![systemMayContain](#)

and CLASSATTS(SUPCLASSES(O))

and CLASSATTS(AUXCLASSES(O))

Let SPC(O) be true when O or any SUPCLASSES(O) is one of [builtinDomain](#), [samServer](#), [samDomain](#), [group](#), or [user](#); and false, otherwise.

### 3.1.1.4.3 Access Checks

An object is not visible to a requestor if the requestor is not granted the necessary rights. But even if an object is visible to a requestor, the requestor may lack the necessary rights to see individual attributes. The values for attributes that are not visible to the requestor are treated as "does not exist" in the returned attributes and the **LDAP** filter. For example, if the requestor requests the value for [displayName](#) but that attribute is not visible, then the returned value will be the same as it would have been if the attribute [displayName](#) did not exist on that Object. Likewise, if [displayName](#) were part of the **LDAP** filter, then, similarly, the filter would behave just as if [displayName](#) did not exist on that Object.

Let O be the Object being considered during search.

Let ON be the root object of the NC containing O.

Let OP be O![parent](#).

Let OA be the Attribute, or the property set containing the Attribute, being considered for O during search.

Generally, the security context of the requestor must be granted rights RIGHT\_DS\_LIST\_CONTENTS (defined in section [5.1.3.2](#)) on OP by OP![nTSecurityDescriptor](#).

Generally, the security context of the requestor must be granted rights RIGHT\_DS\_READ\_PROPERTY on OA by O![nTSecurityDescriptor](#). Otherwise, the value is treated as "does not exist" in the returned attributes and the **LDAP** filter. This behavior changes for special attributes, for attributes with special search flags in their definition, and for some attributes because of [dSHeuristics](#) (section [7.1.1.2.4.1.2](#)), as specified in section [3.1.1.4.4](#).

### 3.1.1.4.4 Extended Access Checks

Some attributes require different access than that specified in the previous section.

The security context of the requestor must be granted the indicated rights on OA by O![nTSecurityDescriptor](#) unless otherwise specified. If not granted, then the value is treated as "does not exist" in the returned attributes and the **LDAP** filter:

OA	Requires Right(s)
<a href="#">nTSecurityDescriptor</a>	(ACCESS_SYSTEM_SECURITY) and (READ_CONTROL)
<a href="#">msDS-QuotaEffective</a>	(RIGHT_DS_READ_PROPERTY on the Quotas container, described in section <a href="#">7.1.1.4.3</a> ) or ((the client is querying the quota for the security principal it is authenticated as) and (DS-Query-Self-Quota <b>control access right</b> on the Quotas container))
<a href="#">msDS-QuotaUsed</a>	(RIGHT_DS_READ_PROPERTY on the Quotas container, described in section <a href="#">7.1.1.4.3</a> ) or ((the client is querying the quota for the security principal it is authenticated as) and (DS-Query-Self-Quota <b>control access right</b> on the Quotas container))
<a href="#">userPassword</a>	When <a href="#">dSHeuristics.fUserPwdSupport</a> is false, RIGHT_DS_READ_PROPERTY. When <a href="#">dSHeuristics.fUserPwdSupport</a> is true, access is never granted.
<a href="#">pekList</a>	Access is never granted
<a href="#">currentValue</a>	Access is never granted
<a href="#">dBCSPwd</a>	Access is never granted
<a href="#">unicodePwd</a>	Access is never granted
<a href="#">ntPwdHistory</a>	Access is never granted
<a href="#">priorValue</a>	Access is never granted
<a href="#">supplementalCredentials</a>	Access is never granted
<a href="#">trustAuthIncoming</a>	Access is never granted
<a href="#">trustAuthOutgoing</a>	Access is never granted
<a href="#">lmPwdHistory</a>	Access is never granted
<a href="#">initialAuthIncoming</a>	Access is never granted
<a href="#">initialAuthOutgoing</a>	Access is never granted
<a href="#">msDS-ExecuteScriptPassword</a>	Access is never granted
Attribute whose <a href="#">attributeSchema</a> has 0x80 set in <a href="#">searchFlags</a> .	(RIGHT_DS_READ_PROPERTY) and (RIGHT_DS_CONTROL_ACCESS)
<a href="#">sDRightsEffective</a>	See section <a href="#">3.1.1.4.5.4</a>

OA	Requires Right(s)
<a href="#">allowedChildClassesEffective</a>	See section <a href="#">3.1.1.4.5.5</a>
<a href="#">allowedAttributesEffective</a>	See section <a href="#">3.1.1.4.5.7</a>
<a href="#">msDS-Approx-Immed-Subordinates</a>	See section <a href="#">3.1.1.4.5.15</a>
<a href="#">msDS-QuotaEffective</a>	See section <a href="#">3.1.1.4.5.22</a>
<a href="#">msDS-ReplAttributeMetaData</a> <a href="#">msDS-ReplValueMetaData</a>	The security context of the requestor must be granted the following rights on the <a href="#">replPropertyMetaData</a> attribute: (RIGHT_DS_READ_PROPERTY) or (RIGHT_DS_REPL_MANAGE_TOPOLOGY by ON! <a href="#">nTSecurityDescriptor</a> )
<a href="#">msDS-NCReplInboundNeighbors</a>	The security context of the requestor must be granted the following rights on <a href="#">repsFrom</a> : (RIGHT_DS_READ_PROPERTY) or (RIGHT_DS_REPL_MANAGE_TOPOLOGY) or (RIGHT_DS_REPL_MONITOR_TOPOLOGY)
<a href="#">msDS-NCReplOutboundNeighbors</a>	The security context of the requestor must be granted the following rights on <a href="#">repsTo</a> : (RIGHT_DS_READ_PROPERTY) or (RIGHT_DS_REPL_MANAGE_TOPOLOGY) or (RIGHT_DS_REPL_MONITOR_TOPOLOGY)
<a href="#">msDS-NCReplCursors</a>	The security context of the requestor must be granted the following rights on <a href="#">replUpToDateVector</a> : (RIGHT_DS_READ_PROPERTY) or (RIGHT_DS_REPL_MANAGE_TOPOLOGY) or (RIGHT_DS_REPL_MONITOR_TOPOLOGY)
<a href="#">msDS-IsUserCachableAtRodc</a>	The security context of the requestor must be granted the DS-Replication-Secrets-Synchronize <b>control access right</b> on the root of the default NC.

### 3.1.1.4.5 Constructed Attributes

The details about individual **constructed attributes** are specified in [\[MS-ADA1\]](#), [\[MS-ADA2\]](#), and [\[MS-ADA3\]](#). But briefly, **constructed attributes** are attributes whose value is constructed by using other attributes, sometimes from other objects. The objects and attributes for specified **constructed attributes** are covered in this section.

Except as otherwise noted, these constructed attributes are applicable to both AD/DS and AD/LDS.

#### 3.1.1.4.5.1 subSchemaSubEntry

The value is the DN equal to the schema NC's DN appended to "CN=Aggregate,".

#### 3.1.1.4.5.2 canonicalName

The value is the canonical name of the object (section [3.1.1.1.7](#)).



### 3.1.1.4.5.3 allowedChildClasses

Let TO be the object from which the [allowedChildClasses](#) attribute is being read.

The value of TO![allowedChildClasses](#) is the set of [LDAPDisplayName](#)s read from each Object O where

- (O.[distinguishedName](#) is in the schema NC)
- and (O![objectClass](#) is [classSchema](#))
- and (not O![systemOnly](#))
- and (not O![objectClassCategory](#) is 2)
- and (not O![objectClassCategory](#) is 3)
- and (there exists C in TO![objectClass](#) such that C is in POSSSUPERIORS(O))

### 3.1.1.4.5.4 sDRightsEffective

Let TO be the object from which the [sDRightsEffective](#) attribute is being read.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	S	D	G	O
																												S	S	S	S
																												I	I	I	I

**Note** Bits are presented in big-endian byte order.

The value of TO![sDRightsEffective](#) is derived as follows from the bits shown in the preceding table:

- OSI (OWNER\_SECURITY\_INFORMATION) and GSI (GROUP\_SECURITY\_INFORMATION) are both set if TO![nTSecurityDescriptor](#) grants RIGHT\_WRITE\_OWNER to the requestor.
- DSI (DACL\_SECURITY\_INFORMATION) is set if TO![nTSecurityDescriptor](#) grants RIGHT\_WRITE\_DAC to the requestor.
- SSI (SACL\_SECURITY\_INFORMATION) is set if TO![nTSecurityDescriptor](#) grants RIGHT\_ACCESS\_SYSTEM\_SECURITY to the requestor.

### 3.1.1.4.5.5 allowedChildClassesEffective

The [allowedChildClassesEffective](#) attribute has different behavior on AD/DS and AD/LDS.

If the DC is running as AD/LDS, then let fAllowPrincipals equal true if the value of the ADAMAllowADAMSecurityPrincipalsInConfigPartition configuration setting (section [3.1.1.3.4.7](#)) is 1, false otherwise. If the DC is running as AD/DS, then let fAllowPrincipals = false.

Let TO be the object from which the [allowedChildClassesEffective](#) attribute is being read.

TO![allowedChildClassesEffective](#) contains each object class O in TO![allowedChildClasses](#) such that:

- (

(TO![nTSecurityDescriptor](#) grants RIGHT\_DS\_CREATE\_CHILD via a simple ACE to the client for instantiating an object beneath TO)

or

(TO.[nTSecurityDescriptor](#) grants RIGHT\_DS\_CREATE\_CHILD via an object-specific ACE to the client for instantiating an object of class O beneath TO)

)

- and (fAllowPrincipals or (not TO![distinguishedName](#) in config NC) or (not SPC(O)))
- and (fAllowPrincipals or (not TO![distinguishedName](#) in schema NC) or (not SPC(O)))

Simple ACEs and object-specific ACEs are discussed in section [5.1.3](#).

#### **3.1.1.4.5.6 allowedAttributes**

Let TO be the object from which the [allowedAttributes](#) attribute is being read.

The value of TO![allowedAttributes](#) is the set of [IDAPDisplayNames](#) read from each Object O where:

- (O.dn is in the schema NC)
- and (O![objectClass](#) is [attributeSchema](#))
- and (O![linkID](#) is even or O![linkID](#) is not present)
- and (not bit 0x4 is set in O![systemFlags](#))
- and (there exists C in TO![objectClass](#) such that O is in CLASSATTS(C))

#### **3.1.1.4.5.7 allowedAttributesEffective**

Let TO be the object from which the [allowedAttributesEffective](#) attribute is being read.

The value of TO![allowedAttributesEffective](#) is the subset of values returned by [allowedAttributes](#) for which:

- TO![nTSecurityDescriptor](#) grants RIGHT\_DS\_WRITE\_PROPERTY to the requestor.

#### **3.1.1.4.5.8 fromEntry**

Let TO be the object from which the [fromEntry](#) attribute is being read.

The value of TO![fromEntry](#) is true if TO![instanceType](#) has bit 0x4 set, otherwise false.

#### **3.1.1.4.5.9 createTimeStamp**

Let TO be the object from which the createTimeStamp attribute is being read.

The value of TO![createTimeStamp](#) is TO![whenCreated](#).

#### **3.1.1.4.5.10 modifyTimeStamp**

Let TO be the object from which the modifyTimeStamp attribute is being read.

The value of TO![modifyTimeStamp](#) is TO![whenChanged](#).

#### 3.1.1.4.5.11 primaryGroupToken

Let TO be the object from which the [primaryGroupToken](#) attribute is being read.

The value of TO![primaryGroupToken](#) is the RID from TO![objectSid](#) when there exists C in TO![objectClass](#) such that C is the [group](#) class. Otherwise, no value is returned. That is, if TO is a group, then the value of this attribute is the RID from the group's SID. If TO is not a group, no value is returned when this attribute is read from TO.

#### 3.1.1.4.5.12 entryTTL

Let TO be the object from which the [entryTTL](#) attribute is being read.

The value of TO![entryTTL](#) is the number of seconds in TO![msDS-Entry-Time-To-Die](#) minus the current system time, and is constrained to the range 0..0xFFFFFFFF by returning 0 if the difference is less than 0 and 0xFFFFFFFF if the difference is greater than 0xFFFFFFFF.

#### 3.1.1.4.5.13 msDS-NCReplInboundNeighbors, msDS-NCReplCursors, msDS-ReplAttributeMetaData, msDS-ReplValueMetaData

If the object from which [msDS-NCReplInboundNeighbors](#) or [msDS-NCReplCursors](#) is being read is not the root object of an NC, the result of the read is no value.

Otherwise, reading any of these four attributes on an object returns an alternate representation of the structures returned by IDL\_DRSGetReplInfo() applied to that object. The result is either a binary data structure or XML (IDL\_DRSGetReplInfo and its associated structures are documented in [\[MS-DRSR\]](#) section 4.1.13). The relationship between these **constructed attributes** and the IDL\_DRSGetReplInfo data is shown in the table below.

Constructed Attribute	Equivalent DS_REPL_INFO Code (see <a href="#">[MS-DRSR]</a> section 4.1.13.1.4)	XML Structure (see <a href="#">[MS-DRSR]</a> section 4.1.13.1)	Binary Structure (see section <a href="#">2.2</a> )
<a href="#">msDS-NCReplInboundNeighbors</a>	DS_REPL_INFO_NEIGHBORS	DS_REPL_NEIGHBORW	DS_REPL_NEIGHBORW_BLOB
<a href="#">msDS-NCReplCursors</a>	DS_REPL_INFO_CURSORS_3_FOR_NC	DS_REPL_CURSORS_3W	DS_REPL_CURSOR_BLOB
<a href="#">msDS-ReplAttributeMetaData</a>	DS_REPL_INFO_METADATA_2_FOR_OBJ	DS_REPL_ATTR_METADATA_2	DS_REPL_ATTR_METADATA_BLOB
<a href="#">msDS-ReplValueMetaData</a>	DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE	DS_REPL_VALUE_METADATA_2	DS_REPL_VALUE_METADATA_BLOB

The information returned is exactly the same information as is returned by a call to IDL\_DRSGetReplInfo when specifying the value in the second column as the value for DRS\_MSG\_GETREPLINFO\_REQ\_V1.InfoType or DRS\_MSG\_GETREPLINFO\_REQ\_V2.InfoType.

Without any attribute qualifier, the data is returned as XML. The parent element of the XML is the name of the structure contained in the "XML Structure" column in the table, and the child element

names and order in the XML exactly follow the names of the fields in that structure as well. The meaning of each child element is the same as the meaning of the corresponding field in the structure. Values of integer types are represented as decimal strings. Values of FILETIME type are represented as XML dateTime values in UTC, for example, "04-07T18:39:09Z", as specified in [XMLSCHEMA2/2]. Values of GUID fields are represented as GUIDStrings.

If the ";binary" attribute qualifier is specified when the attribute is requested, the value of this attribute is returned as binary data, specifically, the structure contained in the "Binary Structure" column. In this representation, fields that would contain strings are represented as integer offsets (relative to the beginning of the binary data) to a null-terminated UTF-16 encoded string embedded in the returned binary data.

#### **3.1.1.4.5.14 msDS-NCReplOutboundNeighbors**

The [msDS-NCReplOutboundNeighbors](#) attribute is equivalent to [msDS-NCReplInboundNeighbors](#), except that it retrieves representations of each [repsTo](#) value for the requested Object (that is, information related to replication notifications for event-driven replication), while [msDS-NCReplInboundNeighbors](#) retrieves representations of each [repsFrom](#) value (that is, information related to inbound replication). Like [msDS-NCReplInboundNeighbors](#), it can return the data in either XML or binary form, depending on the presence of the ";binary" attribute qualifier, and uses the DS\_REPL\_NEIGHBOR and DS\_REPL\_NEIGHBORW\_BLOB structures for its XML and binary representations, respectively.

#### **3.1.1.4.5.15 msDS-Approx-Immed-Subordinates**

Let TO be the object from which the [msDS-Approx-Immed-Subordinates](#) attribute is being read.

The value of TO![msDS-Approx-Immed-Subordinates](#) is the approximate number of Objects contained by this object if TO![ntSecurityDescriptor](#) grants RIGHT\_DS\_LIST\_CONTENTS to the client. This estimate has no guarantee or requirement of accuracy. If the client does not have the RIGHT\_DS\_LIST\_CONTENTS access right, the value 0 is returned as the estimate.

#### **3.1.1.4.5.16 msDS-KeyVersionNumber**

The msDS-KeyVersionNumber attribute exists on AD/DS but not on AD/LDS.

Let TO be the object from which the [msDS-KeyVersionNumber](#) attribute is being read.

If the fKVNOEmuW2k dsHeuristic (section [7.1.1.2.4.1.2](#)) is true, TO![msDS-KeyVersionNumber](#) equals 1. Otherwise, TO![msDS-KeyVersionNumber](#) equals the dwVersion field of the **AttributeStamp** associated with TO's [unicodePwd](#) attribute. See section [3.1.1.1.9](#) for more information about **AttributeStamp** and dwVersion.

#### **3.1.1.4.5.17 msDS-User-Account-Control-Computed**

The [msDS-User-Account-Control-Computed](#) attribute has different behavior on AD/DS and AD/LDS.

Let TO be the object from which the [msDS-User-Account-Control-Computed](#) attribute is being read.

For AD/DS, the following description applies:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	L	0	0	0	0
								E																		O					

**Note** Bits are presented in big-endian byte order.

If the object TO is not in a **domain** NC, TO![msDS-User-Account-Control-Computed](#) = 0.

If the object TO is in a **domain** NC, let D be the root of that NC, and let ST be the current time, read from the system clock. Then the value of TO![msDS-User-Account-Control-Computed](#) is the bit pattern above where:

- LO (ADS\_UF\_LOCKOUT) is set if:
  - (none of bits ADS\_UF\_WORKSTATION\_TRUST\_ACCOUNT, ADS\_UF\_SERVER\_TRUST\_ACCOUNT, ADS\_UF\_INTERDOMAIN\_TRUST\_ACCOUNT are set in TO![userAccountControl](#))
  - and (TO![lockoutTime](#) is non-zero and either (1) Effective-LockoutDuration (regarded as an unsigned quantity) < 0x8000000000000000, or (2) ST + Effective-LockoutDuration (regarded as a signed quantity) ≤ TO![lockoutTime](#) ), where Effective-LockoutDuration is defined in [MS-SAMR] section [3.2.1.5](#).
- PE (ADS\_UF\_PASSWORD\_EXPIRED) is set if:
  - (none of bits ADS\_UF\_SMARTCARD\_REQUIRED, ADS\_UF\_DONT\_EXPIRE\_PASSWD, ADS\_UF\_WORKSTATION\_TRUST\_ACCOUNT, ADS\_UF\_SERVER\_TRUST\_ACCOUNT, ADS\_UF\_INTERDOMAIN\_TRUST\_ACCOUNT are set in TO![userAccountControl](#))
  - and (TO![pwdLastSet](#) = null, or TO![pwdLastSet](#) = 0, or (Effective-MaximumPasswordAge ≠ 0x8000000000000000 and (ST - TO![pwdLastSet](#)) > Effective-MaximumPasswordAge)), where Effective-MaximumPasswordAge is defined in [MS-SAMR] section [3.2.1.5](#).

For AD/LDS, the following description applies:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	P	0	0	0	0	0	0	D	0	0	0	0	0	0	0	0	0	0	P	L	0	0	A	0
								E							E										R	O				D	

**Note** Bits are presented in big-endian byte order.

The value of TO![msDS-User-Account-Control-Computed](#) attribute is the bit pattern above where:

- AD (ADS\_UF\_ACCOUNT\_DISABLE) is set if:
  - TO![msDS-UserAccountDisabled](#) is true
- LO (ADS\_UF\_LOCKOUT) is set if:

- TO![ms-DS-UserAccountAutoLocked](#) is true
- PNR (ADS\_UF\_PASSWD\_NOTREQD) is set if:
  - TO![ms-DS-UserPasswordNotRequired](#) is true
- DEP (ADS\_UF\_DONT\_EXPIRE\_PASSWD) is set if:
  - TO![msDS-UserDontExpirePassword](#) is true
- PE (ADS\_UF\_PASSWORD\_EXPIRED) is set if:
  - TO![msDS-UserPasswordExpired](#) is true

#### 3.1.1.4.5.18 msDS-Auxiliary-Classes

Let TO be the object from which the [msDS-Auxiliary-Classes](#) attribute is being read.

The value of TO![msDS-Auxiliary-Classes](#) is the set of [IDAPDisplayName](#)s from each Object O such that (O is in TO![objectClass](#)) and (O is not in SUPCLASSES(Most Specific class of TO)).

#### 3.1.1.4.5.19 tokenGroups, tokenGroupsNoGCAcceptable

The [tokenGroups](#) attribute exists on both AD/DS and AD/LDS. The [tokenGroupsNoGCAcceptable](#) attribute exists on AD/DS but not on AD/LDS.

These two computed attributes return the set of SIDs from a transitive group membership expansion operation on a given object.

For AD/DS, the [tokenGroups](#) attribute has no value if no GC server is available to evaluate the transitive reverse memberships. The [tokenGroupsNoGCAcceptable](#) attribute can always be retrieved, but if no GC server is available the set of SIDs may be incomplete.

Let U be the object from which the [tokenGroups](#) or [tokenGroupsNoGCAcceptable](#) attribute is being read.

- If U![objectSid](#) does not exist, U![tokenGroups](#) and U![tokenGroupsNoGCAcceptable](#) are not present.
- Otherwise, U![tokenGroups](#) and U![tokenGroupsNoGCAcceptable](#) are the result of the algorithm in [MS-DRSR] section [4.1.8.3](#) (IDL\_DRSGetMemberships) using DRS\_MSG\_REVMEMB\_REQ\_V1.OperationType=RevMembGetGroupsForUser, DRS\_MSG\_REVMEMB\_REQ\_V1.ppDsNames=U, and DRS\_MSG\_REVMEMB\_REQ\_V1.pLimitingDomain = the domain for which the server is a DC.

#### 3.1.1.4.5.20 tokenGroupsGlobalAndUniversal

The [tokenGroupsGlobalAndUniversal](#) attribute exists on AD/DS but not on AD/LDS.

This computed attribute returns the set of SIDs of global and universal groups resulting from a transitive group membership expansion operation on a given object. This attribute has no value if no GC server is available to evaluate the transitive reverse memberships.

Let U be the object from which the [tokenGroupsGlobalAndUniversal](#) attribute is being read.

- If U![objectSid](#) does not exist, U![tokenGroupsGlobalAndUniversal](#) is not present.

- Otherwise let S be the set of SIDs returned by invoking the algorithm in [MS-DRSR] section [4.1.8.3](#) (IDL\_DRSGetMemberships) using DRS\_MSG\_REVMEMB\_REQ\_V1.OperationType=RevMembGetAccountGroups, DRS\_MSG\_REVMEMB\_REQ\_V1.ppDsNames=U, and DRS\_MSG\_REVMEMB\_REQ\_V1.pLimitingDomain = the domain for which the server is a DC.
- Let accumulator set T be the Null set
- For each SID s in S
  - Let X be the set of SIDs returned by invoking the algorithm in [MS-DRSR] section 4.1.8.3 (IDL\_DRSGetMemberships) using DRS\_MSG\_REVMEMB\_REQ\_V1.OperationType=RevMembGetUniversalGroups, DRS\_MSG\_REVMEMB\_REQ\_V1.ppDsNames=s, and DRS\_MSG\_REVMEMB\_REQ\_V1.pLimitingDomain = NULL.
  - T = T union X
- U![tokenGroupsGlobalAndUniversal](#) is the union of T and S

#### 3.1.1.4.5.21 possibleInferiors

Let TO be the object from which the [possibleInferiors](#) attribute is being read.

Let C be the [classSchema](#) object corresponding to TO![governsID](#).

The value of TO![possibleInferiors](#) is the set of O![governsID](#) for each Object O where

- (O is in the schema NC)
- and (O![objectClass](#) is [classSchema](#))
- and (not O![systemOnly](#))
- and (not O![objectClassCategory](#) is 2)
- and (not O![objectClassCategory](#) is 3)
- and ((C is contained in POSSSUPERIORS(O))

#### 3.1.1.4.5.22 msDS-QuotaEffective

Let TO be the object from which the [msDS-QuotaEffective](#) attribute is being read.

Let R be the root object of the NC containing TO.

Let SID be the sid specified by the **LDAP** extended control LDAP\_SERVER\_QUOTA\_CONTROL\_OID or, if none is specified, the requestor's SID.

Let SIDS be the set of SIDs including SID and the set of SIDs returned by [tokenGroups](#).

The value of TO![msDS-QuotaEffective](#) is the maximum of all O![msDS-QuotaAmount](#) for each object O where:

- (TO is the object:

**GetWellknownObject**(n: R, guid: GUID\_NTDS\_QUOTAS\_CONTAINER\_W))

- and (O is a child of TO)
- and (the client has access to O as specified in section [3.1.1.4.3](#))
- and (the client has access to O![msDS-QuotaAmount](#) as specified in section [3.1.1.4.3](#))
- and (the client has access to O![msDS-QuotaTrustee](#) as specified in section [3.1.1.4.3](#))
- and (there exists S in SIDS such that S is equal to O![msDS-QuotaTrustee](#))

#### 3.1.1.4.5.23 msDS-QuotaUsed

Let TO be the object from which the [msDS-QuotaUsed](#) attribute is being read.

Let C be the Most Specific Class from TO![objectClass](#).

Let R be the root object of the NC containing TO.

Let SID be the sid specified by the **LDAP** extended control LDAP\_SERVER\_QUOTA\_CONTROL\_OID or, if none is specified, the requestor's SID.

The value of TO![msDS-QuotaUsed](#) is:

- $(cLive + ((cTombstoned * TO![msDS-TombstoneQuotaFactor](#)) + 99) / 100)$

where:

- cLive is the number of non-tombstoned Objects associated with SID and cTombstoned is the number of Tombstoned Objects associated with SID as detailed in Quota invariants in section [3.1.1.5](#).

when:

- (TO is the object:

**GetWellknownObject**(n: R, guid: GUID\_NTDS\_QUOTAS\_CONTAINER\_W))

#### 3.1.1.4.5.24 msDS-TopQuotaUsage

Let TO be the object from which the [msDS-TopQuotaUsage](#) attribute is being read.

Let R be the root object of the NC containing TO.

TO![msDS-TopQuotaUsage](#) equals a set of XML encoded strings sorted by the element quotaUsed when:

- TO is the object:

**GetWellknownObject**(n: R, guid: GUID\_NTDS\_QUOTAS\_CONTAINER\_W)

Each string represents the quota information for a SID as specified in section Quota invariants in section [3.1.1.5](#). The format of the XML encoded string is:

**<MS\_DS\_TOP\_QUOTA\_USAGE>**

**<partitionDN>** DN of the NC containing TO **</partitionDN>**

**<ownerSID>** SID of quota user **</ownerSID>**



**<quotaUsed>** rounded up value of quota used (computed) **</quotaUsed>**

**<tombstoneCount>** value in the TombstoneCount column **</tombstoneCount>**

**<totalCount>** value in the TotalCount column **</totalCount>**

**</MS\_DS\_TOP\_QUOTA\_USAGE>**

Where quotaUsed is computed as specified in [msDS-QuotaUsed](#) with cLive set to (totalCount - tombstoneCount).

The number of values returned can be controlled with the ";range" syntax as detailed in (see Range Retrieval of Attribute Values in section [3.1.1.3](#)). The default range is 10 for this attribute.

#### 3.1.1.4.5.25 ms-DS-UserAccountAutoLocked

The [ms-DS-UserAccountAutoLocked](#) attribute exists on AD/LDS but not on AD/DS.

Let TO be the object from which the [ms-DS-UserAccountAutoLocked](#) attribute is being read. Let ST be the current time, read from the system clock.

If the machine running AD/LDS is joined to a **domain** D, TO![ms-DS-UserAccountAutoLocked](#) is true if both of the following are true:

- The **LDAP** configurable setting ADAMDisablePasswordPolicies  $\neq$  1.
- TO![lockoutTime](#)  $\neq$  0 and either (1) D![lockoutDuration](#) (regarded as an unsigned quantity)  $<$  0x8000000000000000, or (2) ST + D![lockoutDuration](#) (regarded as a signed quantity)  $\leq$  TO![lockoutTime](#).

If the machine running AD/LDS is not joined to a **domain**, TO![ms-DS-UserAccountAutoLocked](#) is true if both of the following are true:

- The **LDAP** configurable setting ADAMDisablePasswordPolicies  $\neq$  1.
- TO![lockoutTime](#)  $\neq$  0 and (current time - TO![lockoutTime](#))  $\leq$  X, where X is determined by the policy of the machine on which AD/LDS is running,

#### 3.1.1.4.5.26 msDS-UserPasswordExpired

The [msDS-UserPasswordExpired](#) attribute exists on AD/LDS but not on AD/DS.

Let TO be the object from which the [msDS-UserPasswordExpired](#) attribute is being read. Let ST be the current time, read from the system clock.

If the machine running AD/LDS is joined to a **domain**, let D be the root of the **domain** NC of the joined **domain**. Then TO![msDS-UserPasswordExpired](#) is true if all of the following are true:

- The **LDAP** configurable setting ADAMDisablePasswordPolicies  $\neq$  1.
- None of bits ADS\_UF\_SMARTCARD\_REQUIRED, ADS\_UF\_DONT\_EXPIRE\_PASSWD, ADS\_UF\_WORKSTATION\_TRUST\_ACCOUNT, ADS\_UF\_SERVER\_TRUST\_ACCOUNT, ADS\_UF\_INTERDOMAIN\_TRUST\_ACCOUNT is set in TO![userAccountControl](#).
- TO![pwdLastSet](#) = null, or TO![pwdLastSet](#) = 0, or (D![maxPwdAge](#)  $\neq$  0x8000000000000000 and (ST - TO![pwdLastSet](#))  $>$  D![maxPwdAge](#))).

If the machine running AD/LDS is not joined to a **domain**, then TO![msDS-UserPasswordExpired](#) is true if all of the following are true:

- The **LDAP** configurable setting ADAMDisablePasswordPolicies  $\neq$  1.
- None of bits ADS\_UF\_SMARTCARD\_REQUIRED, ADS\_UF\_DONT\_EXPIRE\_PASSWD, ADS\_UF\_WORKSTATION\_TRUST\_ACCOUNT, ADS\_UF\_SERVER\_TRUST\_ACCOUNT, ADS\_UF\_INTERDOMAIN\_TRUST\_ACCOUNT is set in TO![userAccountControl](#).
- TO![pwdLastSet](#) = null, or TO![pwdLastSet](#) = 0, or (ST - TO![pwdLastSet](#)) > X, where X is determined by the policy of the machine on which AD/LDS is running.

#### 3.1.1.4.5.27 msDS-PrincipalName

The [msDS-PrincipalName](#) attribute has different behavior on AD/DS and AD/LDS.

Let TO be the object from which the [msDS-PrincipalName](#) attribute is being read.

For AD/DS, the value of TO![msDS-PrincipalName](#) is either (1) the NetBIOS domain name, followed by a backslash ('\'), followed by TO![sAMAccountName](#), or (2) the value of TO![objectSid](#) in SDDL SID string format ([MS-DTYP] sections [2.4.6](#) and [2.5.1](#)).

For AD/LDS, let OBJSID be the value of TO![objectSid](#). If OBJSID is the SID of a security principal of the computer on which Active Directory is running, then TO![msDS-PrincipalName](#) is the NetBIOS computer name, followed by a backslash ('\'), followed by the name of the security principal. If the computer on which Active Directory is running is a member of a **domain** and OBJSID is a SID for a security principal S in that **domain**, then TO![msDS-PrincipalName](#) is the NetBIOS domain name, followed by a backslash ('\'), followed by S![sAMAccountName](#). Otherwise, the value of TO![msDS-PrincipalName](#) is the value of TO![objectSid](#) in SDDL SID string format ([MS-DTYP] sections [2.4.6](#) and [2.5.1](#)).

#### 3.1.1.4.5.28 parentGUID

This attribute is not present on an object that is the root of an NC. For all other objects, let TO be the object from which the [parentGUID](#) attribute is being read and let TP be TO![parent](#). TO![parentGUID](#) is equal to TP![objectGUID](#).

#### 3.1.1.4.5.29 msDS-SiteName

The [msDS-SiteName](#) attribute exists on AD/DS but not on AD/LDS.

Let TO be the object on which [msDS-SiteName](#) is being read. If TO is an [nTDSDSA](#) object or a [server](#) object, then TO![msDS-SiteName](#) is equal to the value of the RDN of the [site object](#) under which TO is located. For example, given a TO that is an [nTDSDSA](#) object with the DN "CN=NTDS Settings, CN=TESTDC-01, CN=Servers, CN=Default-First-Site-Name, CN=Sites, CN=Configuration, DC=fabrikam, DC=com", the value of TO![msDS-SiteName](#) is "Default-First-Site-Name".

If TO is a [computer](#) object, then let TS be the [server](#) object named by TO![serverReferenceBL](#). TO![msDS-SiteName](#) equals TS![msDS-SiteName](#).

If TO is neither a [computer](#), [server](#), nor [nTDSDSA](#) object, then TO![msDS-SiteName](#) has no value.

#### 3.1.1.4.5.30 msDS-isRODC

The [msDS-isRODC](#) attribute exists on AD/DS but not on AD/LDS.

This attribute indicates whether a specified DC is an RODC. Let TO be the object on which [msDS-isRODC](#) is being read. If TO is not a [nTDSDSA](#), [computer](#), or [server](#) object, then TO![msDS-isRODC](#) has no value.

- If TO is a [nTDSDSA](#) object:
  - If TO![objectCategory](#) equals the DN of the [classSchema](#) object for the [nTDSDSA](#) object class, then TO![msDS-isRODC](#) is false. Otherwise, TO![msDS-isRODC](#) is true.
- If TO is a [server](#) object:
  - Let TN be the [nTDSDSA](#) object whose DN is "CN=NTDS Settings," prepended to the DN of TO. Apply the previous rule for the "TO is an [nTDSDSA](#) object" case, substituting TN for TO.
- If TO is a [computer](#) object:
  - Let TS be the [server](#) object named by TO![serverReferenceBL](#). Apply the previous rule for the "TO is a [server](#) object" case, substituting TS for TO.

#### 3.1.1.4.5.31 msDS-isGC

The [msDS-isGC](#) attribute exists on AD/DS but not on AD/LDS.

This attribute indicates whether a specified DC is a GC server. Let TO be the object on which [msDS-isGC](#) is being read. If TO is not a [nTDSDSA](#), [computer](#), or [server](#) object, then TO.[msDS-isGC](#) has no value.

- If TO is a [nTDSDSA](#) object:
  - TO![msDS-isGC](#) iff TO![options](#) has the NTDSDSA\_OPT\_IS\_GC bit set (section [7.1.1.2.2.1.2.1.1](#)).
- If TO is a [server](#) object:
  - Let TN be the [nTDSDSA](#) object whose DN is "CN=NTDS Settings," prepended to the DN of TO. Apply the previous rule for the "TO is an [nTDSDSA](#) object" case, substituting TN for TO.
- If TO is a [computer](#) object:
  - Let TS be the [server](#) object named by TO![serverReferenceBL](#). Apply the previous rule for the "TO is a [server](#) object" case, substituting TS for TO.

#### 3.1.1.4.5.32 msDS-isUserCachableAtRodc

The [msDS-IsUserCachableAtRodc](#) attribute exists on AD/DS but not on AD/LDS.

This attribute indicates whether a specified RODC is permitted by administrator policy to cache the secrets of a specified security principal. The DN of the security principal is specified using the LDAP Control LDAP\_SERVER\_INPUT\_DN\_OID. The DN specified may be either an RFC 2253-style DN, or one of the alternate DN formats specified in section [3.1.1.3.1.2.4](#).

Let TO be the object on which [msDS-IsUserCachableAtRodc](#) is being read. If TO is not a [nTDSDSA](#), [computer](#), or [server](#) object, then TO![msDS-IsUserCachableAtRodc](#) has no value.

- If TO is a [computer](#) object:
  - If TO![userAccountControl](#) does not have the ADS\_UF\_PARTIAL\_SECRETS\_ACCOUNT bit set, TO![msDS-IsUserCachableAtRodc](#) has no value.

- If TO![userAccountControl](#) has the UF\_PARTIAL\_SECRETS bit set the value of TO![msDS-IsUserCachableAtRode](#) is calculated as follows:
  - Let D be the DN of the user principal specified using LDAP Control LDAP\_SERVER\_INPUT\_DN\_OID. If the DN of a security principal is not explicitly specified, D is the DN of the current requestor.
  - TO![msDS-IsUserCachableAtRode](#) = GetRevealSecretsPolicyForUser(TO![distinguishedName](#), D) (procedure GetRevealSecretsPolicyForUser is defined in [MS-DRSR] section [4.1.10.5.13](#)).
- If TO is a [server](#) object:
  - Let TC be the [computer](#) object named by TO![serverReference](#). Apply the previous rule for the "TO is a [computer](#) object" case, substituting TC for TO.
- If TO is a [nTDSDSA](#) object:
  - Let TS be the [server](#) object that is the parent of TO. Apply the previous rule for the "TO is a [server](#) object" case, substituting TS for TO.

### 3.1.1.4.5.33 msDS-UserPasswordExpiryTimeComputed

The [msDS-UserPasswordExpiryTimeComputed](#) attribute exists on AD/DS but not on AD/LDS.

This attribute indicates the time when the password of the object will expire. Let TO be the object on which the attribute [msDS-UserPasswordExpiryTimeComputed](#) is read. If TO is not in a **domain** NC, then TO![msDS-UserPasswordExpiryTimeComputed](#) = null. Otherwise let D be the root of the **domain** NC containing TO. The DC applies the following rules, in the order specified below, to determine the value of TO![msDS-UserPasswordExpiryTimeComputed](#):

- If any of the ADS\_UF\_SMARTCARD\_REQUIRED, ADS\_UF\_DONT\_EXPIRE\_PASSWD, ADS\_UF\_WORKSTATION\_TRUST\_ACCOUNT, ADS\_UF\_SERVER\_TRUST\_ACCOUNT, ADS\_UF\_INTERDOMAIN\_TRUST\_ACCOUNT bits is set in TO![userAccountControl](#), then TO![msDS-UserPasswordExpiryTimeComputed](#) = 0x7FFFFFFFFFFFFFFF.
- Else, if TO![pwdLastSet](#) = null, or TO![pwdLastSet](#) = 0, then TO![msDS-UserPasswordExpiryTimeComputed](#) = 0.
- Else, if Effective-MaximumPasswordAge = 0x8000000000000000, then TO![msDS-UserPasswordExpiryTimeComputed](#) = 0x7FFFFFFFFFFFFFFF (where Effective-MaximumPasswordAge is defined in [MS-SAMR] section [3.2.1.5](#)).
- Else, TO![msDS-UserPasswordExpiryTimeComputed](#) = TO![pwdLastSet](#) + Effective-MaximumPasswordAge (where Effective-MaximumPasswordAge is defined in [MS-SAMR] section [3.2.1.5](#)).

### 3.1.1.4.5.34 msDS-RevealedList

The [msDS-RevealedList](#) attribute exists on AD/DS (starting with Windows Server 2008) but not on AD/LDS.

The [msDS-RevealedList](#) attribute exists only on the [computer](#) object of an RODC. The value of [msDS-RevealedList](#) is a multi-valued DN-String. The string portion of each value is the [LDAPDisplayName](#) of a secret attribute, and the DN portion of each value names an object. Each value represents the presence of a value for the named attribute on the named object on the RODC, i.e. the value has been "revealed" to the RODC.

The [msDS-RevealedList](#) attribute is constructed from the [msDS-RevealedUsers](#) attribute as follows:

Let *O* be the object from which the [msDS-RevealedList](#) attribute is being read.

Let *RESULT* be a set of DN-String, initially empty.

For each *V* (a DN-Binary) in *O*![msDS-RevealedUsers](#) do

- Let *USER* be the object with DN *V.object\_DN*.
- Let *P* (a PROPERTY\_META\_DATA, see [MS-DRSR] section [4.1.10.2.18](#)) equal *V.binary\_value*.
- Let *SCH* equal SchemaObj(*P.attrType*) ([MS-DRSR] section [5:SchemaObj](#)).
- Let *RV* be a DN-String with *RV.string\_value* equal *SCH*![IDAPDisplayName](#) and *RV.object\_DN* equal *V.object\_DN*.
- Let *A* be *SCH*![IDAPDisplayName](#).
- If AttributeStampCompare(*P.propMetadataExt*, AttrStamp(*USER*, *P.attrType*)) = 0, set *RESULT* = *RESULT* + {*RV*}. (See [MS-DRSR] section [4.1.10.3.5](#) for procedure AttributeStampCompare, and [MS-DRSR] section [5:AttrStamp](#) for procedure AttrStamp.)

Return the set *RESULT* (if empty, the [msDS-RevealedList](#) attribute is not present).

#### 3.1.1.4.5.35 msDS-RevealedListBL

The [msDS-RevealedListBL](#) attribute exists on AD/DS (starting with Windows Server 2008) but not on AD/LDS.

This attribute behaves precisely like a back link attribute for the [msDS-RevealedList](#) **constructed attributes** described in the previous section.

Therefore the [msDS-RevealedList](#) attribute exists only on a [user](#) object, one or more of whose secrets have been "revealed" to an RODC. The value is the set of RODCs (represented by their [computer](#) objects) to which one or more of the given [user](#) object's secrets have been revealed.

#### 3.1.1.4.5.36 msDS-ResultantPSO

The [msDS-ResultantPSO](#) attribute exists on AD/DS beginning with Windows Server 2008. This attribute does not exist on AD/LDS.

The value of [msDS-ResultantPSO](#) is a single value of Object (DS-DN) syntax. This attribute is constructed as follows:

Let *RESULTSET* be a set of DS-DN, initially empty.

Let *U* be the object from which the [msDS-ResultantPSO](#) attribute is being read.

- If the domain functional level is less than DS\_BEHAVIOR\_WIN2008, then there is no value in this attribute.
- If *U*![objectClass](#) does not contain the value "user", then there is no value in this attribute.
- If the bit for ADS\_UF\_NORMAL\_ACCOUNT (see section [2.2.15](#)) is not set in *U*![userAccountControl](#), then there is no value in this attribute.

- If the RID in *U!*[objectSid](#) is equal to DOMAIN\_USER\_RID\_KRBTGT, then there is no value in this attribute.
- If the *U!*[msDS-SecondaryKrbTgtNumber](#) attribute has a value, then there is no value in this attribute.
- Let RESULTSET be the values of *U!*[msDS-PSOApplied](#) that are of object class [msDS-PasswordSettings](#) and are under the Password Settings container (see section [7.1.1.4.10.1](#))
- If RESULTSET is empty:
  - Let *S* be the set of objects returned by invoking the algorithm in [MS-DRSR] section 4.1.8.3 (IDL\_DRSGetMemberships) using  
 DRS\_MSG\_REVMEMB\_REQ\_V1.OperationType=RevMembGetAccountGroups,  
 DRS\_MSG\_REVMEMB\_REQ\_V1.ppDsNames=U, and  
 DRS\_MSG\_REVMEMB\_REQ\_V1.pLimitingDomain = the domain for which the server is a DC.
  - For each *O* (an object) in *S* do
    - *RESULTSET* = *RESULTSET* union *O!*[msDS-PSOApplied](#)
- Sort objects in set *RESULTSET* according to [msDS-PasswordSettingsPrecedence](#) values, breaking ties with [objectGUID](#) values, with smaller values coming first.
- Return the first element in the sorted *RESULTSET* (if empty, the [msDS-ResultantPSO](#) attribute is not present).

#### 3.1.1.4.6 Referrals

When the server returns a referral as documented in section [3.1.1.3.1.4](#), it must determine which server(s) to refer the client to. The set of servers to which the client will be referred is the set of values returned by the following algorithm.

Let *N* be the DSNAME of the base of the ldap search.

Let *NSID* be the sid portion of *N*.

Let *NGUID* be the guid portion of *N*.

Let *NSTR* be the dn portion of *N*.

The value is:

- (the values of *O!*[dnsRoot](#) for the object *O* where:
  - (*NSTR* is not present)
  - and (*NGUID* is not present)
  - and (*NSID* is present)
  - and ((*O!*[nCName](#))![objectSid](#) matches the **domain** sid from *NSID*)
  - and (*O!*[parent](#) is the Partitions Container)
  - and (*O!*[objectClass](#)'s Most Specific class is [crossRef](#))
  - and (*O!*[Enabled](#) is true))

- and (the value for Root-Domain-NC![dnsRoot](#) after prepending "gc.\_msdcs." and either replacing the first matching ".\*" with ":3268" or, if there are no matches of ".\*", then by appending ":3268" when:
  - (NSTR is not present)
  - and (NGUID is present))
- and (the values of O![dnsRoot](#) for the object O where:
  - (NSTR is present)
  - and (O![nCName](#) is a prefix for NSTR and is the longest prefix among all O satisfying these conditions)
  - and (O![parent](#) is the Partitions Container)
  - and (O![objectClass](#)'s Most Specific Object Class is [crossRef](#))
  - and (O![Enabled](#) is true))
- and (the value is Root-Domain-NC![superiorDNSRoot](#) when:
  - (NSTR is present)
  - and (Root-Domain-NC![superiorDNSRoot](#) is present)
  - and (there exists no object O such that
    - ((O![nCName](#) is a prefix for NSTR)
    - and (O![parent](#) is the Partitions Container)
    - and (O![objectClass](#)'s most specific class is [crossRef](#))
    - and (O![Enabled](#) is true)))
- and (the value is the transform of TO.dn into a dotted string by concatenating the value for the first dc component with values for subsequent components separated by "." (for example, CN=bob,DC=One,DC=Two is transformed into One.Two) when:
  - ((NSTR is present)
  - and (Root-Domain-NC![superiorDNSRoot](#) is not present)
  - and (there exists no object O such that
    - ((O![nCName](#) is a prefix for NSTR)
    - and (O![parent](#) is the Partitions Container)
    - and (O![objectClass](#)'s most specific class is [crossRef](#))
    - and (O![Enabled](#) is true))))

#### 3.1.1.4.7 Continuations

When the server returns a continuation reference as documented in section [3.1.1.3.1.4](#), it must determine which server(s) to refer the client to. The set of servers to which the client will be referred is the set of values returned by the following algorithm.

Let TO be the base object of an **LDAP** Search.

Let NC be the NC-replica containing TO

The values are made up of:

- The values from O![dnsRoot](#) for all objects O where
  - (O.dn is listed in NC![subRefs](#))
  - and (O![Enabled](#) is true)
  - and (O![objectClass](#)'s most specific class is [crossRef](#))
  - and
    - (((O![nCName](#) is a prefix of TO.dn for all but the first component)  
and (the scope of the search is LDAP\_SCOPE\_ONELEVEL))
    - or
    - ((O![nCName](#) is a prefix of TO.dn)  
and (the scope of the search is LDAP\_SCOPE\_SUBTREE)))
- and the value for Root-Domain-NC![dnsRoot](#) after prepending "gc.\_msdcs." and either replacing the first matching ":\*" with ":3268" or, if there are no matches of ":", then by appending ":3268" if and only if:
  - (TO![objectClass](#)'s Most Specific Object Class is [addressBookContainer](#))  
and (the scope of the search is LDAP\_SCOPE\_ONELEVEL)

#### 3.1.1.4.8 Effects of Defunct Attributes and Classes

If the forest functional level is less than DS\_BEHAVIOR\_WIN2003, a search that mentions a defunct class or attribute succeeds just as if the class or attribute were not defunct.

If the forest functional level is DS\_BEHAVIOR\_WIN2003 or greater:

- Instances of a defunct attribute cannot be read.
- Instances of a defunct class can be read using the filter term (objectClass=\*)
- When reading any OID-valued attribute that contains identifiers for schema objects, if the attribute identifies a defunct schema object, the read returns an OID (the [attributeID](#) if an attribute, the [governsID](#) if a class) not a name (the [LDAPDisplayName](#) of an attribute or class). This behavior applies to the [objectClass](#) attribute of normal objects, and to the [mustContain](#), [systemMustContain](#), [mayContain](#), [systemMayContain](#), [subClassOf](#), [auxiliaryClass](#), and [possSuperiors](#) attributes of schema objects.



### 3.1.1.5 Updates

#### 3.1.1.5.1 General

##### References

- LDAP attributes: [fSMORoleOwner](#), [invocationId](#), [objectGUID](#).
- State model attributes: [rdnType](#).
- LDAP classes: [classSchema](#), [crossRef](#), [nTDSDSA](#), [rIDManager](#).
- Glossary: config NC, default NC, dsname, NC replica, replicated attribute, schema NC.
- LDAP errors: [\[RFC2251\]](#) section 4.1.10.
- Abstract attribute [repsTo](#): see [MS-DRSR] section [5:repsTo](#).
- IDL\_DRSReplicaSync method: see [MS-DRSR] section [4.1.23](#).
- DRS\_MSG\_REPSYNC message: see [MS-DRSR] section [4.1.23.1.1](#).
- DRS\_MSG\_REPSYNC\_V1 message: see [MS-DRSR] section [4.1.23.1.2](#).
- Urgent replication specified by SAM: see [MS-SAMR] section [3.2.1.8](#).

This section specifies operations that are common for all originating update and replicated update operations. An update could be an Add, Modify, Modify DN, or Delete operation.

When an update results in an error, the error is expressed below in the form:

- *LDAP error*

or:

- *LDAP error / Windows error*

See section [3.1.1.3.1.9](#) for the specification of how Windows errors are returned in an **LDAP** response.

##### 3.1.1.5.1.1 Enforce Schema Constraints

The originating update is validated for schema constraints as explained in Restrictions on Schema Extensions in section [3.1.1.2](#). Schema constraints are not enforced for replicated updates.

##### 3.1.1.5.1.2 Naming Constraints

During originating update of Add, Modify and Modify DN operation, the server validates the following naming constraints. The server returns **LDAP** error *namingViolation* if a naming constraint is not met.

- The RDN must not contain a character with value 0xA or 0x0.
- The DN must be compliant with RFC 2253.
- The RDN size must be less than 255 characters.

Naming constraints are not enforced for replicated updates.

### 3.1.1.5.1.3 Transactional Semantics

The effects of an originating update are captured in the state model by committing a transaction before sending the response. The transaction has the ACID properties [GRAY] and provides at least degree 2 isolation of concurrent read and update requests [GRAY].

Transactions used to implement Active Directory provide degree 2 isolation of concurrent read and update requests.

Each Search request or Update request is performed as a transaction. When multiple Search requests are used to retrieve a large set of results, each request is its own transaction. An originating update is processed as one or more transactions. In some cases a request will cause transactions to occur after the response has been sent. The remainder of section 3.1.1.5 specifies the transaction boundaries used for all originating updates and describes all cases where processing continues after the response.

### 3.1.1.5.1.4 Stamp Construction

Stamps for replicated attributes and link values will be updated for each originating update as defined in section 3.1.1.1. When applying replicated updates, stamps are constructed as described in [MS-DRSR] section 4.1.10.6.

### 3.1.1.5.1.5 Replication Notification

Each NC replica on the server has an associated abstract attribute [repsTo](#). When an originating or replicated update occurs in the NC replica on the server, the server notifies each destination DC that has an entry in [repsTo](#). The server notifies the destination DC by calling method IDL\_DRSReplicaSync. The destination DC contacts the server and requests it to provide updates - this is event-driven replication as described in section 3.1.1.1.14.

The server sends replication notifications as follows:

Let  $N$  be the NC replica where the originating or replicated update has occurred on the server.

For each  $i$  in  $[0 .. (N!\text{repsTo}).length-1]$  do the following:

- Let  $E$  be  $N!\text{repsTo}[i]$ .
- Let  $C$  be the [crossRef](#) object corresponding to  $N$ .
- Let  $pmsgIn$  be a reference to structure of type DRS\_MSG\_REPSYNC.
- Set  $pmsgIn \rightarrow V1.pNC$  to dsname of  $N$ .
- Let  $O$  be the [nTDSDSA](#) object of the server.
- Set  $pmsgIn \rightarrow V1.uuidDsaSrc$  to  $O!\text{objectGUID}$ .
- Set  $pmsgIn \rightarrow V1.ulOptions$  to  $(\text{DRS\_ASync\_OP} \mid \text{DRS\_UPDATE\_NOTIFICATION})$ .
- If  $(E.replicaFlags \ \& \ \text{DRS\_WRITE\_REP} \neq 0)$  then set  $pmsgIn \rightarrow V1.ulOptions$  to  $(pmsgIn \rightarrow V1.ulOptions \mid \text{DRS\_WRITE\_REP})$ .
- If the originating/replicated update satisfies the condition for urgent replication then set  $pmsgIn \rightarrow V1.ulOptions$  to  $(pmsgIn \rightarrow V1.ulOptions \mid \text{DRS\_SYNC\_URGENT})$ .
- Let  $H$  be the handle obtained by calling IDL\_DRSBind against  $E.uuidDsa$ .

- If (*pmsgIn* → *V1.ulOptions* & DRS\_SYNC\_URGENT = 0), then wait for an implementation-specific time *T*. If *i* = 0 the default time *T* is 15 seconds; if *i* > 0 the default time *T* is 3 seconds.
- Let *R* be the result of calling IDL\_DRSReplicaSync(*H*, 1, *pmsgIn*).
- Let *Z* be the current time.
- If *E.timeLastAttempt* > *Z* or *Z.timeLastAttempt* - *Z* > an implementation-specific duration *U*, update *N!repsTo*[*i*] as follows:
  - Set *E.timeLastAttempt* to *Z*.
  - Set *E.ulResultLastAttempt* to *R*.
  - If *R* = 0, set *E.timeLastSuccess* to *Z* and set *E.cConsecutiveFailures* to 0.
  - If *R* ≠ 0, increment *E.cConsecutiveFailures* by 1.
  - Set *N!repsTo*[*i*] to *E*.

The default duration *U* is one hour.

### 3.1.1.5.1.6 Urgent Replication

Let *N* be the NC replica on the server. There are few originating/replicated updates in *N* that need to be replicated immediately to each destination DC that has an entry in *N!repsTo*. Updates that need to be replicated immediately are listed below:

- Creation of [nTDSDSA](#) object.
- Creation of [crossRef](#) object.
- Updates to schema object ([attributeSchema](#) or [classSchema](#)).
- Deletion of [nTDSDSA](#) object.
- Deletion of [crossRef](#) object.
- Update to [secret](#) object.
- Update to [rIDManager](#) object.
- Update to [lockoutTime](#), [pwdLastSet](#), and [userAccountControl](#) attributes as specified in [MS-SAMR] section [3.2.1.8](#).

The server behavior for urgent replication is specified in section [3.1.1.5.1.5](#).

### 3.1.1.5.1.7 Updates Performed Only on FSMOs

Certain originating update operations in AD must be performed on a single master. For example, all schema updates must happen on the schema master FSMO DC; creation and deletion of crossRef objects representing naming contexts must happen on the **domain** naming FSMO DC. If the operation is attempted on a DC that does not hold the FSMO role, then it issues a referral to the current FSMO owner. The following section describes how such updates are handled. The processing is not performed when applying replicated updates.

The following types and functions are used in specifying the FSMO-related processing of originating update.

The function `IsEffectiveRoleOwner(roleObject:object)` verifies that the current DC is the valid owner of the given FSMO role. The FSMO ownership is considered valid if a successful replication of the corresponding NC occurred with some replication partner. This function is defined below.

For a given FSMO role, the function `RoleUpdateScope(roleObject:Object)` returns the set of objects and their attributes that can only be updated on the FSMO owner DC. For example, for `FSMO_SCHEMA`, the set contains all objects residing within Schema NC, with all of their attributes. The function is defined below.

Define variable *timeLastReboot* equal to the time when the server last rebooted.

Define function `IsEffectiveRoleOwner (roleObject: object)` returns Boolean as follows.

- Let *S* be the [nTDSDSA](#) object of the server.
- If  $S \neq \text{roleObject}!\text{fSMORoleOwner}$  then return false.
- Let *N* be the NC containing roleObject.
- If there exists at least one entry *E* in  $N!\text{repsFrom}$  such that  $E.\text{timeLastSuccess} > \text{timeLastReboot}$  then return true.
- Otherwise return false.

Let `RoleType` be the enumeration ( `SchemaMasterRole`, `PartitionNamingMasterRole`, `InfrastructureMasterRole`, `RidAllocationMasterRole`, `PdcEmulationMasterRole` ).

Define function `RoleObject(n: NC, roleType: RoleType)` returns object as follows.

- If `roleType = SchemaMasterRole`,
  - if *n* = Schema NC, return *n* otherwise return null
- If `roleType = PartitionNamingMasterRole`,
  - if *n* = Config NC, return Partition Container of *n* otherwise return null
- If `roleType = InfrastructureMasterRole`,
  - if *n* = Default NC (AD/DS), return Infrastructure Container of *n*, otherwise return null
- If `roleType = RidAllocationMasterRole`,
  - if *n* = Default NC (AD/DS), return RID Manager Container of *n*, otherwise return null
- If `roleType = PdcEmulationMasterRole`,
  - if *n* = Default NC (AD/DS), return *n*, otherwise return null
- Otherwise return null

Define function `RoleUpdateScope(roleObject: object)` returns set *S* as follows. *S* is a set such that each element is an object and a list of attributes associated with the object.

- Let *n* be the NC containing roleObject.
- Let *roleType* be role corresponding to the roleObject, i.e.  $\text{RoleObject}(n, \text{roleType}) = \text{roleObject}$ .
- If *roleType* = `FSMO_SCHEMA`:

- The set of all objects and all attributes in the *roleObject*'s NC.
- If *roleType* = FSMO\_DOMAIN\_NAMING, the union of
  - *roleObject* and all attributes
  - The objects that are children of *roleObject* and all attributes.
- If *roleType* = FSMO\_INFRASTRUCTURE, the union of
  - *roleObject* and all attributes.
  - The Updates container *u* of *roleObject*'s NC and all attributes.
  - The objects that are children *u* and all attributes.
- If *roleType* = FSMO\_RID, the union of
  - *roleObject* and all attributes.
  - Let *I* = GetWellKnownObject(*n*, GUID\_INFRASTRUCTURE\_CONTAINER).
  - All children *C* of *I* and all attributes, such that *C*![objectClass](#) contains [infrastructureUpdate](#) and *C*![proxiedObjectName](#) is present.
    - If *C* is the [computer](#) object for the DC requesting the FSMO operation, *C* and all attributes.
    - The DC's [rIDSet](#) object.
- If *roleType* = FSMO\_PDC,
  - *roleObject* and all attributes.
  - *n* with attributes [wellKnownObjects](#) and [msDS-Behavior-Version](#).
- Otherwise return NULL

Given those preliminaries, the following processing is performed on each object *O* on which an originating update is being made.

Let *O.A* be the attribute that is being updated.

Let *N* be the NC containing *O*.

For each RoleType *T* do the following:

- Let *R* = RoleObject(*N*, *T*)
- If *R* exists then
  - Let *S* = RoleUpdateScope(*R*)
  - If *O* is not an element of {*S*} or *O.A* is not an element of {*S*} then proceed with the originating update operation.
  - If *R*![fSMORoleOwner](#) ≠ distinguished name of the *nTDSDSA* object of the server, then let *K* = (*R*![fSMORoleOwner](#))![parent](#). Return **LDAP** referral error to *K*![dNSHostName](#).
  - If IsEffectiveRoleOwner(*R*) = true proceed with the originating update operation.

- Otherwise, return **LDAP** error *busy*.

#### 3.1.1.5.1.8 Originating Updates Attempted on an RODC

An RODC does not perform originating updates. When an originating update is requested on an RODC, the RODC generates an **LDAP** referral ([\[RFC2251\]](#) section 3.2 and section 4.1.1) to a DC holding a writable NC replica, as specified below. By following the referral the client can perform the desired update.

Define *O* as follows:

- If the originating update is an add, let *O* be the parent of the object to be added.
- If the originating update is a modify, modify DN, or delete, let *O* be the object to be updated.

If *O* does not exist return *noSuchObject*. Otherwise let *N* be the NC containing *O*. Using techniques described in section [7.3.6](#), find a DC *D* that has a writable NC replica for *N*. Generate an **LDAP** referral to *D* as specified in [\[RFC2251\]](#) section 4.1.1.

#### 3.1.1.5.1.9 Constraints and Processing Specifics Defined Elsewhere

In addition to the constraints and processing specifics defined in the remainder of section [3.1.1.5](#), update operations MUST conform to the constraints and processing details defined in [\[MS-SAMR\]](#) and [\[MS-DRSR\]](#). The constraints specified in [\[MS-SAMR\]](#) are enforced only for **originating updates**.

#### 3.1.1.5.2 Add Operation

##### References

LDAP attributes: [objectClass](#), [ntSecurityDescriptor](#), [instanceType](#), [distinguishedName](#), [objectGUID](#), [objectSid](#), [entryTTL](#), [msDS-Entry-Time-To-Die](#), [systemFlags](#), [msDS-AllowedToDelegateTo](#), [objectCategory](#), [defaultObjectCategory](#), [defaultHidingValue](#), [showInAdvancedViewOnly](#), [msDS-DefaultQuota](#), [msDS-QuotaTrustee](#), [msDS-TombstoneQuotaFactor](#), [subRefs](#), [nCName](#), [Enabled](#), [uSNLastObjRem](#), [uSNSALastObjRemoved](#), [whenCreated](#), [uSNCreated](#), [replPropertyMetaData](#), [isDeleted](#), [instanceType](#), [proxiedObjectName](#), [msDS-LockoutObservationWindow](#), [msDS-LockoutDuration](#), [msDS-MaximumPasswordAge](#), [msDS-MinimumPasswordAge](#), [msDS-MinimumPasswordLength](#), [msDS-PasswordHistoryLength](#).

LDAP classes: [dynamicObject](#), [crossRef](#), [trustedDomain](#), [secret](#), [classSchema](#), [attributeSchema](#), [msDS-QuotaControl](#), [foreignSecurityPrincipal](#).

##### Constants

- Win32/status error codes: `ERROR_DS_OBJ_CLASS_NOT_DEFINED`, `ERROR_DS_ILLEGAL_MOD_OPERATION`, `ERROR_DS_OBJECT_CLASS_REQUIRED`, `ERROR_DS_OBJ_CLASS_NOT_SUBCLASS`, `ERROR_DS_BAD_INSTANCE_TYPE`, `ERROR_DS_ADD_REPLICA_INHIBITED`, `ERROR_DS_CANT_ADD_SYSTEM_ONLY`, `ERROR_DS_CLASS_MUST_BE_CONCRETE`, `ERROR_DS_BAD_NAME_SYNTAX`, `ERROR_DS_ATT_NOT_DEF_IN_SCHEMA`, `ERROR_DS_NOT_SUPPORTED`, `ERROR_DS_RDN_DOESNT_MATCH_SCHEMA`, `ERROR_DS_BAD_NAME_SYNTAX`, `STATUS_QUOTA_EXCEEDED`, `ERROR_DS_REFERRAL`, `ERROR_DS_CROSS_REF_EXISTS`
- **Access mask** bits: `RIGHT_DS_CREATE_CHILD`, `RIGHT_DS_ADD_GUID`
- Security privileges: `SE_ENABLE_DELEGATION_PRIVILEGE`

- [instanceType](#) flags: IT\_NC\_HEAD, IT\_WRITE, IT\_NC\_ABOVE
- ObjectClassCategories: DS\_STRUCTURAL\_CLASS, DS\_88\_CLASS
- Generic [systemFlags](#) bits: FLAG\_CONFIG\_ALLOW\_RENAME, FLAG\_CONFIG\_ALLOW\_MOVE, FLAG\_CONFIG\_ALLOW\_LIMITED\_MOVE
- Schema [systemFlags](#) bits: FLAG\_ATTR\_IS\_RDN
- [crossRef systemFlags](#) bits: FLAG\_CR\_NTDS\_NC, FLAG\_CR\_NTDS\_DOMAIN, FLAG\_CR\_NTDS\_NOT\_GC\_REPLICATED

The add operation results in addition of a new object to the directory tree. The requestor supplies the following data:

- The DN of the new object.
- The set of attributes defining the new object.

### 3.1.1.5.2.1 Security Considerations

For regular object creation, the requestor must have RIGHT\_DS\_CREATE\_CHILD on the parent object for the [objectClass](#) of the object being added. The parent object must be visible to the requestor.

For application NC creation (see section [3.1.1.5.2.6](#)), the requestor must have sufficient permissions to create the [crossRef](#) object in the partitions **container** on the **domain** naming FSMO, or to take over an existing [crossRef](#) object (in case of pre-created [crossRef](#)). See section [3.1.1.5.2.6](#) below for more details.

If [msDS-AllowedToDelegateTo](#) attribute is specified as a part of the add operation, then the requestor must possess SE\_ENABLE\_DELEGATION\_PRIVILEGE.

Access checks are not performed for replicated updates.

### 3.1.1.5.2.2 Constraints

The following constraints are enforced for originating update add operations. If any of them are not satisfied, the server returns an error. The error is expressed below in the form:

*LDAP error / Windows error*

See section [3.1.1.3.1.9](#) for the specification of how Windows errors are returned in an **LDAP** response.

These constraints are not enforced for replicated updates.

- The object DN value is a syntactically valid DN (see LDAP, section [3.1.1.3](#)). If it is not, add returns *unwillingToPerform / ERROR\_DS\_BAD\_NAME\_SYNTAX*.
- If [instanceType](#) attribute value is specified, then the following constraints MUST be satisfied:
  - There is exactly one integer value, otherwise add returns *unwillingToPerform / ERROR\_DS\_BAD\_INSTANCE\_TYPE*.
  - If the [instanceType](#) value has IT\_NC\_HEAD bit set, then IT\_WRITE MUST be set. If this is the case, then this operation is considered to be an *NC-Add* operation, and additional constraints and processing specifics apply (see NC-Add below for details).

- If IT\_NC\_HEAD is set, but IT\_WRITE is not set, add returns *unwillingToPerform / ERROR\_DS\_ADD\_REPLICA\_INHIBITED*.
- If IT\_NC\_HEAD bit is not set in the value, then the only allowed values are zero and IT\_WRITE, otherwise add returns *unwillingToPerform / ERROR\_DS\_BAD\_INSTANCE\_TYPE*.
- If the operation is not NC-Add, then the parent object MUST be in an NC whose full replica is hosted at this DC, otherwise *referral / ERROR\_DS\_REFERRAL* is returned.
- If the operation is not NC-Add, then the parent object MUST be present in the directory and visible to the requestor (see section 5.1 for more information). The parent DN is computed from the passed in DN value by removing the first RDN label. If the parent object is not found in the directory, then *noSuchObject / ERROR\_DS\_OBJ\_NOT\_FOUND* is returned.
- At least one [objectClass](#) value MUST be specified. Otherwise, Add returns *unwillingToPerform / ERROR\_DS\_OBJ\_CLASS\_NOT\_DEFINED*.
- The [objectClass](#) attribute MUST be specified only once in the input attribute list. If more than one definition is found, Add returns *unwillingToPerform / ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- All [objectClass](#) values correspond to classes that are defined and active in the schema. If an unknown or defunct class is referenced, Add returns *unwillingToPerform / ERROR\_DS\_OBJ\_CLASS\_NOT\_DEFINED*.
- The set of non-auxiliary [objectClass](#) values defines a (possibly incomplete) inheritance chain with a single structural [objectClass](#). If this is not true, Add returns *unwillingToPerform / ERROR\_DS\_OBJ\_CLASS\_NOT\_SUBCLASS*.
- If the forest version is w2k3 or higher, then any number of auxiliary classes can be included. Otherwise, Add returns *unwillingToPerform / ERROR\_DS\_NOT\_SUPPORTED*.
- The structural [objectClass](#) is not marked as [systemOnly](#). If it is, add returns *unwillingToPerform / ERROR\_DS\_CANT\_ADD\_SYSTEM\_ONLY*.
- The structural [objectClass](#)'s classCategory is either DS\_STRUCTURAL\_CLASS or DS\_88\_CLASS. If it is not, add returns *unwillingToPerform / ERROR\_DS\_CLASS\_MUST\_BE\_CONCRETE*.
- The structural [objectClass](#) is not an LSA-specific object class (section 3.1.1.5.2.3). If it is, add returns *unwillingToPerform / ERROR\_DS\_CANT\_ADD\_SYSTEM\_ONLY*.
- If the structural [objectClass](#) is [crossRef](#), then [crossRef](#) invariants (section 3.1.1.5.2.7), as well as NC naming invariants (section 3.1.1.5.2.6) are enforced.
- It is disallowed to create objects with duplicate RDN values under the same parent **container**, even if objects have distinct values of [rdnType](#). For example, it is disallowed to have two objects named CN=test,DC=Microsoft,DC=com and OU=test,DC=Microsoft,DC=com. The string RDN values are compared according to **Unicode** syntax comparison rules (see LDAP Syntaxes in section 3.1.1.3).
- All attribute names/OIDs refer to attributes that are defined and active in the schema. If an unknown or defunct attribute is referenced, Add returns *unwillingToPerform / ERROR\_DS\_ATT\_NOT\_DEF\_IN\_SCHEMA*.
- Object quota invariants are satisfied for the requestor in the NC where the object is being added (see Quotas in section 3.1.1.5).



- The [objectClass](#) being created satisfies possSuperior schema constraint for the [objectClass](#) of the parent object. Otherwise, *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION* is returned.
- The set of attributes provided for object creation are consistent with the schema definition of the [objectClass](#) values assigned to the object (see section [3.1.1.2](#) for more information):
  - mayContain/mustContain constraints that are applicable based on the selected [objectClass](#) values are enforced. The computation of the mayContain/mustContain set takes into consideration the complete inheritance chain of the structural [objectClass](#) as well as any auxiliary classes supplied.
  - All attribute values are formed correctly according to the attribute syntax, and satisfy schema constraints, such as single-valuedness, minRange/maxRange, etc. See Range Retrieval of Attribute Values in section [3.1.1.3](#) for more information.
  - The attributeType of the first label of the object DN matches the rdnTypeId of the structural object class. Otherwise, *unwillingToPerform* / *ERROR\_DS\_RDN\_DOESNT\_MATCH\_SCHEMA* is returned. For example, it is disallowed to create an [organizationalUnit](#) with CN=test RDN; the correct RDN for an [organizationalUnit](#) object is OU=test.
- If the requestor has supplied a value for RDN attribute, then it matches the first label of the supplied DN value in both attribute type and attribute value (according to **LDAP Unicode** string comparison rules in section [3.1.1.3](#)).
- The RDN value satisfies schema constraints (minRange/maxRange, single-valuedness, syntax, etc.).
- If [site](#) object is being created, then the RDN value is a valid DNS name label (according to the DNS RFC [\[RFC1035\]](#)). Otherwise, *unwillingToPerform* / *ERROR\_DS\_BAD\_NAME\_SYNTAX* is returned.
- If a [subnet](#) object is being created, then the RDN value MUST be a valid subnet object name, according to the algorithm described in section [7.1.1](#)). Otherwise, *unwillingToPerform* / *ERROR\_DS\_BAD\_NAME\_SYNTAX* is returned.
- In the following two cases the requestor specifies the [objectGUID](#) or the [objectSid](#) during add:
  - The requestor is allowed to specify the [objectGUID](#) if the following four conditions are all satisfied:
    - fSpecifyGUIDOnAdd = true in [dSHeuristics](#) (section [7.1.1.2.4.1.2](#)).
    - The requestor has RIGHT\_DS\_ADD\_GUID on the NC root of the NC where the object is being added.
    - The requestor-specified [objectGUID](#) is not currently in use in the forest.
    - Active Directory is operating as AD/DS.
  - The requestor is required to specify the [objectSid](#) when creating a bind proxy object (section [3.1.1.8.2](#)) in an AD/LDS NC. The [objectSid](#) value specified for a bind proxy object must be resolvable by the machine running the AD/LDS DC to an active Windows user. If the SID cannot be resolved to an active Windows user, add returns *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY*. If the requestor-specified [objectSid](#) value is present on an existing object in the same NC, add returns *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY*.

In all other cases it is an error (*unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY*) for the requestor to specify the [objectGUID](#) or [objectSid](#) during add; these values are automatically generated (as specified in the Processing Specifics section below) by the system as required.

- If the requestor has specified a value for the [ntSecurityDescriptor](#) attribute, then it is a valid SD value in self-relative format, and it satisfies the security descriptor constraints (see Security Descriptor Invariants in section [7.1.3](#)).
- If the requestor has specified a value for the [objectCategory](#) attribute, then it points to an existing [classSchema](#) object in the schema container.
- If the requestor has specified a value for the [servicePrincipalName](#) attribute, then it is a syntactically valid SPN value (see section [5.1.1.4](#), "Mutual Authentication").
- The requestor cannot specify a value for the [proxiedObjectName](#) attribute.
- If [msDS-Entry-Time-To-Die](#) attribute is set, then the [objectClass](#) value includes the [dynamicObject](#) auxiliary class.
- If a value is provided for the [entryTTL](#) **constructed attribute**, then it is greater than or equal to the value of the DynamicObjectMinTTL **LDAP** setting (section [3.1.1.3.4.7](#)).
- If a value is provided for the [msDS-Entry-Time-To-Die](#) attribute, then it is greater than or equal to the current system time plus the value of the DynamicObjectMinTTL LDAP setting (section [3.1.1.3.4.7](#)).
- If a non-[dynamicObject](#) child is created under a [dynamicObject](#) parent (see Delete, section [3.1.1.5.5](#)), then *unwillingToPerform* / *ERROR\_DS\_UNWILLING\_TO\_PERFORM* is returned.
- If a value is provided for the [msDS-PasswordHistoryLength](#) attribute, then it is less than or equal to 1024. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-MinimumPasswordAge](#) attribute, then it is less than or equal to 0. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-MaximumPasswordAge](#) attribute, then it is less than or equal to 0. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-MaximumPasswordAge](#) attribute, then it is less than or equal to the value of the [msDS-MinimumPasswordAge](#) attribute on the same object after the add would have completed. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-MinimumPasswordLength](#) attribute, then it is less than or equal to 256. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-LockoutDuration](#) attribute, then it is less than or equal to 0. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-LockoutObservationWindow](#) attribute, then it is less than or equal to 0. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-LockoutDuration](#) attribute, then it is less than or equal to the value of the [msDS-LockoutObservationWindow](#) attribute on the same object after the add would

have completed. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.

- An AD/LDS security principal (an object that is not an NC root and contains the [objectSid](#) attribute) can be created in an application NC. In addition, if the ADAMAllowADAMSecurityPrincipalsInConfigPartition **LDAP** policy equals 1, an AD/LDS security principal can also be created in the config NC. The constraint on security principal creation in the config NC does not apply to the creation of well-known groups in the Roles container (section [7.1.1.4.12](#)). An AD/LDS security principal can never be created in the schema NC.
- In AD/LDS, if the **LDAP** policy ADAMDisablePasswordPolicies does not equal 1, and a password value (either [unicodePwd](#) or [userPassword](#)) is specified in an add, the password must satisfy the current password policy in effect on the AD/LDS server as reported by SamrValidatePassword ([MS-SAMR] section 3.2.5.13.8). If the provided password value does not satisfy the password policy, the add returns *constraintViolation* / *ERROR\_PASSWORD\_RESTRICTION*.
- In AD/LDS, if the [dSHeuristics](#) value gfAllowPasswordOperationsOverNonSecureConnection does not equal true, and a password value (either [unicodePwd](#) or [userPassword](#)) is specified in an add, the **LDAP** connection must be encrypted with cipher strength of at least 128 bits. If the connection does not pass the test, the add returns *operationsError* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- In AD/LDS, if the [userPrincipalName](#) value is specified in an add, then the value must be unique within all NCs on this DC. If another object exists with the same [userPrincipalName](#) value, the add returns *constraintViolation* / *ERROR\_DS\_NAME\_NOT\_UNIQUE*.
- In AD/LDS, the following attributes are disallowed in an add: [badPwdCount](#), [badPasswordTime](#), [lastLogonTimestamp](#), [pwdLastSet](#). If one of these attributes is specified in an add, the add returns *constraintViolation* / *ERROR\_DS\_ATTRIBUTE\_OWNED\_BY\_SAM*.
- Additional constraints may be enforced if the object being created is a SAM-specific object (section [3.1.1.5.2.3](#)); [MS-SAMR] section 3.2.1.6 specifies these constraints.
- Additional constraints may be enforced if the object being created is a schema object (section [3.1.1.5.2.3](#)). See Schema section [3.1.1.2](#) for more information.

### 3.1.1.5.2.3 Special Classes and Attributes

This section defines three sets of object classes: LSA-specific object classes, SAM-specific object classes, and schema object classes. These sets are mentioned elsewhere in the specification, because special processing is applied to instances of these classes.

Each set includes both the specific object classes mentioned below and any subclasses of these object classes.

- LSA-specific object classes: [secret](#), [trustedDomain](#) (originating updates only, in AD/DS only).
- SAM-specific object classes: [group](#), [samDomain](#), [samServer](#), [user](#) (originating updates only, in AD/DS only).
- Schema object classes: [attributeSchema](#), [classSchema](#) (originating and replicated updates).

This section also defines one set of attributes: FPO-enabled attributes. This set is mentioned elsewhere in the specification, because special processing is applied to instances of these attributes.

- FPO-enabled attributes: [member](#), [msDS-MembersForAzRole](#), [msDS-NeverRevealGroup](#), [msDS-NonMembers](#), [msDS-RevealOnDemandGroup](#), [msDS-ServiceAccount](#).

### 3.1.1.5.2.4 Processing Specifics

- For originating updates, a new [objectGUID](#) value is generated and set on the object. For replicated updates, the received [objectGUID](#) is set on the object.
- In AD/DS, if the object is a security principal (according to its [objectClass](#) values), then for originating updates the [objectSid](#) value is generated and set on the object (see [\[MS-SAMR\]](#) sections [3.2.1.6](#) and [3.2.1.9](#)). For replicated updates, the received [objectSid](#) is set on the object.
- In AD/LDS, if the object being added is an NC root and not the schema NC root, then it is given an [objectSid](#) value, ignoring schema constraints. The [objectSid](#) value is generated using the following algorithm, which produces a random SID with 1 sub-authority:
  - The SID authority value (6 bytes) is generated as follows: the first 2 bytes are zero, the high 4 bits of the 3<sup>rd</sup> byte are 0001, and the remaining 3.5 bytes (the lower 4 bits of the 3<sup>rd</sup> byte, and bytes 4, 5 and 6) are randomly generated.
  - The first sub-authority value (DWORD) is randomly generated.
- In AD/LDS, if the object being added is an AD/LDS security principal (an object that is not an NC root and contains the [objectSid](#) attribute), then the [objectSid](#) value is generated using the following algorithm, which produces a random SID with 5 sub-authorities:
  - The SID of the NC root is taken as a prefix (SID authority plus one sub-authority)
  - A randomly generated **GUID** value (16 bytes or 4 DWORDs) is taken as 2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup> and 5<sup>th</sup> sub-authorities of the new SID value. This **GUID** value is unrelated to the [objectGUID](#) value that is also generated randomly for the object being added.
- In AD/LDS, if a [group](#) object is being created (i.e. an object containing the value [group](#) in its [objectClass](#)), and the [groupType](#) attribute is not specified, then the following value is assigned to [groupType](#): GROUP\_TYPE\_ACCOUNT\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED.
- In AD/LDS, if an AD/LDS user is being created, and the password value (either [unicodePwd](#) or [userPassword](#)) was not supplied, then the password value is defaulted to an empty string.
- In AD/LDS, if an AD/LDS user is being created, and the password value (either supplied or defaulted) does not satisfy the password policy in effect on the AD/LDS server (as reported by SamrValidatePassword, [\[MS-SAMR\]](#) section 3.2.5.13.8), then the user is created in the disabled state, i.e. [msDS-UserAccountDisabled](#) = true. However, if the add operation specified the [msDS-UserAccountDisabled](#) attribute with the value of false, the add returns *constraintViolation / ERROR\_PASSWORD\_RESTRICTION*.
- In AD/LDS, if an AD/LDS user is being created, then [badPwdCount](#) and [badPasswordTime](#) values are set to zero.
- The [ntSecurityDescriptor](#) value is computed and set on the object (see section [7.1.3](#) for more details).
- Any values specified for attributes marked as constructed in the schema are ignored, with one exception: the [entryTTL](#) attribute.
- If a value of the [entryTTL](#) **constructed attributess** is specified in the Add request, it is processed as follows: Write the current system time, plus the specified [entryTTL](#) value interpreted as seconds, into the [msDS-Entry-Time-To-Die](#) attribute.

- If [dynamicObject](#) is present among [objectClass](#) values, but neither [entryTTL](#) nor [msDS-Entry-Time-To-Die](#) was specified in an originating update, then Add proceeds as if the value of the [DynamicObjectDefaultTTL](#) **LDAP** policy had been specified as the value of the [entryTTL](#) attribute.
- Any values specified by the requestor for the following attributes are ignored: [distinguishedName](#), [subRefs](#), [uSNLastObjRem](#), [uSNSALastObjRemoved](#), [uSNCreated](#), [replPropertyMetaData](#), [isDeleted](#).
- For an originating update, any value specified for the [whenCreated](#) attribute is ignored.
- If a value of the [systemFlags](#) attribute is specified by the requestor, the DC removes any flags not listed below from the [systemFlags](#) value before storing it on the new object:
  - FLAG\_CONFIG\_ALLOW\_RENAME
  - FLAG\_CONFIG\_ALLOW\_MOVE
  - FLAG\_CONFIG\_ALLOW\_LIMITED\_MOVE
  - FLAG\_ATTR\_IS\_RDN (removed unless the object is an [attributeSchema](#) object)
- For the following scenarios, the DC sets additional bits in the [systemFlags](#) value of the object created:
  - [server](#) objects: FLAG\_DISALLOW\_MOVE\_ON\_DELETE, FLAG\_CONFIG\_ALLOW\_RENAME, FLAG\_CONFIG\_ALLOW\_LIMITED\_MOVE.
  - [site](#), [serversContainer](#), [nTDSDSA](#) objects: FLAG\_DISALLOW\_MOVE\_ON\_DELETE.
  - [siteLink](#), [siteLinkBridge](#), [nTDSConnection](#) objects, and any object whose parent is a [site](#) or [subnet](#) object: FLAG\_CONFIG\_ALLOW\_RENAME.
- If a value for the [objectCategory](#) attribute was not specified by the requestor, then it is defaulted to the current value of the [defaultObjectCategory](#) attribute on the [classSchema](#) object corresponding to the Most specific structural object class of the object being added.
- The complete inheritance chain of object classes (starting from the most specific structural object class as well as from all dynamic auxiliary classes specified by the user) is computed and set. The correct ordering of [objectClass](#) values is performed (see section [3.1.1.2.4.3](#) for more details).
- The value of [instanceType](#) attribute is written. For originating updates of regular objects, it is IT\_WRITE. For NC root object specifics, see NC-Add operation section below. For replicated updates, the [instanceType](#) value computed by the IDL\_DRSGetNCChanges client is written.
- [distinguishedName](#) attribute is written, matching the supplied object DN value.
- RDN attribute of the correct attribute type is written, as computed from the supplied object DN value.
- If [showInAdvancedViewOnly](#) value was not provided by the requestor and the [defaultHidingValue](#) of the [objectClass](#) is true, then [showInAdvancedViewOnly](#) attribute value is set to true.
- If the add assigns a value to an FPO-enabled attribute (section [3.1.1.5.2.3](#)) of the new object, and the DN value in the add request has <SID=stringizedSid> format (section [3.1.1.3.1.2.4](#)), then the DC creates a corresponding [foreignSecurityPrincipal](#) object in the ForeignSecurityPrincipals container (section [7.1.1.4.9](#)) and assigns a reference to the new [foreignSecurityPrincipal](#) object as the FPO-enabled attribute value. [MS-SAMR] section 3.2.1.8.9 specifies the creation of the [foreignSecurityPrincipal](#) object.

- If [attributeSchema](#) or [classSchema](#) object is created in schema NC, then apply special processing as described in section [3.1.1.2.5](#).
- If an [infrastructureUpdate](#) object is created, then, let O be the object that is created. If (O![dnReferenceUpdate](#) has a value) then for each object P in each NC replica on the server do the following:
  - Let S be the set of all attributes of P with attribute syntax Object(DS-DN), Object(DN-String), Object(DN-Binary), Object(OR-Name), or Object(Access-Point).
  - For each attribute A in set S and for each value V of A do the following:
    - If attribute syntax of A is Object(DS-DN) then let G be P.A.guid\_value.
    - Otherwise, let G be P.A.V.object\_DN.guid\_value.
    - Let RG be O![dnReferenceUpdate](#).guid\_value.
    - Let RD be O![dnReferenceUpdate](#).dn.
    - If (RG = G) then delete V from P.A.
    - If (RG = G) and A is not a link value attribute then add attribute value of O![dnReferenceUpdate](#) to P.A
    - If (RG = G) and A is a link value attribute and RDN of RD is not a delete-mangled RDN (see section [3.1.1.5.5](#)) then add value of O![dnReferenceUpdate](#) to P.A.

#### 3.1.1.5.2.5 Quota Invariants

The quotas control the number of objects and tombstones that a requestor may own within an NC. The requestor is considered the *owner* of an object if the OWNER field in the object's [nTSecurityDescriptor](#) value equals requestor's SID. If the USAGE value computed for the requestor exceeds the MAX-USAGE value (see below), then the add operations in this NC are prevented for the requestor, and the server returns *unwillingToPerform* / *STATUS\_QUOTA\_EXCEEDED* error. Note that quota enforcement also applies to undelete (reanimation) operations (see Undelete section for more details) as well as modify operations that change the owner of the object.

The USAGE value is computed as follows:

$$\text{USAGE} = \text{owned\_objects} + \text{ceil}(\text{tombstone-factor}/100 * \text{owned\_tombstones})$$

In the formula above, *owned\_objects* is the total number of live (not deleted) objects that the requestor owns. *Owned\_tombstones* is the total number of tombstones (see Delete operation in section [3.1.1.5.5](#)) that the requestor owns. *Tombstone-factor* is the integer value stored in [msDS-TombstoneQuotaFactor](#) attribute on the Quotas container in the NC. Ceil() is the "ceiling" mathematical function.

The MAX-USAGE value is computed as follows:

1. A set of applicable [msDS-QuotaControl](#) objects in Quotas container is obtained. An [msDS-QuotaControl](#) object is applicable for the requestor if its [msDS-QuotaTrustee](#) attribute contains a SID that is present in requestor's authorization information.
2. If the set of applicable [msDS-QuotaControl](#) objects is non-empty, then the maximum value of [msDS-QuotaAmount](#) attribute is chosen as the MAX-USAGE value.



3. If the set of applicable [msDS-QuotaControl](#) objects is empty, then the value of [msDS-DefaultQuota](#) attribute on the Quotas container is chosen as the MAX-USAGE value.

#### 3.1.1.5.2.6 NC Invariants

The following invariants are maintained for DNs of AD/DS NCs (the set of NCs that are parts of the AD forest) other than the config NC and schema NC:

- Each RDN label within the DN has the DC= type.
- Each RDN label within the DN has a value, which is a valid DNS name label.

The following invariants are maintained for DNs of all AD NCs:

- The full DN of the NC does not match the DN of another existing object in an AD NC.
- If the immediate parent of the NC is not an AD NC, then none of the ancestors (grandparent, grand-grandparent etc) are an AD NC. In other words, the set of AD NCs is a set of non-intersecting trees, and each tree does not have "holes".

The following invariants apply to the data stored in NC roots:

- IT\_NC\_HEAD bit is set in the [instanceType](#) attribute.
- If the NC has an immediate parent (which must be an NC root per the rules above), then IT\_NC\_ABOVE bit is set in its [instanceType](#) attribute.
- If the NC has child NCs, then their DNs are listed in its [subRefs](#) attribute.

The default structure of data in NCs is covered in Naming Contexts in section [7.1](#).

#### 3.1.1.5.2.7 crossRef Invariants

[crossRef](#) objects represent NCs within the AD forest, as well as "external" (foreign) NCs. The relationship between the [crossRef](#) and the NC is represented by [nCName](#) attribute on the [crossRef](#). The value of this attribute is the DN of the corresponding NC. Each AD NC has a corresponding [crossRef](#) object. A [crossRef](#) object can also represent an *intention* to create a new AD NC with the specified DN.

The following invariants apply to [crossRef](#) objects.

- The FLAG\_CR\_NTDS\_NC bit is set in [systemFlags](#) if and only if the [nCName](#) represents an AD NC.
- The FLAG\_CR\_NTDS\_DOMAIN bit is set in [systemFlags](#) if and only if the [nCName](#) represents a **domain** AD NC.
- The FLAG\_CR\_NTDS\_NOT\_GC\_REPLICATED bit is set in [systemFlags](#) if and only if the [nCName](#) represents an Application AD NC.
- If the FLAG\_CR\_NTDS\_NC bit is set in [systemFlags](#) and the [Enabled](#) attribute value is false, then the [crossRef](#) represents an *intention* to create an AD NC. Otherwise, it represents an AD NC that is actually present.

#### 3.1.1.5.2.8 NC-Add Operation

For originating updates, NC-Add operation is distinguished by the presence of [instanceType](#) attribute with (IT\_NC\_HEAD | IT\_WRITE) value in the input attribute set. For replicated updates, NC-Add

operation is distinguished by the presence of [instanceType](#) attribute with IT\_NC\_HEAD value in the input attribute set. The DN of the object represents the new NC DN, and the DC enforces the constraints on NC naming described above.

For originating updates, NC-Add operation is only supported for application NCs. If a new **domain** NC needs to be created, then IDL\_DRSAddEntry **RPC** MUST be used to create the [crossRef](#) (see [MS-DRSR] section [4.1.1](#)).

#### 3.1.1.5.2.8.1 Constraints

Regular add operation constraints apply to NC-Add operation (as defined above), with the exception of constraints pertaining to the parent object (for example, [possSuperiors](#) schema constraint).

There are two distinct NC-Add scenarios that are supported with regards to maintaining [crossRef](#) invariants:

1. The [crossRef](#) corresponding to the new NC does not exist. In this case, a new [crossRef](#) object is created. If the DC is the **domain** naming FSMO, then the [crossRef](#) is created locally. Otherwise, the [crossRef](#) is created on the **domain** naming FSMO DC using the IDL\_DRSAddEntry call with appropriate parameters (see [MS-DRSR] section [4.1.1](#) for details).
2. The [crossRef](#) corresponding to the new NC has been pre-created, i.e., was created previously. It is a [crossRef](#) object that satisfies the following requirements:
  1. The [nCName](#) matches the DN of the NC being created;
  2. [Enabled](#) attribute is false;
  3. [dnsRoot](#) attribute value matches the dnsName of the DC processing the Add-NC operation;
  4. [systemFlags](#) value has these bits set: FLAG\_CR\_NTDS\_NC and FLAG\_CR\_NTDS\_NOT\_GC\_REPLICATED.

#### 3.1.1.5.2.8.2 Security Considerations

Regular add access checks do not apply to NC-Add operation, because the parent object may not even exist in the directory. Instead, the requestor must have sufficient permissions to either create a new [crossRef](#) or modify the pre-created [crossRef](#) object. Regular add and modify permission checks apply for these operations.

No **access check** is performed for replicated updates.

#### 3.1.1.5.2.8.3 Processing Specifics

The following operations are performed during an NC-Add operation performed as an originating update:

- The matching [crossRef](#) object is obtained (see details in section [3.1.1.5.2.8.1](#) above).
- The NC root object is created per the Add request. Regular Add processing applies (as defined in sections [3.1.1.5.2.1](#) through [3.1.1.5.2.3](#) above).
- The default NC tree structure is generated (see Naming Contexts in section [7.1](#)), and the appropriate [wellKnownObjects](#) references are written on the NC root.
- The matching [crossRef](#) object is updated as follows: the [Enabled](#) attribute is removed, the [dnsRoot](#) is updated to contain the full DNS name of the NC, as computed from the NC DN.



These steps are not performed for replicated updates.

### 3.1.1.5.3 Modify Operation

#### References

LDAP attributes: [objectClass](#), [nTSecurityDescriptor](#), [instanceType](#), [distinguishedName](#), [objectGUID](#), [objectSid](#), [entryTTL](#), [msDS-Entry-Time-To-Die](#), [systemFlags](#), [objectCategory](#), [msDS-AllowedToDelegateTo](#), [member](#), [sAMAccountName](#), [msDS-AdditionalSamAccountName](#), [dNSHostName](#), [msDS-AdditionalDnsHostName](#), [servicePrincipalName](#), [uSNCreated](#), [subRefs](#), [uSNLastObjRem](#), [uSNSALastObjRemoved](#), [name](#), [isDeleted](#), [hasMasterNCs](#), [msDS-hasMasterNCs](#), [hasPartialReplicaNCs](#), [msDS-hasFullReplicaNCs](#), [whenCreated](#), [managedBy](#), [msDS-LockoutObservationWindow](#), [msDS-LockoutDuration](#), [msDS-MaximumPasswordAge](#), [msDS-MinimumPasswordAge](#), [msDS-MinimumPasswordLength](#), [msDS-PasswordHistoryLength](#).

LDAP classes: [dynamicObject](#), [crossRef](#), [server](#), [computer](#), [foreignSecurityPrincipal](#).

WKO GUIDs: GUID\_USERS\_CONTAINER\_W, GUID\_COMPUTERS\_CONTAINER\_W.

#### Constants

- Win32/status error codes: ERROR\_DS\_REFERRAL, ERROR\_DS\_WKO\_CONTAINER\_CANNOT\_BE\_SPECIAL, ERROR\_DS\_CONFIDENTIALITY\_REQUIRED, ERROR\_DS\_ILLEGAL\_MOD\_OPERATION.
- Access mask bits, CARs: RIGHT\_DS\_WRITE\_PROPERTY, RIGHT\_DS\_WRITE\_PROPERTY\_EXTENDED, RIGHT\_DS\_CHANGE\_INFRASTRUCTURE\_MASTER, RIGHT\_DS\_CHANGE\_SCHEMA\_MASTER, RIGHT\_DS\_CHANGE\_RID\_MASTER, RIGHT\_DS\_CHANGE\_PDC, RIGHT\_DS\_CHANGE\_DOMAIN\_MASTER, RIGHT\_DS\_REANIMATE\_TOMBSTONES.
- Security privileges: SE\_ENABLE\_DELEGATION\_PRIVILEGE
- [systemFlags](#) bits: FLAG\_DISALLOW\_DELETE, FLAG\_DOMAIN\_DISALLOW\_RENAME, FLAG\_DOMAIN\_DISALLOW\_MOVE, FLAG\_ATTR\_IS\_RDN.
- LDAP: LDAP\_SERVER\_PERMISSIVE\_MODIFY\_OID

The modify operation results in modification of a single existing object in the directory tree. The requestor supplies the following data:

- The DN of the object.
- The set of attributes defining the modifications that should be performed.

#### 3.1.1.5.3.1 Security Considerations

For originating updates, the following access checks are performed. No access checks are performed for replicated updates.

The object being modified must be visible to the requestor. In addition, the requestor needs to have RIGHT\_DS\_WRITE\_PROPERTY access to all attributes being directly affected by the modify operation. Note, some attributes may be modified indirectly as a result of triggers/processing rules. The requestor is not required to have write access to those attributes.

Additional access checks may apply if [nTSecurityDescriptor](#) value is being modified. See Security Descriptor Invariants in section [7.1.3](#) for more details.

If the modify operation represents an Undelete operation, then additional security checks apply (see the Undelete operation in section [3.1.1.5.3.7](#)).

If [msDS-AllowedToDelegateTo](#) attribute is modified, then the requestor must possess SE\_ENABLE\_DELEGATION\_PRIVILEGE.

In AD/LDS, if a password value is being modified as a password change operation, then the requestor needs to have the RIGHT\_DS\_USER\_CHANGE\_PASSWORD **control access right** on the object being modified. A password change operation is defined as removing the old password value and adding the new password value, where the old password value matches the current password on the object.

In AD/LDS, if a password value is being modified as a password reset operation, then the requestor needs to have RIGHT\_DS\_USER\_FORCE\_CHANGE\_PASSWORD **control access right** on the object being modified. A password reset operation is defined as a replace operation on the password attribute.

In AD/LDS, if a password unexpire operation is being performed, then the requestor needs to have RIGHT\_DS\_USER\_UNEXPIRE\_PASSWORD **control access right** on the object being modified. A password unexpire operation is defined as setting the [pwdLastSet](#) attribute to the value -1.

### 3.1.1.5.3.1.1 Validated Writes

In some cases, when the requestor does not have RIGHT\_DS\_WRITE\_PROPERTY on an attribute, but has RIGHT\_DS\_WRITE\_PROPERTY\_EXTENDED (also called "validated write"), then the write is allowed, subject to additional constraints for the attribute value. The following sub-sections specify the additional checks performed for validated writes.

#### 3.1.1.5.3.1.1.1 Member

The operation is either add value or remove value, and the value is the DN of the [user](#) object representing the requestor. In other words, it is allowed that one can add/remove oneself to/from a group.

#### 3.1.1.5.3.1.1.2 dNSHostName

The object has class [computer](#) or [server](#) (or a subclass of [computer](#) or [server](#)).

In AD/DS, the value of the [dNSHostName](#) attribute being written is in the following format: *computerName.fullDomainDnsName*, where *computerName* is the current [sAMAccountName](#) of the object (without the final '\$' character), and the *fullDomainDnsName* is the DNS name of the **domain** NC where the object being modified is located.

#### 3.1.1.5.3.1.1.3 msDS-AdditionalDnsHostName

The object has class [computer](#) or [server](#) (or a subclass of [computer](#) or [server](#)).

In AD/DS, the value of the [msDS-AdditionalDnsHostName](#) attribute being written is in the following format: *anyDnsLabel.suffix*, where *anyDnsLabel* is a valid DNS name label, and *suffix* matches one of the values of [msDS-AllowedDNSSuffixes](#) on the **domain** NC root (if any).

#### 3.1.1.5.3.1.1.4 servicePrincipalName

The object has class [computer](#) (or a subclass of [computer](#)).

In AD/DS, the [servicePrincipalName](#) value satisfies the following constraints:

- The SPN is a syntactically correct 2-part SPN (see section [5.1.1.4](#), "Mutual Authentication").
- The instance name matches either the full DNS name of the machine or the [sAMAccountName](#) of the machine (without the terminating '\$') or one of the [msDS-AdditionalDnsHostName](#) or one of the [msDS-AdditionalSamAccountName](#) (without the terminating '\$').

### 3.1.1.5.3.1.2 FSMO Changes

If a write to the [fSMORoleOwner](#) attribute is performed, and the [objectClass](#) of the object being modified is one of the classes listed below, then the requestor is required to have an additional **CAR** (**control access right**) on the object. The following CARs are checked, depending on the [objectClass](#) of the object being modified:

- [infrastructureUpdate](#) (domain infrastructure master FSMO, in AD/DS only):  
RIGHT\_DS\_CHANGE\_INFRASTRUCTURE\_MASTER
- [dMD](#) (schema FSMO): RIGHT\_DS\_CHANGE\_SCHEMA\_MASTER
- [rIDManager](#) (domain RID FSMO, in AD/DS only): RIGHT\_DS\_CHANGE\_RID\_MASTER
- [domainDNS](#) (PDC emulator FSMO, in AD/DS only): RIGHT\_DS\_CHANGE\_PDC
- [crossRefContainer](#) (domain naming FSMO): RIGHT\_DS\_CHANGE\_DOMAIN\_MASTER.

### 3.1.1.5.3.2 Constraints

The following constraints are enforced for a modify operation performed as an originating update. If any of them are not satisfied, the server returns *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION* (unless specified otherwise). These constraints are not enforced for replicated updates.

- The object resides in a writable NC replica, otherwise the modify returns *referral* / *ERROR\_DS\_REFERRAL*.
- In AD/DS, if the object being modified is in the config NC or schema NC, and the RM control of the SD is present and contains the SECURITY\_PRIVATE\_OBJECT (section [7.1.3](#)) bit, the DC requires one of the following two conditions to be true:

- The DC is a member of the root **domain** in the forest.
- The DC is a member of the same **domain** to which the current object owner belongs.

If neither condition is true the modify returns *referral* / *ERROR\_DS\_REFERRAL*.

- If a LostAndFound container is being modified, the modify returns *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- If the object being modified has class [subSchema](#), then only [nTSecurityDescriptor](#) modifications are allowed.
- Modifying an object with [isDeleted](#) = true (a tombstone) is allowed only if one of the following conditions is true:
  - The operation is an undelete operation (either setting [isDeleted](#) = false, or removing the [isDeleted](#) attribute). Note that the undelete operation is a special case of the modify operation. See section [3.1.1.5.3.7](#) for more information.

- The object being modified is the Deleted Objects container (section [7.1.1.4.2](#)).
- The modification only affects the [nTSecurityDescriptor](#) attribute, and the requestor has RIGHT\_DS\_REANIMATE\_TOMBSTONES on the NC root of the object's NC.
- In AD/DS, modifications to objects of LSA-specific object classes (section [3.1.1.5.2.3](#)) are disallowed.
- Changes to objects in the Partitions container (class [crossRef](#)) are only allowed when the DC is the **domain** naming FSMO, otherwise *referral* / *ERROR\_DS\_REFERRAL* is returned.
- Modifications of **constructed attributes** are disallowed (with the exception of the [entryTTL](#) attribute)
- Updates to the [name](#) attribute, as well as updates to the object's naming attribute (the attribute named by the [rdnType](#) attribute), are disallowed. ModifyDN performs these updates.
- A Modify of an object whose [objectClass](#) is defunct fails.
- If the forest functional level is less than DS\_BEHAVIOR\_WIN2003, a Modify is allowed to remove all values of a defunct attribute. In all other cases, a Modify that references a defunct attribute fails.
- [objectCategory](#) modifications on [classSchema](#) objects that have FLAG\_SCHEMA\_BASE\_OBJECT present in [systemFlags](#) are disallowed.
- If the forest functional level is less than DS\_BEHAVIOR\_WIN2003, then modifications of [msDS-AdditionalDnsHostName](#) are disallowed.
- If the [msDS-UpdateScript](#) attribute is being modified, and the connection is not encrypted with at least 128-bit cipher, then *unwillingToPerform* / *ERROR\_DS\_CONFIDENTIALITY\_REQUIRED* is returned.
- In AD/DS, if the [wellKnownObjects](#) attribute is being modified, then additional constraints are enforced. See the [wellKnownObjects](#) Updates section below for more information.
- If the [dSHeuristics](#) attribute is being modified, the new value must satisfy the following constraints:
  - If the length of the value is 10 or more characters, then 10<sup>th</sup> character must be '1';
  - If the length of the value is 20 or more characters, then 20<sup>th</sup> character must be '2';
  - If the length of the value is 30 or more characters, then 30<sup>th</sup> character must be '3';
  - Same for '4' through '9'.
- If the [msDS-Behavior-Version](#) attribute is modified, then additional constraints apply. See the BehaviorVersion updates section below for more information.
- [nTMixedDomain](#) attribute may only be modified on the **domain** NC root, but not on [crossRef](#) objects.
- If [servicePrincipalName](#) attribute is modified, then the values must be syntactically valid SPN values (note additional constraints may apply if the requestor did not have WRITE\_PROPERTY access to the attribute, see Validated writes section above). See section [5.1.1.4](#), "Mutual Authentication," for SPN syntax.

- If the [fsmoRoleOwner](#) attribute is modified, then the only allowed attribute value is the DN of the DSA object of the current DC. In other words, the FSMO role can only be "taken", or transferred to the current DC. It cannot be given away.
- [proxiedObjectName](#) attribute modifications are disallowed.
- It is disallowed to add or remove the [dynamicObject](#) auxiliary class to/from an existing object.
- If a value is provided for the [entryTTL](#) **constructed attributes**, then it is greater than or equal to the value of the DynamicObjectMinTTL LDAP setting (section [3.1.1.3.4.7](#)).
- If a value is provided for the [msDS-Entry-Time-To-Die](#) attribute, then it is greater than or equal to the current system time plus the value of the DynamicObjectMinTTL LDAP setting (section [3.1.1.3.4.7](#)).
- System-only attribute modifications (including the case of adding an auxiliary class with a must-have system-only attribute) are disallowed, as well as modifications of the following attributes: [distinguishedName](#), [uSNCreated](#), [subRefs](#), [uSNLastObjRem](#), [uSNDALastObjRemoved](#), [name](#), [isDeleted](#), [hasMasterNCs](#), [msDS-hasMasterNCs](#), [hasPartialReplicaNCs](#), [msDS-hasFullReplicaNCs](#), [whenCreated](#), and all back link attributes; with the following exceptions:
  - [objectClass](#) may be modified, subject to additional constraints (see ObjectClass modifications section below).
  - [msDS-Behavior-Version](#) may be modified (see the BehaviorVersion updates section below).
  - [msDS-AdditionalDnsHostName](#) may be modified.
  - [systemFlags](#) may be modified, only in the following case: the modify is on an [attributeSchema](#) object in the schema container, and the change is to set (but not reset) FLAG\_ATTR\_IS\_RDN bit.
  - [wellKnownObjects](#) may be modified (subject to the constraints described below)
  - [isDeleted](#) and [distinguishedName](#) may be modified, only when the modify operation is Undelete (see Undelete in section [3.1.1.5.3.7](#))
  - [mAPIID](#) may be modified, subject to the constraints described in section [3.1.1.2.3](#).
- It is disallowed to insert duplicate values into an attribute (per appropriate syntax comparison rules), or remove values that are not currently present. However, this constraint must be relaxed if the requestor is passing LDAP\_SERVER\_PERMISSIVE\_MODIFY\_OID control.
- If a value is provided for the [msDS-PasswordHistoryLength](#) attribute, then it is less than or equal to 1024. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-MinimumPasswordAge](#) attribute, then it is less than or equal to 0. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-MaximumPasswordAge](#) attribute, then it is less than or equal to 0. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-MaximumPasswordAge](#) attribute, then it is less than or equal to the value of the [msDS-MinimumPasswordAge](#) attribute on the same object after the modify would have completed. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.

- If a value is provided for the [msDS-MinimumPasswordLength](#) attribute, then it is less than or equal to 256. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-LockoutDuration](#) attribute, then it is less than or equal to 0. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-LockoutObservationWindow](#) attribute, then it is less than or equal to 0. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- If a value is provided for the [msDS-LockoutDuration](#) attribute, then it is less than or equal to the value of the [msDS-LockoutObservationWindow](#) attribute on the same object after the modify would have completed. Otherwise *unwillingToPerform* / *ERROR\_DS\_SECURITY\_ILLEGAL\_MODIFY* is returned.
- In AD/LDS, if the **LDAP** policy *ADAMDisablePasswordPolicies* does not equal 1, and a password value (either [unicodePwd](#) or [userPassword](#)) is specified in a modify, the password must satisfy the current password policy in effect on the AD/LDS server as reported by *SamrValidatePassword* ([MS-SAMR] section 3.2.5.13.8). If the provided password value does not satisfy the password policy, the modify returns *constraintViolation* / *ERROR\_PASSWORD\_RESTRICTION*.
- In AD/LDS, if the [dsHeuristics](#) value *gfAllowPasswordOperationsOverNonSecureConnection* does not equal true, and a password value (either [unicodePwd](#) or [userPassword](#)) is specified in a modify, the **LDAP** connection must be encrypted with cipher strength of at least 128 bits. If the connection does not pass the test, the modify returns *operationsError* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- In AD/LDS, if the [userPrincipalName](#) value is modified, then the new value must be unique within all NCs on this DC. If another object exists with the same [userPrincipalName](#) value, the modify returns *constraintViolation* / *ERROR\_DS\_NAME\_NOT\_UNIQUE*.
- In AD/LDS, if the [pwdLastSet](#) attribute is modified, then the operation must be a replace value with a single value 0 or -1. Otherwise, *constraintViolation* / *ERROR\_INVALID\_PARAMETER* is returned.
- In AD/LDS, if the [msDS-UserAccountDisabled](#) attribute is being set to false, then the operation succeeds if one of the following is true:
  - The **LDAP** policy *ADAMDisablePasswordPolicies* equals 1.
  - The [ms-DS-UserPasswordNotRequired](#) attribute equals true.
  - The current password value on the object satisfies the current password policy (as reported by *SamrValidatePassword*, [MS-SAMR] section 3.2.5.13.8).

If this check fails, the modify returns *constraintViolation* / *ERROR\_PASSWORD\_RESTRICTION*.

- After the modify operation, the object must remain compliant with the schema with regards to *mayContain/mustContain* constraints, single-valuedness, etc. See section [3.1.1.2.3](#) through section [3.1.1.2.5](#) for more information. Exception: In AD/LDS, the [objectSid](#) attribute is present on all application NC roots, even if this violates the schema *mayContain/mustContain* constraints.
- If the object being modified is a SAM-specific object (section [3.1.1.5.2.3](#)), then additional constraints apply (specified in [MS-SAMR] section 3.2.1.6).

- If the modify operation affects [nTSecurityDescriptor](#) attribute, then additional constraints apply (see Security Descriptor Invariants section [7.1.3](#) for more information).

### 3.1.1.5.3.3 Processing Specifics

The following processing rules apply to modify operation:

- If a value of the [entryTTL](#) **constructed attributes** is specified in the Modify request, it is processed as follows: Write the current system time, plus the specified [entryTTL](#) value interpreted as seconds, into the [msDS-Entry-Time-To-Die](#) attribute.
- If the modify assigns a value to an FPO-enabled attribute (section [3.1.1.5.2.3](#)) of the existing object, and the DN value in the modify request has <SID=stringizedSid> format (section [3.1.1.3.1.2.4](#)), then the DC creates a corresponding [foreignSecurityPrincipal](#) object in the ForeignSecurityPrincipals container (section [7.1.1.4.9](#)) and assigns a reference to the new [foreignSecurityPrincipal](#) object as the FPO-enabled attribute value. [MS-SAMR] section [3.2.1.8.9](#) specifies the creation of the [foreignSecurityPrincipal](#) object.
- If the [msDS-UpdateScript](#) attribute is changed in an originating update of the Partitions container, then the [msDS-ExecuteScriptPassword](#) value is removed from the Partitions container.
- If [objectClass](#) value is updated, then additional operations are performed (see ObjectClass updates section below for more details).
- In AD/DS, if the [wellKnownObjects](#) value is updated, then additional operations are performed (see the [wellKnownObjects](#) updates section below for more details).
- In AD/LDS, if a password value ([unicodePwd](#) or [userPassword](#)) is modified on a bind proxy, then the password operation is "forwarded" to Windows as follows:
  - The [objectSid](#) on the bind proxy object is resolved to a Windows user object.
  - A DC hosting the Windows user's **domain** is discovered.
  - The currently bound user is impersonated.
  - For a change password operation, the NetUserChangePassword API is invoked with the new and old password values.
  - For a reset password operation, then NetUserSetInfo(level=1003) API is invoked with the new password value.
  - The currently bound user is unimpersonated.

If any of the operations above fail, then the modify returns *unwillingToPerform*.

- In AD/LDS, if the [pwdLastSet](#) attribute is set to -1 (i.e. an unexpire-password operation is performed), then the current time is written as the value of [pwdLastSet](#) attribute.
- For originating updates, additional operations may be performed if the object being modified is a SAM-specific object (section [3.1.1.5.2.3](#)); [MS-SAMR] section [3.2.1.8](#) specifies these additional operations.
- Additional operations may be performed if the object being modified is a schema object (section [3.1.1.5.2.3](#)); the additional operations are specified in section [3.1.1.2.5](#).



### 3.1.1.5.3.4 BehaviorVersion Updates

Originating updates of behavior version are subject to the following additional constraints. When any request would violate one of these constraints, the server returns *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*, just as for constraints described in section [3.1.1.5.3.2](#).

- The [msDS-Behavior-Version](#) attribute may only be modified on the NC root of the **domain** NC (domain version) or the CN=Partitions child of the config NC (forest version).
- Behavior version may only be raised, i.e., the new value must be greater than the old value.
- The operation is performed on the FSMO (PDC for **domain** version updates, **domain** naming FSMO for forest version updates), otherwise *referral* / *ERROR\_DS\_REFERRAL* is returned.
- The **domain**/forest may not contain a DC whose behavior version is lower than the new value. This is determined by searching the config NC for [nTDSDSA](#) objects whose [msDS-Behavior-Version](#) attribute value is below the new value.
- If the version is raised from a value below w2k3 to a value w2k3 or greater, then the forest may not contain mixed-mode **domains**.

### 3.1.1.5.3.5 ObjectClass Updates

When [objectClass](#) value is modified, the following additional constraints are enforced:

- It is disallowed to modify the structural object class of an object, with two exceptions:
  - A [user](#) object may be converted to an [inetOrgPerson](#) by the addition of [inetOrgPerson](#) to the [objectClass](#) attribute; and
  - An [inetOrgPerson](#) may be converted to a [user](#) by the removal of [inetOrgPerson](#) from the [objectClass](#) attribute.
- The [objectClass](#) attribute must contain a single most specific structural object class. If the set of object classes specified by an update contains "holes", i.e., classes are missing on the inheritance chain from the most specific structural object class to the distinguished class [top](#), the server fills the "holes" during the update
- The set of auxiliary classes may be changed, with the following exceptions:
  - [dynamicObject](#) auxClass may not be added or removed.
  - If the forest functional level is less than DS\_BEHAVIOR\_WIN2003, only objects in application NCs may have aux classes added or removed dynamically.

#### Processing specifics:

- The set of values is updated to include the full inheritance chains of the structural object class as well as all auxiliary classes present in the value.
- The set of values is sorted according to the [objectClass](#) invariants (see section [3.1.1.2.4.3](#) for more info).
- A new value of [nTSecurityDescriptor](#) is computed and written based on the new [objectClass](#) values, according to the security descriptor invariants (see section [7.1.3](#)).



### 3.1.1.5.3.6 wellKnownObjects Updates

In AD/DS, when a [wellKnownObjects](#) value is modified by an originating update, the following additional constraints apply. These constraints are not enforced for replicated updates.

- The update is performed on the PDC FSMO, otherwise *referral* / *ERROR\_DS\_REFERRAL* is returned.
- The update is on the **domain** NC root object, otherwise, *unwillingToPerform* / *ERROR\_DS\_UNWILLING\_TO\_PERFORM* is returned.
- The **domain** behavior version is at least w2k3, otherwise *unwillingToPerform* / *ERROR\_DS\_NOT\_SUPPORTED* is returned.
- Only the "users" and "computers" container [wellKnownObjects](#) references may be updated. This corresponds to GUID\_USERS\_CONTAINER\_W and GUID\_COMPUTERS\_CONTAINER\_W WKO GUIDs, respectively, otherwise, *unwillingToPerform* / *ERROR\_DS\_UNWILLING\_TO\_PERFORM* is returned.
- Only add-value and remove-value **LDAP** verbs are supported, otherwise, *unwillingToPerform* / *ERROR\_DS\_UNWILLING\_TO\_PERFORM* is returned.
- The added value (new **container** value) is a **container** in **domain** NC, and it may not reside within the **container** identified by the DN of "CN=System,<domain NC DN>", otherwise, *unwillingToPerform* / *ERROR\_DS\_UNWILLING\_TO\_PERFORM* is returned.
- The new **container** may not have the following bits set in its [systemFlags](#) value: FLAG\_DISALLOW\_DELETE, FLAG\_DOMAIN\_DISALLOW\_RENAME or FLAG\_DOMAIN\_DISALLOW\_MOVE, otherwise *unwillingToPerform* / *ERROR\_DS\_WKO\_CONTAINER\_CANNOT\_BE\_SPECIAL* must be returned.
- The removed value matches the corresponding existing value of the WKO reference.

#### Processing specifics:

- The following bits MUST be set in the [systemFlags](#) of the new **container**: FLAG\_DISALLOW\_DELETE, FLAG\_DOMAIN\_DISALLOW\_RENAME and FLAG\_DOMAIN\_DISALLOW\_MOVE.
- The following bits MUST be reset in the [systemFlags](#) of the old **container**: FLAG\_DISALLOW\_DELETE, FLAG\_DOMAIN\_DISALLOW\_RENAME and FLAG\_DOMAIN\_DISALLOW\_MOVE.
- [isCriticalSystemObject](#) MUST be set to true on the new container.
- [isCriticalSystemObject](#) MUST be set to false on the old container.

### 3.1.1.5.3.7 Undelete Operation

Undelete operation is used to revert the effects of delete operation, i.e., to turn a tombstone into a regular object (see section [3.1.1.5.5](#) for more details). Undelete operation is represented by a regular **LDAP** modify operation, which contains special instructions that are used to distinguish it from a modify operation. These instructions (attribute modifications) are disallowed for regular modify operations.

The undelete operation is identified by the presence of the following attribute LDAPMods (both MUST be present):

- REMOVE [isDeleted](#) attribute
- REPLACE [distinguishedName](#) attribute with a new value

The undelete operation combines characteristics of both Modify and ModifyDN operations. It modifies object's attributes and moves it in the same transaction.

### 3.1.1.5.3.7.1 Undelete Security Considerations

In order to be able to perform the undelete operation as an originating update, the requestor must have the following permissions. No permissions are required for replicated updates.

- The object being undeleted must be visible to the requestor, in the sense of section [5.1.3.3.6](#).
- RIGHT\_DS\_REANIMATE\_TOMBSTONES on the NC root of the NC where the operation is being performed.
- All the permissions required to rename an object (section [3.1.1.5.4](#)).
- CREATE\_CHILD on the new parent **container** for the [objectClass](#) of the object being undeleted.
- The new parent **container** must be visible to the requestor, in the sense of section [5.1.3.3.6](#).

**Note** Unlike with the ModifyDN operation, the Delete/DeleteChild permission is not required.

### 3.1.1.5.3.7.2 Undelete Constraints

For originating updates, the following constraints are enforced for the Undelete operation, otherwise *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION* is returned (unless specified otherwise). These constraints do not apply to replicated updates.

- All the modify constraints as they apply to the attributes being modified within the undelete processing (see above)
- All the ModifyDN constraints as they apply to the "move" portion of the undelete operation, with the exception of the "disallowed to move in or out of system **container**" constraint.
- The target object is a tombstone, i.e., [isDeleted](#) attribute must be true.
- The target object is not the Deleted Objects **container** in its NC.
- The target object is not the [user](#) object of the currently connected user (i.e., the user may not undelete his own object).
- After the modify attribute updates are applied, the object is checked for full schema compliance with regards to both mayContain and mustContain constraints.
- The new object DN is specified in string format (as opposed to <GUID=*stringized-guid*> or <SID=*stringized-sid*> format).
- The new parent **container** is in the same NC as the target tombstone object (i.e., cross-NC undelete is not allowed).

### 3.1.1.5.3.7.3 Undelete Processing Specifics

The undelete operation comprises two sub-operations: modifying the object and moving it to a new location. The destination of the move operation is obtained from the DN specified in the request.

- All the Modify operation processing specifics apply.
- All the ModifyDN operation processing specifics apply.
- If the user did not specify the value for [objectCategory](#) attribute, and the tombstone did not have this value retained at the time of deletion, then the default [objectCategory](#) attribute is written, as obtained from the [objectClass](#)'s [defaultObjectCategory](#) value (section [3.1.1.2.4.8](#)).
- On originating updates, additional processing may apply if the object being reanimated is a SAM-related object (see [MS-SAMR] section [3.2.1.8](#)).

#### 3.1.1.5.4 Modify DN

##### References:

- LDAP control LDAP\_SERVER\_CROSSDOM\_MOVE\_TARGET\_OID: see section [3.1.1.3](#).
- LDAP Modify DN operation: see [\[RFC2251\]](#) section 4.9.
- Concrete structure DRS\_MSG\_MOVEREQ: see [MS-DRSR] section [4.1.15.1.1](#).
- Concrete structure DRS\_MSG\_MOVEREQ\_V2: see [MS-DRSR] section [4.1.15.1.3](#).
- Concrete structure DRS\_SecBufferDesc: see [MS-DRSR] section 5: **DRS\_SecBufferDesc**.
- Concrete structure DRS\_MSG\_MOVEREPLY: see [MS-DRSR] section [4.1.15.1.4](#).
- Concrete structure DRS\_MSG\_MOVEREPLY\_V2: see [MS-DRSR] section [4.1.15.1.6](#).
- Concrete method IDL\_DRSInterDomainMove: see [MS-DRSR] section [4.1.15](#).
- Concrete method IDL\_DRSBind: see [MS-DRSR] section [4.1.3](#).
- Function RoleObject: section [3.1.1.5.1](#).
- Function GetWellknownObject: section [3.1.1.1.6](#).
- Kerberos delegation: [\[MS-KILE\]](#).
- LDAP errors: see [\[RFC2251\]](#) section 4.1.10.
- Glossary: global group, config NC, default NC, dsname, NC replica, **prefix table**, primary group, RID, schema NC, SID, structural class.
- Access control rights RIGHT\_DELETE, RIGHT\_DS\_DELETE\_CHILD.
- LDAP attributes: [distinguishedName](#), [groupType](#), [instanceType](#), [isCriticalSystemObject](#), [isDeleted](#), [LDAPDisplayName](#), [member](#), [msDS-NonMembers](#), [name](#), [nCName](#), [objectSid](#), [proxiedObjectName](#), [systemFlags](#), [systemOnly](#), [userAccountControl](#), [wellKnownObjects](#).
- State model attributes: [parent](#), [rdnType](#).
- LDAP classes: [classSchema](#), [crossRef](#), [infrastructureUpdate](#).

##### Constants:

- Access mask bits: RIGHT\_DELETE, RIGHT\_DS\_DELETE\_CHILD: see section [5.1](#).

- GROUP\_TYPE\_BUILTIN\_LOCAL\_GROUP, GROUP\_TYPE\_ACCOUNT\_GROUP, GROUP\_TYPE\_RESOURCE\_GROUP, GROUP\_TYPE\_APP\_QUERY\_GROUP, GROUP\_TYPE\_SECURITY\_ENABLED: see section [2.2.12](#).
- ADS\_UF\_WORKSTATION\_ACCOUNT, ADS\_UF\_INTERDOMAIN\_TRUST\_ACCOUNT: see [MS-DRSR] section 5: **userAccountControl Bits**.
- GUID\_INFRASTRUCTURE\_CONTAINER\_W, GUID\_SYSTEMS\_CONTAINER\_W: see section [7.1.1.4](#).

The Modify DN originating update operation modifies the DN of the object.

The requestor supplies the following data:

- *OldDN*: DN of the object being modified by the Modify DN operation.
- *NewRDN*: RDN that will form the left most component of the new name of the object.
- *NewParentDN*: DN of the object that becomes the immediate superior of the object.
- *DeleteOldRDN*: Boolean value that says whether the old RDN value should be retained. True means the old RDN value should NOT be retained.

Let *NewDN* be the DN of the renamed object. The value *NewDN* is *NewParentDN* preceded by *NewRDN*.

#### Definitions:

Let O be the object such that O![distinguishedName](#) = OldDN.

Let P be O![parent](#).

If NewParentDN = NULL then NP is O![parent](#).

Otherwise, let NP be an object such that NP![distinguishedName](#) = NewParentDN.

The originating update is a rename operation if P = NP.

The originating update is a move operation if P ≠ NP.

### 3.1.1.5.4.1 Intra Domain Modify DN

For originating updates, if the requestor does not specify LDAP\_SERVER\_CROSSDOM\_MOVE\_TARGET\_OID **LDAP** control in the Modify DN request then server interprets the update as an intra-domain Modify DN operation. Replicated updates are always interpreted as intra-domain Modify DN operations. The request must have the LDAP\_SERVER\_CROSSDOM\_MOVE\_TARGET\_OID control (see section [3.1.1.3.4.1.2](#)) if the requestor intends to perform a cross **domain** move operation. Cross **domain** move is not supported by AD/LDS.

#### 3.1.1.5.4.1.1 Security Considerations

For originating updates, the requestor must have all permissions stated below to perform Modify DN operation. If the security check does not succeed then server returns **LDAP** error *insufficientAccessRights*.

The security context of the requestor must be granted rights RIGHT\_DS\_WRITE\_PROPERTY permission on O![name](#) to perform move or rename operation.

For move operation, the requestor must be granted right `RIGHT_DS_CREATE_CHILD` on *NP* for the [objectClass](#) of the object being added. *NP* must be visible to the requestor.

For move operation, the requestor must be granted rights `RIGHT_DELETE` on *O* or must be granted right `RIGHT_DS_DELETE_CHILD` on *P*.

In AD/DS, if *O* is within the config NC or schema NC and the RM control field of the security descriptor of the object has the `SECURITY_PRIVATE_OBJECT` bit set, then the requestor must be the owner of the object to perform this operation.

No **access check** is performed for replicated updates.

### 3.1.1.5.4.1.2 Constraints

For originating updates, the following constraints must be satisfied for the Modify DN operation. These constraints are not enforced for replicated updates.

- *DeleteOldRDN* = true. Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- *OldDN* ≠ NULL and *NewRDN* ≠ NULL. Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- All naming constraints on *NewRDN* must be satisfied. This is explained in section [3.1.1.3.1.2](#).
- *O* is present. Otherwise, server returns error *noSuchObject* / *ERROR\_DS\_OBJ\_NOT\_FOUND*.
- *NP* is present. Otherwise, server returns error *other* / *ERROR\_DS\_NO\_PARENT\_OBJECT*.
- Both *O* and *NP* must be within the same NC Replica. Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_NO\_PARENT\_OBJECT*.
- If *O* is in System **container** then *NP* is System Container or an object inside System Container. Otherwise, server returns error *other* / *ERROR\_DS\_DISALLOWED\_IN\_SYSTEM\_CONTAINER*.
- If *NP* is in System **container** then *O* is in System Container. Otherwise, server returns error *other* / *ERROR\_DS\_DISALLOWED\_IN\_SYSTEM\_CONTAINER*.
- Let *C* be the [classSchema](#) object of most specific structural class of *O*. (*C!*[LDAPDisplayName](#) ≠ [trustedDomain](#)) and (*C!*[LDAPDisplayName](#) ≠ [secret](#)). Otherwise, server returns **LDAP** error *unwillingToPerform*.
- *O!*[isDeleted](#) ≠ true. Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- *O* must not be NC root. Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- If (*O* is in config NC) and (operation is rename) then (*O!*[systemFlags](#) & `FLAG_CONFIG_ALLOW_RENAME` ≠ 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- If (*O* is in config NC) and (operation is move) then either (*O!*[systemFlags](#) & `FLAG_CONFIG_ALLOW_MOVE` ≠ 0) or ((*O!*[parent](#))!parent before and after move is the same) and (*O!*[systemFlags](#) & `FLAG_CONFIG_ALLOW_LIMITED_MOVE` ≠ 0)). Otherwise, server returns **LDAP** error *unwillingToPerform*. The `FLAG_CONFIG_ALLOW_LIMITED_MOVE` flag is used to move server objects between site **containers**.

- If (operation is move) and (*O* is in schema NC) then server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- If (operation is rename) and (*O* is in schema NC) then (*O*![systemFlags](#) & FLAG\_SCHEMA\_BASE\_OBJECT = 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- If (*O* is in **domain** NC) and (operation is rename) and (attribute *O*![systemFlags](#) is present) then (*O*![systemFlags](#) & FLAG\_DOMAIN\_DISALLOW\_RENAME = 0). Otherwise, server returns **LDAP** error *unwillingToPerform*.
- If (*O* is in **domain** NC) and (operation is move) and (attribute *O*![systemFlags](#) is present) then (*O*![systemFlags](#) & FLAG\_DOMAIN\_DISALLOW\_MOVE = 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- The object class of *O* must satisfy possSuperior schema constraint for the [objectClass](#) of *NP*. Schema constraints are explained in Restrictions on schema extensions in section [3.1.1.2](#).
- If (operation is move) then (there exists no object *CC* such that *CC*![parent](#) = *NP* and *CC*![name](#) = *O*![name](#)). Otherwise, server returns error *entryAlreadyExists* / *ERROR\_DS\_OBJ\_STRING\_NAME\_EXISTS*.

#### 3.1.1.5.4.1.3 Processing Specifics

- If the operation is move, set *O*![parent](#) to the [objectGUID](#) of the new parent object *NP*.
- Let *A* be the attribute on *O* equal to *O*![rdnType](#). Set *O*!*A* to *newRDN*.
- Set *O*![name](#) to *newRDN*.

#### 3.1.1.5.4.2 Cross Domain Move

The Modify DN LDAP request must have LDAP\_SERVER\_CROSSDOM\_MOVE\_TARGET\_OID control to indicate that the requestor intends to perform cross **domain** move operation. Cross **domain** move is not supported by AD/LDS.

The controlValue field of LDAP\_SERVER\_CROSSDOM\_MOVE\_TARGET\_OID control has the DNS hostname of the target DC that must be used as a helper to perform cross **domain** move. If the DNS hostname is not specified in the controlValue field of the **LDAP** control, then the server will only perform constraint check as explained in section [3.1.1.3](#).

#### 3.1.1.5.4.2.1 Security Considerations

The requestor must have all permissions stated below to perform cross **domain** move operation. If the security check does not succeed then server returns **LDAP** error *insufficientAccessRights*.

For move operation, the requestor must be granted rights RIGHT\_DELETE on *O* or must be granted right RIGHT\_DS\_DELETE\_CHILD on *P*.

The requestor must have performed a Kerberos **LDAP** bind with delegation enabled (see [RFC4120](#) section 2.8). Delegation should be enabled because the server impersonates the requestor when it contacts the target DC to perform cross **domain** move. If Kerberos delegation is not enabled on **LDAP** connection then server returns **LDAP** error *inappropriateAuthentication*.

### 3.1.1.5.4.2.2 Constraints

Following constraints must be satisfied for the Modify DN operation.

- *DeleteOldRDN* = true. Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- *OldDN* ≠ NULL and *NewRDN* ≠ NULL and *NewParentDN* ≠ NULL. Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- (O![systemFlags](#) & FLAG\_DISALLOW\_DELETE = 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION*.
- *IsEffectiveRoleOwner*(RoleObject(default NC, RidAllocationMaster)) = true. Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_INCORRECT\_ROLE\_OWNER*. This constraint is enforced to avoid conflicting cross **domain** move operation.
- Let C be the [classSchema](#) object of most specific structural class of O. C![systemOnly](#) = false. Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_CANT\_DELETE*.
- C![LDAPDisplayName](#) must not be any of the following. Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_XDOM\_MOVE\_OPERATION*.
  - [addressBookContainer](#)
  - [attributeSchema](#)
  - [builtinDomain](#)
  - [certificationAuthority](#)
  - [classSchema](#)
  - [configuration](#)
  - [cRLDistributionPoint](#)
  - [crossRef](#)
  - [crossRefContainer](#)
  - [dMD](#)
  - [domain](#)
  - [dSA](#)
  - [foreignSecurityPrincipal](#)
  - [infrastructureUpdate](#)
  - [linkTrackObjectMoveTable](#)
  - [linkTrackOMTEntry](#)
  - [linkTrackVolEntry](#)
  - [linkTrackVolumeTable](#)

- [lostAndFound](#)
- [nTDSConnection](#)
- [nTDSDSA](#)
- [nTDSSiteSettings](#)
- [rIDManager](#)
- [rIDSet](#)
- [samDomain](#)
- [samDomainBase](#)
- [samServer](#)
- [site](#)
- [siteLink](#)
- [siteLinkBridge](#)
- [sitesContainer](#)
- [subnet](#)
- [subnetContainer](#)
- [trustedDomain](#)
- (O![systemFlags](#) & FLAG\_DOMAIN\_DISALLOW\_MOVE = 0). Otherwise, server returns error *unwillingToPerform* / *DIRERR\_ILLEGAL\_MOD\_OPERATION*.
- (O![isCriticalSystemObject](#) ≠ true). Otherwise, server returns error *unwillingToPerform* / *DIRERR\_ILLEGAL\_MOD\_OPERATION*.
- (O![userAccountControl](#) & ADS\_UF\_SERVER\_TRUST\_ACCOUNT = 0) and (O![userAccountControl](#) & ADS\_UF\_INTERDOMAIN\_TRUST\_ACCOUNT = 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_XDOM\_MOVE\_OPERATION*.
- Let *K* be the RID of SID O![objectSid](#). (*K* > 1000). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_XDOM\_MOVE\_OPERATION*.
- (O![instanceType](#) & IT\_WRITE ≠ 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_XDOM\_MOVE\_OPERATION*.
- (O![instanceType](#) & IT\_NC\_HEAD = 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_XDOM\_MOVE\_OPERATION*.
- (O![isDeleted](#) ≠ true). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_CANT\_MOVE\_DELETED\_OBJECT*.
- If (*O* is a [group](#) object) then (O![groupType](#) & GROUP\_TYPE\_BUILTIN\_LOCAL\_GROUP = 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_XDOM\_MOVE\_OPERATION*.



- If (*O* is a [group](#) object) and ((attribute *O!*[member](#) is present) or (attribute *O!*[msDS-NonMembers](#) is present)) then (*O!*[groupType](#) & GROUP\_TYPE\_ACCOUNT\_GROUP = 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_CANT\_MOVE\_ACCOUNT\_GROUP*.
- If (*O* is a group object) and ((attribute *O!*[member](#) is present) or (attribute *O!*[msDS-NonMembers](#) is present)) then (*O!*[groupType](#) & GROUP\_TYPE\_RESOURCE\_GROUP = 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_CANT\_MOVE\_RESOURCE\_GROUP*.
- If (*O* is a group object) and ((attribute *O!*[member](#) is present) or (attribute *O!*[msDS-NonMembers](#) is present)) then (*O!*[groupType](#) & GROUP\_TYPE\_APP\_BASIC\_GROUP = 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_CANT\_MOVE\_APP\_BASIC\_GROUP*.
- If (*O* is a group object) and ((attribute *O!*[member](#) is present) or (attribute *O!*[msDS-NonMembers](#) is present)) then (*O!*[groupType](#) & GROUP\_TYPE\_APP\_QUERY\_GROUP = 0). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_CANT\_MOVE\_APP\_QUERY\_GROUP*.
- If ((*O* is a [user](#) object) or (*O* is a [group](#) object)) and (*O* is a member of any global group) then (*O* is a member of only one global group and that group is its primary group). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_CANT\_WITH\_ACCT\_GROUP\_MEMBERSHPS*.
- Let *N* be the root of NC replica where *OldDN* exists. Let *R* be a [crossRef](#) object such that *R!*[nCName](#) = *N*. *R* must exist and (*R!*[systemFlags](#) & FLAG\_CR\_NTDS\_NC ≠ 0) and (*R!*[systemFlags](#) & FLAG\_CR\_NTDS\_DOMAIN ≠ 0). Otherwise, server returns error *noSuchObject* / *ERROR\_DS\_CANT\_FIND\_EXPECTED\_NC*.
- Let *NN* be the root of NC replica where *NP* exists. Let *NR* be a [crossRef](#) object such that *NR!*[nCName](#) = *NN!*[distinguishedName](#). *NR* must exist and (*NR!*[systemFlags](#) & FLAG\_CR\_NTDS\_NC ≠ 0) and (*NR!*[systemFlags](#) & FLAG\_CR\_NTDS\_DOMAIN ≠ 0). Otherwise, server returns error *noSuchObject* / *ERROR\_DS\_CANT\_FIND\_EXPECTED\_NC*.
- *R* ≠ *NR*. Otherwise, server returns error *invalidDNSyntax* / *ERROR\_DS\_SRC\_AND\_DST\_NC\_IDENTICAL*.
- Let *WKS* be a set of all attribute values for *N!*[wellKnownObjects](#). There is no attribute value *V* in *WKS* such that *V.object\_DN* = *O!*[distinguishedName](#). Otherwise, server returns error *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_XDOM\_MOVE\_OPERATION*.
- *O* has no child objects. Otherwise, server returns error *notAllowedOnNonLeaf* / *ERROR\_DS\_CHILDREN\_EXIST*.

### 3.1.1.5.4.2.3 Processing Specifics

Once the above constraint checking is done the server performs the move operation on the target DC as specified below. The server then performs cleanup operation as specified below. Constraint checking and cleanup operation are performed in two separate local transactions.

The caller specifies the DNS hostname of the target DC in the *controlValue* field of LDAP\_SERVER\_CROSSDOM\_MOVE\_TARGET\_OID **LDAP** control.

If the *controlValue* field is empty, then server performs only constraints checking as mentioned above. It returns *success* if it passes all the constraints.

#### Invoke move operation on target DC:

Let *S* be the [nTSDSA](#) object of the server.

Let *NN* be the root of NC replica where *NP* exists.

Let *pmsgIn* be a reference to a structure of type DRS\_MSG\_MOVEREQ.

Set *pmsgIn*->*V2.pSrcDSA* to dsname of *S*.

*pmsgIn*->*V2.pSrcObject* is a reference to structure of type ENTINF. Define ENTINF for *O* as described below.

Set *pmsgIn*->*V2.pDstName* to dsname of *NewDN*.

Set *pmsgIn*->*V2.pExpectedTargetNC* to dsname of *NN*.

*pmsgIn*->*V2.pClientCreds* is a reference to DRS\_SecBuffer structure. It is set to the GSS Kerberos authentication token (see [\[RFC1964\]](#)) derived from the security context of the caller.

Set *pmsgIn*->*V2.PrefixTable* to dc.prefixTable as specified in section [3.1.1.1.9](#).

Set *pmsgIn*->*V2.ulFlags* to 0.

Let *H* be the bind handle derived by calling IDL\_DRSBind method against target DC.

Let *pdwOutVersion* be a reference to *dwOutversion* of type integer.

Let *pmsgOut* be a reference to DRS\_MSG\_MOVEREPLY structure.

Call IDL\_DRSInterDomainMove(*H*, 2, *pmsgIn*, *pdwOutVersion*, *pmsgOut*). If the method returns an error then server returns **LDAP** error *unavailable*.

If (*dwOutVersion* ≠ 2) then server returns **LDAP** error *operationsError*.

If (*pmsgOut*->*v2.win32Error* ≠ 0) then server returns **LDAP** error *unwillingToPerform*.

### Create proxy object and perform cleanup

The [proxiedObjectName](#) attribute is present on the [infrastructureUpdate](#) object used to communicate the cross-domain move from the originating NC replica to other replicas of the NC. The [proxiedObjectName](#) attribute is also present on an object that has been moved across **domain**, as specified in [MS-DRSR] section [4.1.15.3](#).

The [proxiedObjectName](#) attribute has syntax Object(DN-Binary); see section [3.1.1.2.2.2.3](#) for the specification of this syntax, which contains the fields *char\_count*, *binary\_value*, and *object\_DN*. The *binary\_value* part of a [proxiedObjectName](#) value is 16 characters. Bytes 0 to 7 contain the character string "00000001" for a cross-domain move. Bytes 8 to 15 contain the hexadecimal representation of a number called the cross **domain** move epoch.

The cross **domain** move epoch *E* of the [proxiedObjectName](#) attribute on an [infrastructureUpdate](#) object is determined as follows:

- If *O!*[proxiedObjectName](#) is present then let *B* be the *binary\_value* of *O!*[proxiedObjectName](#). Let *E* be value given by the least significant 32 bits of *B*.
- Otherwise, let *E* be 0.

Create an attribute value *K* of type Object (DN-Binary). Set *K.char\_count* to 16. Let *J* be a string of 8 characters that is the hexadecimal representation of value *E*. Set *K.binary\_value* to the concatenation of the strings "00000001" and *J*. Set *object\_DN* part of *K* to *NewDN*.

Expunge object *O* from *NC replica*.

Let *I* = GetWellknownObject(default NC, GUID\_INFRASTRUCTURE\_CONTAINER\_W).

Create an [infrastructureUpdate](#) object *L* such that *L*![parent](#) = *I* and *L*![name](#) is any name unique among the children of *I* and *L*![proxiedObjectName](#) = *K* and *L*![systemFlags](#) = (FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DISALLOW\_MOVE\_ON\_DELETE | FLAG\_DOMAIN\_DISALLOW\_MOVE).

Delete *L* and turn it to a *tombstone* object.

### Defining ENTINF structure for object *O*.

Let *t* be the prefix table *dc.prefixTable* specified in section [3.1.1.1.9](#).

Let *AttsSet* be the set of all attributes (represented as ATTRTYP) of object *O*.

Let *Atts* be a sequence of ATTRTYP whose elements are elements of *AttsSet*.

Let *EntInf* be a structure of type ENTINF.

Set *EntInf.pName* to the dsname of *O*.

Set *EntInf.ulFlags* to 0.

Let *AttrBlock* be a structure of type ATTRBLOCK of length *Atts.length*.

Give *AttrBlock.pAttr[i]* a value determined by *Atts[i]* as follows, for all *i* in [0...*Atts.length*) (in any order)

- Let *K* be the [attributeSchema](#) object *SchemaObj(Atts[i])*. *SchemaObj* is specified in [MS-DRSR] section [5:SchemaObj](#).
- Let *syntax* be *K*![attributeSyntax](#).
- Let *AttrBlock.pAttr[i].AttribTyp* be the value returned by *MakeAttid(t, oid)*.
- Let *Vals* be the sequence of values *O.Atts[i]*.
- Let *AttrBlock.pAttr[i].AttrVal* be a structure of type ATTRVALBLOCK of length *Vals.length*.
- Set *AttrBlock.pAttr[i].AttrVal.valCount* = *Vals.length*.
- Give *AttrBlock.pAttr[i].AttrVal.pAVal[j]* a value determined by *Vals[j]* as follows, for all *j* in [0..*Vals.length*) (in any order).
  - Set *AttrBlock.pAttr[i].AttrVal.pAVal[j]* = *ATTRVALFromValue(Vals[j], syntax, t)*

### 3.1.1.5.5 Delete Operation

#### References

LDAP attributes: [distinguishedName](#), [isDeleted](#), [entryTTL](#), [msDS-Entry-Time-To-Die](#), [nTSecurityDescriptor](#), [attributeID](#), [attributeSyntax](#), [dnReferenceUpdate](#), [dnHostName](#), [flatName](#), [governsID](#), [groupType](#), [instanceType](#), [LDAPDisplayName](#), [legacyExchangeDN](#), [mS-DS-CreatorSID](#), [mSMOOwnerID](#), [nCName](#), [objectClass](#), [objectGUID](#), [objectSid](#), [oMSyntax](#), [proxiedObjectName](#), [name](#), [replPropertyMetaData](#), [sAMAccountName](#), [securityIdentifier](#), [sIDHistory](#), [subClassOf](#), [systemFlags](#), [trustPartner](#), [trustDirection](#), [trustType](#), [trustAttributes](#), [userAccountControl](#), [uSNChanged](#), [uSNCreated](#), [whenCreated](#), [searchFlags](#), [isCriticalSystemObject](#), [objectCategory](#), [sAMAccountType](#), [isDeleted](#), [lastKnownParent](#).

State model attributes: [rdnType](#)

LDAP classes: [dynamicObject](#), [crossRef](#).

#### Constants

- Win32/status error codes: ERROR\_DS\_REFERRAL, ERROR\_DS\_ILLEGAL\_MOD\_OPERATION, ERROR\_DS\_CHILDREN\_EXIST, ERROR\_DS\_TREE\_DELETE\_NOT\_FINISHED
- Access mask bits, CARs: SECURITY\_PRIVATE\_OBJECT, RIGHT\_DELETE, RIGHT\_DS\_DELETE\_CHILD, RIGHT\_DS\_DELETE\_TREE
- Security privileges:
- [systemFlags](#) bits: FLAG\_DISALLOW\_DELETE, FLAG\_DISALLOW\_MOVE\_ON\_DELETE
- Schema bits: fPRESERVEONDELETE
- LDAP:

The delete operation results in transformation of an existing object in the directory tree into a tombstone. The requestor supplies the DN of the object to be removed.

A *tombstone* is a special placeholder object that replicates around, signaling replica partners that the original object was deleted. Tombstones are invisible to **LDAP** searches by default, so for an **LDAP** application, it appears that the object was physically removed from the directory after a delete operation has taken place. Tombstones are distinguished from regular objects by the presence of [isDeleted](#) attribute with the value true. After some time, the tombstone is removed from the directory.

Normally, only leaf objects (objects without descendants in the directory tree) may be deleted. There is also a special tree-delete operation, with which whole trees of objects are removed (see Tree-delete operation in section [3.1.1.5.5.7](#) below).

In most cases, upon deletion, a tombstone is moved into the Deleted Objects **container** of its NC; for exceptions see section [3.1.1.5.5.6](#). The RDN of the object is changed to a "delete-mangled RDN" - an RDN that is guaranteed to be unique within the Deleted Objects **container**. If *O* is the object that is deleted, the delete-mangled RDN is the concatenation of *O!*[name](#), the character with value 0x0A, the string "DEL:", and the dashed string representation ([\[RFC4122\]](#) section 3) of *O!*[objectGUID](#). A "delete-mangled DN" is a DN such that the leaf RDN is a delete-mangled RDN.

An object whose class is defunct, or whose class is active but some of whose attributes are defunct, can still be deleted.

#### 3.1.1.5.5.1 Tombstone Invariants

The following invariants are maintained for the deleted objects (tombstones):

- [isDeleted](#) attribute is set to true on tombstones.
- The tombstone retains the [objectGUID](#) and [objectSid](#) (if any) attributes of the original object.
- The tombstone does not have descendant objects.
- The tombstone remains in the database and is available for outbound replication for at least the tombstone lifetime time interval (see section [7.1.1](#)) after its deletion, with the following exception:
  - The Deleted Objects **container** (which is itself a tombstone) always remains available.

- A tombstone does not have outgoing linked attribute values (that is, it may not point to other objects via a linked attribute).
- NC replicas do not contain linked references to tombstones. In other words, when an object is deleted, any linked attribute values pointing to it are also removed.
- The non-linked references to the object are retained when the object is deleted and is converted into a tombstone (i.e., other objects that pointed to the original object via non-linked DN references continue pointing to the tombstone indefinitely after the object is deleted, until these values are removed explicitly).
- A protected object may not be deleted (see Protected Objects in section [3.1.1.5.5.3](#) below).

### 3.1.1.5.5.2 dynamicObject Invariants

See section [7.1.7](#).

### 3.1.1.5.5.3 Protected Objects

The following objects are considered protected and may not be deleted:

- The DC's [nTDSDSA](#) object and all of its ancestors
- The DC's [rIDSet](#) object and all of its ancestors. A DC's [rIDSet](#) object is the referent of the [rIDSetReferences](#) attribute of the DC's Domain Controller object (section [7.1.1.3.1](#)).
- The [crossRef](#) objects corresponding to the DC's config, schema and default **domain** NCs.

### 3.1.1.5.5.4 Security Considerations

For originating updates, the requestor must have the following permissions. No permissions are required for replicated updates.

The object being deleted must be visible to the requestor.

For regular delete operation, at least one of the following permissions must be granted to the requestor:

- RIGHT\_DELETE on the object being deleted, or
- RIGHT\_DS\_DELETE\_CHILD on the parent of the object being deleted, when the object is not a NC root.

### 3.1.1.5.5.5 Constraints

For originating updates, the following constraints are enforced for the delete operation. If any of them are not satisfied, the server returns *unwillingToPerform* / *ERROR\_DS\_ILLEGAL\_MOD\_OPERATION* (unless specified otherwise). These constraints are not enforced for replicated updates.

- The object resides in a writable NC replica, otherwise the delete returns *referral* / *ERROR\_DS\_REFERRAL*.
- If the object being modified is in config NC or schema NC, and the RM control of the SD is present and contains SECURITY\_PRIVATE\_OBJECT bit, then additional requirements on the DC performing the operation are enforced (if neither is true, then *referral* / *ERROR\_DS\_REFERRAL* must be returned):

- the DC must be a member of the root **domain** in the forest, or
- the DC must be a member of the same **domain** where the current object owner belongs.
- If FLAG\_DISALLOW\_DELETE bit is set in the [systemFlags](#) attribute, then *unwillingToPerform* / *ERROR\_DS\_CANT\_DELETE* is returned.
- Tombstones may not be deleted.
- If the object has descendants, then the delete operation fails with *unwillingToPerform* / *ERROR\_DS\_CHILDREN\_EXIST*.
- Objects that are children of the CN=Partitions child of the config NC root ([crossRef](#) objects) may only be deleted when the DC is the **domain** naming FSMO, otherwise *referral* / *ERROR\_DS\_REFERRAL* is returned.
- Objects in the schema NC may not be deleted.
- If the object being modified is a SAM-specific object (section [3.1.1.5.2.3](#)), then additional constraints apply (see [MS-SAMR] section [3.2.1.6](#)).

### 3.1.1.5.5.6 Processing Specifics

The following processing rules apply to the delete operation:

- For originating updates, the RDN for the tombstone is the object's delete-mangled RDN, as specified in Delete Operation in section [3.1.1.5](#). For replicated updates, the received RDN for the tombstone is set on the object.
- All attribute values are removed from the object, with the following exceptions:
  - [nTSecurityDescriptor](#), [attributeID](#), [attributeSyntax](#), [dNReferenceUpdate](#), [dNSHostName](#), [flatName](#), [governsID](#), [groupType](#), [instanceType](#), [LDAPDisplayName](#), [legacyExchangeDN](#), [msDS-CreatorSID](#), [mSMOOwnerID](#), [nCName](#), [objectClass](#), [distinguishedName](#), [objectGUID](#), [objectSid](#), [oMSyntax](#), [proxiedObjectName](#), [name](#), [replPropertyMetaData](#), [sAMAccountName](#), [securityIdentifier](#), [sIDHistory](#), [subClassOf](#), [systemFlags](#), [trustPartner](#), [trustDirection](#), [trustType](#), [trustAttributes](#), [userAccountControl](#), [uSNChanged](#), [uSNCreated](#), [whenCreated](#) attribute values are retained.
  - In AD/LDS, the [msDS-PortLDAP](#) attribute is also retained.
  - The attribute that equals the [rdnType](#) of the object (for example, [cn](#) for a [user](#) object) is retained.
  - Any attribute that has fPRESERVEONDELETE flag set in its [searchFlags](#) is retained, except [objectCategory](#) and [sAMAccountType](#), which are always removed, regardless of the value of their [searchFlags](#).
- All outgoing linked attribute values are removed.
- All incoming linked attribute values are removed.
- [isDeleted](#) attribute is set to true.
- For originating updates, [lastKnownParent](#) attribute value is set to the DN of the current parent object.

- The object is moved into the Deleted Objects **container** in its NC, except in the following scenarios, when it must remain in its current place:
  - The object is an NC root.
  - The object's [systemFlags](#) value has FLAG\_DISALLOW\_MOVE\_ON\_DELETE bit set.
- For originating updates, additional operations may be performed if the object being modified is a SAM-specific object (section [3.1.1.5.2.3](#)); see [MS-SAMR] section [3.2.1.8](#)).

### 3.1.1.5.5.7 Tree-delete Operation

The tree-delete operation is a special mode of delete operation that simplifies deleting trees of objects. The regular delete operation can only delete leaf objects. The tree-delete operation processes a tree of objects one-by-one, deleting objects starting from the leaf objects, and continuing up until the root can be deleted.

A tree-delete operation is never performed as a replicated update.

#### 3.1.1.5.5.7.1 Tree-delete Security Considerations

The object being deleted (root of the tree) must be visible to the requestor, in the sense of section [5.1.3.3.6](#).

The requestor must have the RIGHT\_DS\_DELETE\_TREE on the object being deleted. Note, no additional permissions are needed on the descendants of the object.

#### 3.1.1.5.5.7.2 Tree-delete Constraints

- All regular delete operation constraints apply on each object being deleted.
- The tree-delete operation may not be applied to an NC root.
- Objects with [isCriticalSystemObject](#) attribute equal true may not be deleted by the tree-delete operation (this also applies to objects in the subtree being deleted). This constraint is checked object-by-object and deletion stops if some deletion would violate this constraint. Because, as explained in the next section, deleted objects never have children, the result after deletion stops due to this constraint is a tree. The resultant tree may not be the same as the original tree because some objects may have been deleted prior to the failure.

#### 3.1.1.5.5.7.3 Tree-delete Processing Specifics

- The tree-delete operation proceeds by removing the tree starting from the leaf objects and making its way to the root of the tree. The order of processing is not important, as long as each node is only deleted after all of its descendants have been deleted and moved into Deleted Objects **container** (section [7.1.1.4.2](#)).
- Regular delete processing specifics apply to each object being deleted.
- The tree-delete operation is implemented using multiple transactions.
- It is allowed for the tree-delete operation not to delete the complete subtree. If the server failed to complete the tree-delete operation and the error is recoverable (i.e. no user intervention is required), it returns a special error code *adminLimitExceeded / ERROR\_DS\_TREE\_DELETE\_NOT\_FINISHED* to the user. However, it is required that at least one object in the subtree was deleted (i.e., some progress was made). The clients continue repeating

the tree-delete request until they either receive a *success* (indicating that the tree was successfully removed) or receive an error code other than `ERROR_DS_TREE_DELETE_NOT_FINISHED` (as specified in section [3.1.1.5.5](#)).

### 3.1.1.6 Background Tasks

In AD/DS, the server runs background tasks periodically to:

- Protect **security principals** that have elevated administrative privilege and
- Maintain referential integrity (see Referential integrity in section [3.1.1.1](#)) on object references.

These tasks are specified below.

References:

- Special Objects in section [7.1](#): Windows NT

Glossary Terms: Active Directory, **security principals**, privileges, ownership, audited operations, PDC, FSMO, security descriptor, transitive membership, member of, RID

LDAP attributes: [nTSecurityDescriptor](#), [groupType](#), [objectClass](#), [member](#), [objectSid](#), [dSHeuristics](#)

LDAP classes: [container](#), [user](#), [group](#)

Constants

- Access mask bits, CARs:
- [groupType](#) bits: `GROUP_TYPE_SECURITY_ENABLED`
- Constant RIDs: `DOMAIN_ALIAS_RID_ADMINS`, `DOMAIN_ALIAS_RID_ACCOUNT_OPS`, `DOMAIN_ALIAS_RID_SYSTEM_OPS`, `DOMAIN_ALIAS_RID_PRINT_OPS`, `DOMAIN_ALIAS_RID_BACKUP_OPS`, `DOMAIN_ALIAS_RID_REPLICATOR`, `DOMAIN_GROUP_RID_SCHEMA_ADMINS`, `DOMAIN_GROUP_RID_ADMINS`, `DOMAIN_GROUP_RID_CONTROLLERS`, `DOMAIN_USER_RID_KRBTGT`, `DOMAIN_USER_RID_ADMIN`

#### 3.1.1.6.1 AdminSDHolder

If a security principal object with elevated administrative privileges in Active Directory has a weak security descriptor, Active Directory is vulnerable to straightforward attack. Therefore Active Directory protects the security descriptors of such objects from updates that might give them weak security descriptors.

Each security principal is represented as an object *o* in Active Directory. For every *o* there is an attribute *o!*[nTSecurityDescriptor](#). The value is the security descriptor defining ownership, permissions, and audited operations for *o*.

Active Directory protects the security descriptor on certain objects by periodically overwriting any changes. This mechanism loosely establishes an upper bound on the length of time that a protected object may have a weak security descriptor.



### 3.1.1.6.1.1 Authoritative Security Descriptor

The security descriptor that is written to protected objects is stored in the [nTSecurityDescriptor](#) attribute on the AdminSDHolder object in Active Directory. The AdminSDHolder object is of class [container](#) and has a DN of "CN=AdminSDHolder,CN=System,<Domain NC DN>".

### 3.1.1.6.1.2 Protected Objects

In **domain** d, the set S of all security principal objects o that are protected is defined as follows:

- (o![objectClass](#) = [group](#) AND attribute o![groupType](#) & GROUP\_TYPE\_SECURITY\_ENABLED ≠ 0) OR (o![objectClass](#) = [user](#))
- AND (o![objectSid](#) = d![objectSid](#) + RID)
- AND either
  - o is a member, directly or transitively, of any group in the set:
    - built-in well-known group with RID = DOMAIN\_ALIAS\_RID\_ADMINS
    - built-in well-known group with RID = DOMAIN\_ALIAS\_RID\_ACCOUNT\_OPS
    - built-in well-known group with RID = DOMAIN\_ALIAS\_RID\_SYSTEM\_OPS
    - built-in well-known group with RID = DOMAIN\_ALIAS\_RID\_PRINT\_OPS
    - built-in well-known group with RID = DOMAIN\_ALIAS\_RID\_BACKUP\_OPS
    - built-in well-known group with RID = DOMAIN\_ALIAS\_RID\_REPLICATOR
    - **account domain** well-known group with RID = DOMAIN\_GROUP\_RID\_ADMINS
    - **account domain** well-known group with RID = DOMAIN\_GROUP\_RID\_SCHEMA\_ADMINS
    - **account domain** well-known group with RID = DOMAIN\_GROUP\_RID\_ENTERPRISE\_ADMINS
  - OR, is one of the following well-known **security principals**:
    - of class [user](#) with RID = DOMAIN\_USER\_RID\_ADMIN
    - of class [user](#) with RID = DOMAIN\_USER\_RID\_KRBTGT
    - of class [group](#) with RID = DOMAIN\_GROUP\_RID\_CONTROLLERS
    - of class [group](#) with RID = DOMAIN\_GROUP\_RID\_READONLY\_CONTROLLERS

### 3.1.1.6.1.3 Protection Operation

Every object in the protected set is examined at least once every 120 minutes, every 60 minutes by default, at **domain** d's PDC FSMO role owner. For any object o where o![nTSecurityDescriptor](#) ≠ AdminSDHolder![nTSecurityDescriptor](#) an originating write is performed replacing o![nTSecurityDescriptor](#) with the value of AdminSDHolder![nTSecurityDescriptor](#). Other replicas of **domain** d see the effects of this operation after a delay due to replication.

#### 3.1.1.6.1.4 Configurable State

Let *C* be the object in the config NC identified by the DN of "CN=Windows NT,CN=Services,CN=Configuration,<forest root DN>". *C!*[dSHeuristics](#) (section [7.1.1.2.4.1.2](#)) is a **Unicode** string attribute, in which the 16<sup>th</sup> character, *dwAdminSDExMask*, can optionally be set to cause the protection operation to exclude one or more protected objects.

The valid values of *dwAdminSDExMask* are the characters '0'-'9' and 'a'-'f'. The value is interpreted as a hex digit, of which each bit represents a specific set of **security principals** that is to be excluded from the AdminSDHolder protection operation.

The set of security principal objects that are excluded are a member, directly or transitively, of any group in the set defined by bits set in the list below:

- *C!*[dSHeuristics](#)[15] & 0x1 ≠ 0 then DOMAIN\_ALIAS\_RID\_ACCOUNT\_OPS
- *C!*[dSHeuristics](#)[15] & 0x2 ≠ 0 then DOMAIN\_ALIAS\_RID\_SYSTEM\_OPS
- *C!*[dSHeuristics](#)[15] & 0x4 ≠ 0 then DOMAIN\_ALIAS\_RID\_PRINT\_OPS
- *C!*[dSHeuristics](#)[15] & 0x8 ≠ 0 then DOMAIN\_ALIAS\_RID\_BACKUP\_OPS

#### 3.1.1.6.2 Reference Update

##### References:

- Variable: *dsname*
- LDAP attributes: [dNReferenceUpdate](#).
- LDAP classes: [infrastructureUpdate](#).
- Glossary: *dsname*, Infrastructure FSMO master, NC replica, tombstone, GC.
- IDL\_DRSVerifyNames method: see [MS-DRSR] section [4.1.27](#).
- Well-known Objects

In AD/DS, attributes of attribute syntax Object (DS-DN), Object(DN-String), Object(DN-Binary), Object(Access-Point) and Object(OR-Name) can have attribute values that reference objects in an NC for which no NC replica is present on the server. The server does not get a replicated update when an object in the NC replica not present on the server is modified or deleted. In such a case, references to such objects will remain to an old *dsname* on the server. In order to update these kinds of references, the Infrastructure FSMO master runs a background task called reference update at regular intervals. By default, each reference is examined every two days.

The reference update task does processing as below:

For each object *P* in each NC replica on the server do the following:

- Let *S* be the set of all attributes of *P* with attribute syntax Object(DS-DN), Object(DN-String), Object(DN-Binary), Object(OR-Name) and Object(Access-Point).
- For each attribute *A* in set *S* and for each value *V* of *A* do the following:
  - If there exists an object with *dsname V* in any NC replica on this DC, then skip this value *V*.
  - If attribute syntax of *A* is Object(DS-DN) then let *G* be *P.A.V.guid\_value*. Let *D* be *P.A.V.dn*.

- Otherwise, let *G* be *P.A.V.object\_DN.guid\_value*. Let *D* be *P.A.object\_DN.dn*.
- Retrieve the dsname *N* of object with [objectGUID](#) *G* from a GC by calling method IDL\_DRSVerifyNames. IDL\_DRSVerifyNames is explained in [MS-DRSR] section 4.1.27.
- If *N!name* ≠ *D* then create an [infrastructureUpdate](#) object *I* in the well-known infrastructure update container (see section [7.1.1.4](#)). Set *I!dNReferenceUpdate* to *N*. Delete *I* immediately to turn it to a tombstone.

Creation of an [infrastructureUpdate](#) object *K* with attribute [dNReferenceUpdate](#) will trigger an update of all references to dsnames corresponding to *K!dNReferenceUpdate* as explained in section [3.1.1.5.2.4](#).

### 3.1.1.7 NT4 Replication Support

AD/DS supports the NT4 replication protocol as specified in [MS-NRPC] section 3.5 by maintaining two variables: *nt4ReplicationState* and *pdnChangeLog*. These variables are referenced by [MS-DRSR] section [4.1.11.3](#) in order to specify the IDL\_DRSGetNT4ChangeLog method. This section normatively specifies the format of these variables and how they are maintained during state changes in AD/DS.

This section also normatively specifies the format of the referent of the pmsgOut.V1.pLog field of the DRS\_MSG\_NT4\_CHGLOG\_REPLY\_V1 response message of the IDL\_DRSGetNT4ChangeLog method [MS-DRSR] section 4.1.11.3.

#### 3.1.1.7.1 Format of nt4ReplicationState and pdnChangeLog

##### 3.1.1.7.1.1 nt4ReplicationState

*nt4ReplicationState* is a tuple containing the following fields:

**SamNT4ReplicationUSN:** this field, a signed 64-bit value, is an update sequence number for updates that occur in AD/DS that are relevant to the NT4 replication protocol. Relevant updates are described in section [3.1.1.7.2.2](#)

**SamCreationTime:** this field, a FILETIME, records the timestamp when SamNT4ReplicationUSN is set to one.

**BuiltinNT4ReplicationUSN:** this field, a signed 64-bit value, is an update sequence number for updates that occur in AD/DS that are relevant to the NT4 replication protocol. It is different from SamNT4ReplicationUSN in that this value is used only to identify changes to objects whose [objectSid](#) has the **domain prefix** of the built-in **domain** SID.

**BuiltinCreationTime:** this field, a FILETIME, is used to record the timestamp when BuiltinNT4ReplicationUSN is set to one.

##### 3.1.1.7.1.2 pdnChangeLog

The variable *pdnChangeLog* maintains a sequence of elements, each representing a unique update to Active Directory that is exposed through the NT4 replication protocol ([\[MS-NRPC\]](#) section 3.5).

Though *pdnChangeLog* is an internal variable, its contents are sent over the network.

The *pdnChangeLog* variable is a sequence of CHANGELOG\_ENTRY elements. These CHANGELOG\_ENTRY elements are defined in [\[MS-NRPC\]](#) section 3.5.4.5.4.

### 3.1.1.7.2 State Changes

This section describes state changes in AD/DS that cause the *nt4ReplicationState* and *pdchangeLog* variables to change values.

#### 3.1.1.7.2.1 Initialization

*nt4ReplicationState* and *pdchangeLog* are reset on Active Domain **domain** creation (for example, when the first DC in an AD/DS **domain** is installed). See section [3.1.1.7.2.4](#) on resetting the *pdchangeLog* for the specific values of the variables in this condition.

#### 3.1.1.7.2.2 Directory Updates

Entries are added to the *pdchangeLog* on select directory updates, specified here. The *pdchangeLog* is maintained as a circular buffer - once an implementation-specific size limit (64K bytes) is exceeded, the least-recently-added entries are removed to make room for new entries.

If the following condition is true during a directory update then the following action occurs:

1. Condition

1. The update, create, or delete occurs within the **domain** NC (both for an originating and replicated update).
2. The AD/DS **domain** is in **mixed mode**.
3. A condition listed in the Trigger Condition Tables (below) matches the update.

2. Action

- An entry is added to *pdchangeLog* with the associated fields in the Trigger Condition Tables that satisfied condition 1.c. The remaining fields in the *pdchangeLog* entry are as follows:
  1. If the [objectSid](#) attribute value of the object being updated has a **domain** prefix of the built-in **domain** SID, then DbIndex is 0x1; otherwise, DbIndex is 0x0.
  2. The SerialNumber field is set as follows:
    1. If DbIndex is 0x0, SamNT4ReplicationUSN is incremented by one and the resulting value is used for the SerialNumber field.
    2. If DbIndex is 0x1, BuiltinNT4ReplicationUSN is incremented by one and the resulting value is used for the SerialNumber field.
  3. The SID field is not specified.

**Trigger Condition Tables:** Database Triggers for *pdchangeLog* update

- **Trigger Condition:** An update occurs to one or more of the attributes specified in Table A on a **domain** object or **builtin domain** object.

<i>pdchangeLog</i> entry	
Field	Value
RelativeId	0x0

<b><i>pdchangeLog</i> entry</b>	
<b>Field</b>	<b>Value</b>
Flags	CHANGELOG_SID
DeltaType	AddOrChangeDomain

- **Trigger Condition:** A [group](#) object creation or update to one or more of the attributes specified in Table B occurs when the [groupType](#) attribute is GROUP\_TYPE\_ACCOUNT\_GROUP.

<b><i>pdchangeLog</i> entry</b>	
<b>Field</b>	<b>Value</b>
RelativeId	RelativeId of the <a href="#">objectSid</a> attribute value
Flags	CHANGELOG_SID
DeltaType	AddOrChangeGroup
Name	<a href="#">sAMAccountName</a> attribute value

- **Trigger Condition:** A [group](#) object creation or update to one or more of the attributes specified in Table B occurs when the [groupType](#) attribute is GROUP\_TYPE\_RESOURCE\_GROUP.

<b><i>pdchangeLog</i> entry</b>	
<b>Field</b>	<b>Value</b>
RelativeId	RelativeId of the <a href="#">objectSid</a> attribute value
Flags	CHANGELOG_SID
DeltaType	AddOrChangeAlias
Name	<a href="#">sAMAccountName</a> attribute value

- **Trigger Condition:** A [user](#) object creation or update to one of more of the attribute specified in Table C occurs.

<b><i>pdchangeLog</i> entry</b>	
<b>Field</b>	<b>Value</b>
RelativeId	RelativeId of the <a href="#">objectSid</a> attribute value
Flags	CHANGELOG_SID
DeltaType	AddOrChangeUser
Name	<a href="#">sAMAccountName</a> attribute value

- **Trigger Condition:** A [group](#) object deletion whose [groupType](#) attribute value is GROUP\_TYPE\_ACCOUNT\_GROUP occurs.

<b><i>pdchangeLog</i> entry</b>	
<b>Field</b>	<b>Value</b>
RelativeId	RelativeId of the <a href="#">objectSid</a> attribute value
Flags	0x8
DbType	DeleteGroup
Name	<a href="#">sAMAccountName</a> attribute value

- **Trigger Condition:** A [group](#) object deletion whose [groupType](#) attribute value is GROUP\_TYPE\_RESOURCE\_GROUP occurs.

<b><i>pdchangeLog</i> entry</b>	
<b>Field</b>	<b>Value</b>
RelativeId	RelativeId of the <a href="#">objectSid</a> attribute value
Flags	CHANGELOG_SID
DeltaType	DeleteAlias
Name	<a href="#">sAMAccountName</a> attribute value

- **Trigger Condition:** A [user](#) object deletion occurs.

<b><i>pdchangeLog</i> entry</b>	
<b>Field</b>	<b>Value</b>
RelativeId	RelativeId of the <a href="#">objectSid</a> attribute value
Flags	CHANGELOG_SID
DeltaType	DeleteUser
Name	<a href="#">sAMAccountName</a> attribute value

**Table A: Domain Attributes for NT4 Replication**

<b>Attributes</b>
<a href="#">nTSecurityDescriptor</a>
<a href="#">oEMInformation</a>
<a href="#">minPwdLength</a>
<a href="#">pwdHistoryLength</a>
<a href="#">pwdProperties</a>
<a href="#">maxPwdAge</a>

Attributes
<a href="#">minPwdAge</a>
<a href="#">lockoutDuration</a>
<a href="#">lockOutObservationWindow</a>
<a href="#">lockoutThreshold</a>

**Table B: Group Attributes for NT4 Replication**

Attributes
<a href="#">nTSecurityDescriptor</a>
<a href="#">sAMAccountName</a>
<a href="#">description</a>
<a href="#">member</a>

**Table C: User Attributes for NT4 Replication**

Attributes
<a href="#">sAMAccountName</a>
<a href="#">displayName</a>
<a href="#">primaryGroupID</a>
<a href="#">description</a>
<a href="#">comment</a>
<a href="#">homeDirectory</a>
<a href="#">homeDrive</a>
<a href="#">scriptPath</a>
<a href="#">profilePath</a>
<a href="#">userWorkstations</a>
<a href="#">logonHours</a>
<a href="#">accountExpires</a>
<a href="#">userAccountControl</a>
<a href="#">userParameters</a>
<a href="#">countryCode</a>
<a href="#">codePage</a>

Attributes
<a href="#">pwdLastSet</a>
<a href="#">unicodePwd</a>
<a href="#">dBCSPwd</a>
<a href="#">nTSecurityDescriptor</a>
<a href="#">groupType</a>

### 3.1.1.7.2.3 Acquiring the PDC Role

When the PDC role is acquired through a FSMO role transfer, one of the following two predicates is true following the transfer:

1. The new PDC's *pdcChangeLog* is in the reset state described in section [3.1.1.7.2.4](#)
2. All of the following are true:
  1. The new PDC's *pdcChangeLog* has the same ordering of entries for all entries that existed in the *pdcChangeLog* on the old PDC during the PDC role transfer.
  2. All updates to the state of objects in the **domain** NC replica of the old PDC are reflected in the state of objects in the **domain** NC replica of the new PDC when the transfer is complete.
  3. All updates to the state of objects in **domain** NC replica on the new PDC that are not present on the old PDC have a corresponding entry in the *pdcChangeLog* on the new PDC, as described in section [3.1.1.7.2.2](#).
  4. The SamNT4ReplicationUSN and BuiltNT4ReplicationUSN variables were increased by adding 0x1000000000 during the transfer.

When predicate (2) above is satisfied after a transfer, the transfer does not cause NT4 BDCs to perform a full synchronization (described in [MS-NRPC] section [3.5](#)). The implementation satisfies predicate (2) above when possible.

Once the PDC role is acquired the following two entries are added to the *pdcChangeLog*. This notifies NT4 BDCs that the PDC has changed. SamNT4ReplicationUSN and BuiltNT4ReplicationUSN are updated prior to use in creating these entries.

<i>pdcChangeLog</i> entry	Field	Value
Entry 1	RelativeId	0x0
	Flags	CHANGELOG_SID
	DbDelta	AddOrChangeDomain
	DbIndex	0x0
	SerialNumber	SamNT4ReplicationUSN
Entry 2	RelativeId	0x0
	Flags	CHANGELOG_SID
	DbDelta	AddOrChangeDomain
	DbIndex	0x1
	SerialNumber	BuiltNT4ReplicationUSN



#### 3.1.1.7.2.4 Resetting the pdcChangeLog

To reset the *pdcChangeLog*, set the array to have 0 elements, set SamCreationTime and BuiltinCreationTime to the current time and SamNT4ReplicationUSN and BuiltinNT4ReplicationUSN to one.

Resetting the *pdcChangeLog* has the effect of causing NT4 BDCs to perform a full sync.

#### 3.1.1.7.3 Format of referent of pmsgOut.V1.pLog

The DRS\_MSG\_NT4\_CHGLOG\_REPLY\_V1 ([MS-DRSR] section [4.1.11.1.4](#)) response message to a IDL\_DRSGetNT4ChangeLog request ([MS-DRSR] section [4.1.11](#)) contains a BYTE \*pLog field. The format of the referent of this field is not specified in [MS-DRSR] section 4.1.11; it is specified here.

The referent of this field is a CHANGE\_LOG\_ENTRIES structure:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Size																															
Version																															
SequenceNumber																															
Flags																															
ChangeLogEntries (variable)																															
...																															

**Size:** The size in bytes of the part of the buffer preceding the **ChangeLogEntries** field. Equals 0x00000010.

**Version:** The version of the message. Equals 0x00000001.

**SequenceNumber:** The sequence number for the buffer. Is set to 0x00000001 in a response to a IDL\_DRSGetNT4ChangeLog request with pmsgIn.V1.pRestart = NULL. The value of pmsgOut.V1.pRestart in any IDL\_DRSGetNT4ChangeLog response encapsulates SequenceNumber. In a response to a IDL\_DRSGetNT4ChangeLog request with pmsgIn.V1.pRestart ≠ NULL, SequenceNumber is the value encapsulated in pmsgIn.V1.pRestart, plus one.

**Flags:** Equals 0x00000000. Ignored upon receipt.

**ChangeLogEntries:** A sequence of CHANGELOG\_ENTRY. Each CHANGELOG\_ENTRY is followed by padding bytes with value zero such that the number of bytes in CHANGELOG\_ENTRY plus the padding is congruent to zero mod 8.

The server stores the total number of bytes in the fixed-length and variable-length portions of the CHANGE\_LOG\_ENTRIES structure in the DWORD cbLog field of the

DRS\_MSG\_NT4\_CHGLOG\_REPLY\_V1 response message. This field allows the client to determine the number of CHANGELOG\_ENTRY structures contained in the CHANGE\_LOG\_ENTRIES structure.

### 3.1.1.8 AD/LDS Special Objects

AD/LDS NCs can contain the special types of objects: AD/LDS users and AD/LDS bind proxies. Special processing applies to these types of objects.

#### 3.1.1.8.1 AD/LDS Users

An AD/LDS user object is a security principal object in AD/LDS that contains a password.

If at least one of the following statements applies to an object class within an AD/LDS schema, then each instance of that object class functions as an AD/LDS user:

1. The object class contains [msDS-BindableObject](#) as a static auxiliary class.
2. The object class contains a static auxiliary class that is a subclass of [msDS-BindableObject](#).
3. The object class is a subclass of another object class that satisfies statement a) or b).

An AD/LDS user object has these special properties and behavior:

- Its [objectSid](#) is assigned during add as specified in section [3.1.1.5.2.4](#).
- It can be a member of [group](#) objects in its AD/LDS forest, subject to the limitations on inter-NC references specified in section [3.1.1.2.2.3](#), Referential Integrity.
- It can be named in an **LDAP** bind; section [5.1.1.5](#) specifies the supported authentication mechanisms and protocols. If the bind succeeds, it creates a security context for the **LDAP** connection as specified in section [5.1.3.4](#).
- Its password can both be assigned an initial value and updated. Special processing is performed on both the initial assignment and on update. Sections [3.1.1.5.2.2](#), [3.1.1.5.2.4](#), [3.1.1.5.3.1](#), [3.1.1.5.3.2](#), and [3.1.1.5.3.3](#) specify this processing.
- Its [objectSid](#) can be written into an AD/LDS security descriptor, subject to restrictions specified in section [7.1.3.3](#).

#### 3.1.1.8.2 Bind Proxies

An AD/LDS bind proxy object is a security principal object that represents a Windows user, not an AD/LDS user. A bind proxy object does not contain a password.

If at least one of the following statements applies to an object class within an AD/LDS schema, then each instance of that object class functions as an AD/LDS bind proxy:

1. The object class contains [msDS-BindProxy](#) as a static auxiliary class.
2. The object class contains a static auxiliary class that is a subclass of [msDS-BindProxy](#).
3. The object class is a subclass of another object class that satisfies statement a) or b).

An AD/LDS bind proxy object has these special properties and behavior:

- Its [objectSid](#) is assigned during Add and is the SID of some Windows user in a security realm trusted by the machine running the AD/LDS DC that performed the add. For instance, if an

AD/LDS DC is running on a machine that is joined to an Active Directory **domain** D, then the [objectSid](#) of a bind proxy created by that DC can be a user within D or within the forest that contains D, or within any **domain** or forest trusted by D or the forest that contains D.

- It can be a member of [group](#) objects in its AD/LDS forest, subject to the limitations on inter-NC references specified in section [3.1.1.2.2.3](#), Referential Integrity.
- It can be named in an **LDAP** bind; section [5.1.1.5](#) specifies the supported authentication mechanisms and protocols. If the bind succeeds, it creates a security context for the **LDAP** connection as specified in section [5.1.3.4](#).
- It does not contain a password. Special processing is performed on update to its password attribute, as specified in section [3.1.1.5.3.3](#).

## 4 Protocol Examples

**Note** To examine a sample scenario for joining a domain, see [\[MS-SYS\] Appendix A](#).

The Active Directory Technical Specification (this document) does not specify a protocol, but rather a state model and a set of behaviors that must be followed such that protocols in the documentation set (such as the protocols specified in [\[MS-DRSR\]](#) and [\[MS-SAMR\]](#)) will expose the expected behavior to Windows clients. While it includes a discussion of **LDAP**, it does so only to specify Active Directory's conformance and extensions to that protocol, and not to specify the protocol itself.

As such, no protocol examples are appropriate for this document. This section is left in place to maintain section numbering consistency with the documentation template used throughout the protocol documentation set.

## 5 Security

The following sections specify security considerations for implementers of Active Directory.

### 5.1 LDAP Security

#### References

LDAP attributes [userPassword](#), [sAMAccountName](#), [userPrincipalName](#), [uPNSuffixes](#), [supportedCapabilities](#), [servicePrincipalName](#), [nTSecurityDescriptor](#), [schemaIDGUID](#), [attributeSecurityGUID](#), [dSHeuristics](#), [validAccesses](#), [rightsGuid](#), [appliesTo](#): [\[MS-ADA1\]](#), [\[MS-ADA2\]](#), [\[MS-ADA3\]](#)

LDAP object class [controlAccessRight](#): [\[MS-ADSC\]](#)

ACCESS\_MASK structure and access right bits: [\[MS-DTYP\]](#)

ACE structure: [MS-DTYP]

ACL structure: [MS-DTYP]

SECURITY\_DESCRIPTOR structure: [MS-DTYP]

#### 5.1.1 Authentication

This section discusses the use of the **LDAP** bind mechanism in Active Directory to perform **authentication**, and the various **authentication** methods supported.

##### 5.1.1.1 Supported Authentication Methods

[\[RFC2251\]](#) section 4.2 defines an AuthenticationChoice structure for a BindRequest that contains two alternatives, simple and SASL. The former is for **LDAP** simple binds, while the latter is for **LDAP** SASL binds (as documented in [\[RFC2829\]](#)). Active Directory supports both of these mechanisms. In addition, it supports a third mechanism named "Sicily" that is primarily intended for compatibility with legacy systems. Sicily support adds three choices to the AuthenticationChoice structure, resulting in the following:

```
AuthenticationChoice ::= CHOICE {  
    simple                [0]    OCTET STRING,  
    sasl                  [3]    SaslCredentials  
    sicilyPackageDiscovery [9]    OCTET STRING  
    sicilyNegotiate       [10]   OCTET STRING  
    sicilyResponse        [11]   OCTET STRING }
```

The relationship of the three **authentication** mechanisms, and the **authentication** protocols supported by each, is summarized in the following tables.

Authentication Mechanism: Simple

Authentication protocols	Comments
-	-

Authentication Mechanism: SASL

Authentication protocols	Comments
GSS-SPNEGO	GSS-SPNEGO in turn uses Kerberos or NTLM as the underlying <b>authentication</b> protocol.
GSSAPI	GSSAPI in turn always uses Kerberos as the underlying <b>authentication</b> protocol.
EXTERNAL	-
DIGEST-MD5	-

Authentication Mechanism: Sicily

Authentication protocols	Comments
NTLM	-

Each of the three **authentication** mechanisms supported by Active Directory is discussed in more detail in the following sections.

#### 5.1.1.1.1 Simple Authentication

The support of simple bind in Active Directory is consistent with [\[RFC2251\]](#) section 4.2 and [\[RFC2829\]](#). Active Directory does not require, but supports, the use of a SSL/TLS-encrypted or otherwise protected connection when performing a simple bind. Also, while section 6.2 of [\[RFC2829\]](#) specifies that an object possessing a [userPassword](#) attribute is a prerequisite to being able to perform a simple bind using that object's credentials, Active Directory does not use the [userPassword](#) attribute to store the user's password in most cases, and possession of such an attribute is not a prerequisite to performing a simple bind against an object. The password attributes used in Active Directory are discussed in more detail in **LDAP** Password Modify Operations in section [3.1.1.3](#).

When performing a simple bind, Active Directory accepts several forms of name in the name field of the BindRequest. Each form of name is tried in turn. If the name field of the BindRequest maps to a single object using the attempted name form, the password on that object is checked and the **authentication** succeeds or fails (with error *invalidCredentials*) depending on the result. If the name field of the BindRequest maps to more than one object, the BindRequest fails with the error *invalidCredentials*. If the name field of the BindRequest maps to no object, the next form of object name is tried; if all forms have been tried, the BindRequest fails with the error *invalidCredentials*.

For AD/DS, the forms of name are tried in the order they are listed below. For AD/LDS, the forms of name are tried in the order below, except that forms marked "Only for AD/DS" are not tried, and the User Principal Name mapping (the second form below) is tried last.

The name forms are:

1. The DN of the object.
2. The User Principal Name (UPN) of the object. The UPN of an object is either:
  - A value of the [userPrincipalName](#) attribute of the object, or
  - Only for AD/DS: the value of the [sAMAccountName](#) attribute of the object, followed by a '@' sign, followed by either:

- The DNS name of a **domain** in the same forest as the object, or
- A value in the [uPNSuffixes](#) attribute of the Partitions object in the config NC replica.

When a name matches both the [userPrincipalName](#) attribute of one object and the UPN generated from the [sAMAccountName](#) of another object, the simple bind processing attempts to authenticate as the first object, i.e., priority is given to the value of the [userPrincipalName](#) attribute, rather than failing the bind due to duplicate objects.

3. Only for AD/DS: The NetBIOS name of the **domain**, followed by a backslash ('\'), followed by the value of the [sAMAccountName](#) attribute of the object.
4. The **canonical name** of the object.
5. The value of the [objectGUID](#) attribute of the object, expressed in dashed-string form ([\[RFC4122\]](#) section 3) and surrounded by curly braces (for example, "{ca2e693f-6280-4589-9376-b3707345d3ad}").
6. The value of the [displayName](#) attribute of the object.
7. Only for AD/DS: A value of the [servicePrincipalName](#) attribute of the object.
8. Only for AD/DS: A value V that when the MapSPN(V, M) algorithm of [MS-DRSR] section [4.1.4.2.19](#) is applied to it corresponds to a value of the [servicePrincipalName](#) attribute of the object. M is the value of the [sPNMappings](#) attribute of the [nTDSservice](#) object.
9. The value of the [objectSid](#) attribute of the object, in SDDL SID string form ([\[MS-DTYP\]](#) sections [2.4.2](#) and [2.5.1](#)).
10. Only for AD/DS: A value from the [sidHistory](#) attribute of the object, in SDDL SID string form ([\[MS-DTYP\]](#) sections [2.4.2](#) and [2.5.1](#)).
11. The **canonical name** of the object in which the rightmost forward slash (/) is replaced with a newline character (\n).

#### 5.1.1.1.2 SASL Authentication

The support of SASL bind in Active Directory is consistent with [\[RFC2251\]](#) section 4.2.1 and [\[RFC2829\]](#). The following SASL mechanisms are supported by **Active Directory**. They are briefly described in LDAP SASL Mechanisms, section [3.1.1.3](#):

- GSS\_SPNEGO [\[MS-SPNG\]](#)
- GSSAPI [\[RFC2078\]](#)
- EXTERNAL [\[RFC2829\]](#)
- DIGEST-MD5 [\[RFC2831\]](#)

Active Directory supports the optional use of integrity verification or encryption that is negotiated as part of the SASL authentication. While Active Directory permits SASL binds to be performed on a SSL/TLS-protected connection, it does not permit the use of SASL-layer encryption/integrity verification mechanisms on such a connection. Additionally, once a SASL-layer encryption/integrity verification mechanism is in use on a connection, the client must not send an additional bind request on that connection (for example, to change the credentials with which the connection is authenticated), unless the LDAP\_CAP\_ACTIVE\_DIRECTORY\_LDAP\_INTEG\_OID capability is present in the [supportedCapabilities](#) attribute of the rootDSE for that DC (see LDAP Capabilities in section

[3.1.1.3](#)). If the client sends an additional bind to a DC on which that capability is not present, the DC returns the *unwillingToPerform* error.

Regarding [\[RFC2829\]](#) section 9: when using the EXTERNAL SASL mechanism, Active Directory supports the *authzId* field. However, it only supports the *dnAuthzId* form, and not the *uAuthzId* form. Additionally, it does not permit an authorization identity to be established on the connection that is different from the **Authentication** identity used on the connection. Violation of either of these rules causes the DC to return the *invalidCredentials* error.

Regarding [\[RFC2829\]](#) section 6.1: when using the DIGEST-MD5 mechanism, Active Directory does not support subsequent authentication. However, the credentials field contains the string defined by "response-auth" in [\[RFC2831\]](#) section 2.1.3.

#### 5.1.1.1.3 Sicily Authentication

Sicily is a combination of a package discovery mechanism and an **authentication** mechanism. Unlike SASL, Sicily includes package discovery in the **authentication** mechanism itself. The package discovery mechanism permits a client to discover the **authentication** protocols (also known as packages) a DC supports. The **authentication** mechanism then permits a client to authenticate using one of those protocols. The **authentication** mechanism is independent of the package discovery mechanism in that a client may skip the package discovery mechanism entirely and proceed directly to the **authentication** mechanism (for example, if the client has some out-of-band knowledge of which **authentication** protocols the server supports).

All versions of Active Directory expose and support only the NTLM authentication protocol, as specified in [\[MS-NLMP\]](#), via Sicily.

The package discovery mechanism is performed by the client sending a *BindRequest* to the DC in which the *name* field of the *BindRequest* is empty and the *authentication* field contains the *sicilyPackageDiscovery* choice. The octet string contained in the *sicilyPackageDiscovery* choice is not used and is empty.

The DC responds to a *sicilyPackageDiscovery* by returning a *SicilyBindResponse*. A *SicilyBindResponse* is similar to a [\[RFC2251\]](#) *BindResponse*, but some of the fields differ. The *SicilyBindResponse* is defined as follows:

```
SicilyBindResponse ::= [APPLICATION 1] SEQUENCE {
    resultCode    ENUMERATED {
        success                (0),
        protocolError          (2),
        adminLimitExceeded     (11),
        inappropriateAuthentication (48),
        invalidCredentials      (49),
        busy                   (51),
        unavailable            (52),
        unwillingToPerform     (53),
        other                  (80) },
    serverCreds    OCTET STRING,
    errorMessage   LDAPString }
```

Note that *resultCode* is a subset of the enumeration present in *LDAPResult*. If the *sicilyPackageDiscovery* request is successful, the DC sets the *resultCode* to *success* in its *SicilyBindResponse* and returns in *serverCreds* an ANSI string consisting of the semicolon-separated names of the **authentication** protocols it supports via the Sicily authentication mechanism. Active Directory supports NTLM, and returns the string "NTLM" in the package discovery response. The



names of the **authentication** protocols are ordered in the server's preferred order, starting with the most-preferred **authentication** protocol. If the `sicilyPackageDiscovery` request is not successful, the DC returns an error in the `resultCode` field of the `SicilyBindResponse`. If the `sicilyPackageDiscovery` request fails because the DC does not support any **authentication** protocols via Sicily, the DC returns the error *inappropriateAuthentication*. The `errorMessage` field of the `SicilyBindResponse` may contain additional implementation-specific details indicating why the request failed.

Once the client has determined which **authentication** protocol it will use, it uses the Sicily authentication mechanism to authenticate the connection. The Sicily authentication mechanism consists of two requests, both of which take the form of a **LDAP** `BindRequest`. The first request is the `sicilyNegotiate` request. If successful, this is followed by the `sicilyResponse` request.

The **authentication** begins when the client sends the `sicilyNegotiate` request to the DC. This constitutes a `BindRequest` in which the `name` field is set to "NTLM" and the `authentication` field contains the `sicilyNegotiate` choice. The `sicilyNegotiate` choice contains an octet string consisting of binary data supplied by and dependent on the **authentication** protocol used, and which serves as a representation of the credentials with which the client wishes to authenticate the connection. If successful, the DC responds with a `SicilyBindResponse` in which the `resultCode` is set to *success* and the `serverCreds` contains binary data supplied by the **authentication** protocol on the server side. The client is expected to pass this binary data, whose content is **authentication** protocol-specific, to its implementation of the **authentication** package. If not successful, the DC returns an error in the `resultCode` field of the `SicilyBindResponse`, indicating that the `sicilyNegotiate` request was not successful. If the credentials supplied by the client are invalid, the DC returns the *invalidCredentials* error. If the client requests an **authentication** protocol that is not supported by the DC, it returns the *inappropriateAuthentication* error. The `errorMessage` field of the `SicilyBindResponse` may contain additional implementation-specific details indicating why the request failed.

If the `sicilyNegotiate` request is successful, the client then sends the `sicilyResponse` request to the DC by sending a `BindRequest` in which the `name` field is empty and the `authentication` field contains the `sicilyResponse` choice. The octet string in the `sicilyResponse` choice contains **authentication** protocol-specific data, generated in response to the data received in the `serverCreds` field of the `SicilyBindResponse`. The DC responds to this `sicilyResponse` request by sending a `SicilyBindResponse`. The `serverCred` field is not used in this response and is empty. If successful, the DC sets the `resultCode` field to *success* and the connection is now authenticated as the client supplied credentials. If the bind fails, the DC sets `resultCode` to an error and the connection is not authenticated. As in the previous case, the DC uses the error code *invalidCredentials* to indicate that the client presented incorrect credentials, and the error code *inappropriateAuthentication* to indicate that the client requested an unsupported protocol. The `errorMessage` field of the `SicilyBindResponse` may contain additional implementation-specific details indicating why the request failed.

As with SASL, integrity verification or encryption can be negotiated as part of the Sicily authentication. The support for, and means of implementation of, such mechanisms is dependent on the particular **authentication** protocol used (for example, NTLM). As with SASL, such mechanisms cannot be used on a connection protected by SSL/TLS mechanisms, and once such a mechanism is in use, the connection cannot be rebound unless the `LDAP_CAP_ACTIVE_DIRECTORY_LDAP_INTEG_OID` capability is present in the `supportedCapabilities` attribute of the rootDSE of the DC.

#### 5.1.1.2 Using SSL/TLS

Active Directory permits two means of establishing a SSL/TLS-protected connection to a DC. The first is by connecting to a DC on a protected LDAPS port (TCP ports 636 and 3269 in AD/DS, and an configuration-specific port in AD/LDS). The second is by connecting to a DC on a regular **LDAP** port (TCP ports 389 or 3268 in AD/DS, and a configuration-specific port in AD/LDS), and later sending an `LDAP_SERVER_START_TLS_OID` extended operation [\[RFC2830\]](#). In both cases, the DC will request

(but not require) the client's certificate as part of the SSL/TLS handshake [\[RFC2246\]](#). If the client presents a valid certificate to the DC at that time, it can be used by the DC to authenticate (bind) the connection as the credentials represented by the certificate.

If the client establishes the SSL/TLS-protected connection by means of connecting on a protected LDAPS port, then the connection is considered to be immediately authenticated (bound) as the credentials represented by the client certificate. An EXTERNAL bind is not required, but is permitted. If the client does not present a certificate during the SSL/TLS handshake, the connection is not authenticated and is treated as anonymous. In that case, the DC rejects any attempt to perform an EXTERNAL bind with the error *invalidCredentials*.

If the client establishes the SSL/TLS-protected connection by means of an LDAP\_SERVER\_START\_TLS\_OID operation, the **authentication** state of the connection remains the same after the operation as it was before the operation. The DC authenticates the connection as the credentials represented by the client's certificate only if an EXTERNAL SASL bind is subsequently performed. This is similar to the "implicit assertion" of section 5.1.2.1 of [\[RFC2830\]](#), except that neither the **authentication** identity nor the authorization identity is established on the connection until the EXTERNAL bind takes place. If the client includes the authzId field in the EXTERNAL bind, in accord with the "explicit assertion" of [\[RFC2830\]](#) section 5.1.2.2, then as described in section [5.1.1.1.2](#) the authzId field contains the DN of the object that the EXTERNAL bind is authenticating the connection as, in other words, the object associated with the credentials represented by the certificate. Therefore, the implicit assertion and explicit assertion are functionally identical. If the client performs an EXTERNAL bind but does not supply a certificate during the SSL/TLS handshake, the EXTERNAL bind fails with the error *invalidCredentials*.

Alternatively, the client can perform any other form of **LDAP** bind that is permissible on a SSL/TLS-protected connection, or the client can perform no bind to continue to use any **authentication** and authorization identity that was previously established on the connection.

### 5.1.1.3 Using Fast Bind

Active Directory supports a mode of operation known as "fast bind" that can be enabled per **LDAP** connection. The intent of this mode is to provide a means for clients to use the **LDAP** bind request to simply validate credentials without the overhead of establishing the authorization information. Fast bind mode is enabled on a connection by sending the LDAP\_SERVER\_FAST\_BIND\_OID LDAP extended operation on the connection, documented in LDAP Extended Operations in section [3.1.1.3](#).

Once fast bind mode is enabled on a connection, it cannot be disabled on that connection. This mode cannot be enabled on a connection on which a successful bind was previously performed, and the server returns *unwillingToPerform* if such an attempt is made.

When fast bind mode is enabled on a **LDAP** connection, the DC accepts bind requests and validates the credentials presented, returning an error code indicating a success or failure. However, on successful binds the DC does not perform authorization steps and the connection is treated as if it was authenticated and authorized as the anonymous user.

While section 4.2.1 of [\[RFC2251\]](#) specifies that a bind request causes all operations currently in progress on a connection to be abandoned, when the connection is in fast bind mode, multiple independent binds (for example, using different credentials) can simultaneously be in progress on the same connection without any of them being abandoned. This permits a client to validate multiple sets of credentials at the same time, while the DC always considers the connection to be authenticated and authorized as the anonymous user.

Only simple binds are accepted on a connection in fast bind mode. The client can use SSL/TLS protection on a connection in fast bind mode.

#### 5.1.1.4 Mutual Authentication

[MS-DRSR] sections 2.2.2 and 2.2.4 specify the mutual **Authentication** requirements for client-to-DC interactions over the **RPC** interfaces documented in [MS-DRSR]. The requirements are the same for mutual **Authentication** in an **LDAP** connection.

Therefore by registering its SPNs for the **RPC** interfaces documented in [MS-DRSR], a DC also satisfies its SPN registration requirements for LDAP.

#### 5.1.1.5 Supported Types of Security Principals

For AD/DS, the concept of **security principal** is straightforward: a security principal is an object in the directory that possesses an [objectSid](#) attribute. But for AD/LDS, the notion of security principal is more complex because AD/LDS recognizes three distinct types of security principal, any of which can authenticate via an **LDAP** bind request.

- AD/LDS **security principals** that are created in an AD/LDS NC.
- Principals that are defined by the operating system of the computer on which AD/LDS is running.
- Principals that are defined in an Active Directory **domain** to which the computer on which AD/LDS is running is joined, or principals that are in **domains** trusted by the joined **domain**.

In addition to these three types of **security principals**, AD/LDS also supports bind proxies, which are not **security principals** but which can be authenticated via a **LDAP** bind request. This section will discuss each of the three types of security principals in turn, and follow that with a discussion of bind proxies. Finally, it will conclude with an explanation of which types of **LDAP** binds a AD/LDS server must support for each type of principal and bind proxy.

The first type of security principal in AD/LDS is unchanged from AD/DS: an object in the directory that possesses an [objectSid](#) attribute. However, while AD/DS restricts **security principals** to the **domain** NC, AD/LDS (which has no **domain** NCs) permits **security principals** to be stored in an application NC. Additionally, if the ADAMAllowADAMSecurityPrincipalsInConfigPartition configuration setting equals 1 (section [3.1.1.3.4.7](#)), AD/LDS permits **security principals** to be created in the config NC.

In AD/DS, the set of security principal object classes is fixed. In AD/LDS, any object class that statically links (section [3.1.1.2.4.6](#)) to the [msDS-BindableObject](#) auxiliary class is a security principal object class. Dynamically instantiating the [msDS-BindableObject](#) auxiliary class does not have the same effect.

The second and third types of principals are similar to each other in that both are means for AD/LDS to *pass-through* the **Authentication** to the underlying operating system on which it is running. AD/LDS recognizes as a security principal those **security principals** (users and groups) that are stored locally on the computer on which AD/LDS is running. Additionally, if the computer is a member of a **domain**, then AD/LDS recognizes as **security principals** any **security principals** which are in that **domain**, or which are in a **domain** trusted by that **domain**. Such **security principals** may be included in the security descriptors of objects in the AD/LDS directory in the same fashion as **security principals** of the first type. Additionally, such **security principals** may be included in the membership of [group](#) objects in AD/LDS, and in the [msDS-ServiceAccount](#) attribute of [nTDSDSA](#) objects in AD/LDS, via the automatic creation of [foreignSecurityPrincipal](#) objects (sections [3.1.1.5.2.4](#) and [3.1.1.5.3.3](#)).

Note that, except for the creation of [foreignSecurityPrincipal](#) objects as needed to represent group members or service accounts, the second and third types of principals are not represented as objects in AD/LDS. Instead, upon receipt of a **LDAP** Bind request for such a principal, AD/LDS

provides the credentials it receives in the Bind request to the host operating system and relies on the host operating system to validate those credentials. The means of passing the received credentials to the host operating system, as well as the method that the host operating system uses to validate those credentials, is implementation-specific.

Bind proxies are objects in AD/LDS that contain the [msDS-BindProxy](#) auxiliary class. A bind proxy contains an [objectSid](#) attribute but is not a security principal. Rather, it is a means of associating an object in AD/LDS with a security principal of the underlying operating system (that is, the second or third type of security principal). The [objectSid](#) attribute contains the SID of a security principal of the second or third type. When a **LDAP** Bind request is received in which the object identified in the name field of the BindRequest is a [msDS-BindProxy](#) object, the server performs the following actions:

- Retrieve the value V of the [objectSid](#) attribute from the named object.
- Pass-through the **Authentication** request to the host operating system as a request to authenticate a principal whose SID is V and whose password is as supplied in the **LDAP** bind request.

An **LDAP** bind request that targets an [msDS-BindProxy](#) object O has nearly the same effect as an **LDAP** bind request for a security principal S of the second or third type. Instead of directly naming S in the **LDAP** bind request, the client names an object O such that O![objectSid](#) equals the SID of S. The security context generated by the two requests is slightly different as specified in section [5.1.3.4](#).

In order for an object class to be usable in an **LDAP** bind request in AD/LDS, that object class must either contain the [msDS-BindableObject](#) class or the [msDS-BindProxy](#) class.

AD/LDS servers restrict the **authentication** mechanisms and protocols that can be used to authenticate different types of security principal and bind proxies. The **authentication** mechanisms and protocols supported by AD/LDS for each type of principal or proxy is specified in the following table.

Type of principal/proxy	Supported authentication mechanism	Supported authentication protocol
First type (AD/LDS principal)	Simple SASL	- DIGEST-MD5
Second or third type (computer or domain principal)	SASL SASL SASL SASL Sicily	GSSAPI GSS-SPNEGO DIGEST-MD5 EXTERNAL NTLM
Bind proxy	Simple	-

In particular, note that simple bind is not supported for principals of the second or third type, and that DIGEST-MD5 is the only SASL protocol supported for all types of **security principals** in AD/LDS.

## 5.1.2 Message Security

### 5.1.2.1 Using SASL

Active Directory supports the optional use of **LDAP** message security layer providing message integrity and/or confidentiality protection services that is negotiated as part of the SASL authentication. Support for such mechanisms and their implementation is dependent on the specific **authentication** protocol used (for example, Kerberos or **Digest**), and is documented in the SASL specification for each **authentication** protocol.

Once a SASL negotiated security layer is in effect in the **LDAP** protocol data stream, it remains in effect until either a subsequently negotiated security layer is installed or the underlying transport connection is closed. When in effect, the security layer processes protocol data into buffers of protected data as per [\[RFC2222\]](#).

While Active Directory permits SASL binds to be performed on a SSL/TLS-protected connection, it does not permit the use of SASL-layer confidentiality/integrity protection mechanisms on such a connection.

### 5.1.2.2 Using SSL/TLS

Active Directory supports **LDAP** message security on a SSL/TLS-protected connection to a DC in accordance with [\[RFC2246\]](#).

As indicated in the previous section, Active Directory does not permit the use of SASL-layer message confidentiality/integrity protection mechanisms to be employed on a SSL/TLS-protected **LDAP** connection.

## 5.1.3 Authorization

Although the **LDAP** security model does not include mechanisms for access control, Active Directory provides access control in the form of **access control lists** (ACLs) on directory objects.

If the `fLDAPBlockAnonOps` `dsHeuristic` (section [7.1.1.2.4.1.2](#)) is true, anonymous (unauthenticated) users are limited to performing rootDSE searches and binds. If `fLDAPBlockAnonOps` is false, anonymous users can perform any **LDAP** operation, subject to access checks that use the ACL mechanisms described in this section.

### 5.1.3.1 Background

The *security context* of a requestor (see **security context** in Glossary) requesting access to an Active Directory object represents the authorization information associated with the requestor. A DC performs an **access check** to determine whether the security context, and thus the requestor, is authorized for the type of access that has been requested before allowing any further processing to continue. Access control information associated with an object is contained in the security descriptor of the object.

Every object in Active Directory has an [nTSecurityDescriptor](#) attribute whose value is the security descriptor containing access control information for the object.

An **access check** compares information in the thread's security context with information in the object's security descriptor:

- The security context contains a SID that identifies the principal associated with the thread and SIDs that identify the groups of which the principal is a member.

- The security descriptor contains a **discretionary access control list (DACL)** that specifies the access rights allowed or denied to specific principals or groups. It also identifies the *owner* of the object. The structure of a security descriptor is described in [MS-DTYP] section **2.4.6**.

A DACL in a security descriptor is an ordered list of **access control entries (ACEs)** that define the protections that apply to an object and its properties. Each ACE identifies a security principal (that is, a [user](#), [group](#), etc.) and specifies a set of access rights allowed, denied, or audited for that security principal. The data structures for an ACE and a DACL are described in [MS-DTYP] sections [2.4.4](#) and [2.4.5](#).

There are two types of ACEs: simple and object-specific. A simple ACE applies to an entire object. If a simple ACE gives a particular user read access, the user can read all information associated with the object. An object-specific ACE, on the other hand, can apply to any individual attribute of an object or to a set of attributes. It makes it possible to place independent access controls on each attribute of an Active Directory object.

During an access check, the server steps through the ACEs in the order in which they appear in the object's DACL, looking for ACEs that apply to the principal and group SIDs from the thread's security context. It steps through each ACE until it finds one that either allows or denies access to the principal or one of the principal's groups, or until there are no more ACEs to check. If it comes to the end of the DACL and the thread's desired access is still not explicitly allowed or denied, the server denies access to the object.

The order in which ACEs are listed in a DACL is important. For example, an object's DACL might contain one ACE that allows access to a group and another ACE that denies access to a principal who is a member of the group. If the **access check** process encounters the ACE that allows access to the group before the ACE that denies access to the principal, the principal is allowed access the object. If the ACEs are encountered in the reverse order, then the principal is denied access to the object.

AD/LDS **security principals** cannot appear in an AD/DS ACE. Section [7.1.3.3](#) specifies a restriction on the AD/LDS **security principals** that can be used in an AD/LDS ACE.

### 5.1.3.2 Access Rights

The following diagram specifies access rights that can be assigned to or requested for an Active Directory object. The **access mask** in an **access control entry (ACE)** contains a combination of these values.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
G	G	G	G	X	X	X	X	X	X	X	X	W	W	R	D	X	X	X	X	X	X	X	C	L	D	W	R	V	L	D	C
R	W	X	A									O	D	C	E								R	O	T	P	P	W	C	C	C

**Note** The values are presented in big-endian byte order.

**GR (RIGHT\_GENERIC\_READ):** The right to read permissions on this object, read all the properties on this object, list this object name when the parent **container** is listed, and list the contents of this object if it is a **container**.

**GW (RIGHT\_GENERIC\_WRITE):** The right to read permissions on this object, write all the properties on this object, and perform all validated writes to this object.



**GX (RIGHT\_GENERIC\_EXECUTE):** The right to read permissions on, and list the contents of, a **container** object.

**GA (RIGHT\_GENERIC\_ALL):** The right to create or delete child objects, delete a subtree, read and write properties, examine child objects and the object itself, add and remove the object from the directory, and read or write with an extended right.

**WO (RIGHT\_WRITE\_OWNER):** The right to modify the owner of an object in the object's security descriptor. A user can only take ownership of an object, but cannot transfer ownership of an object to other users.

**WD (RIGHT\_WRITE\_DAC):** The right to modify the **discretionary access-control list (DACL)** in the object security descriptor.

**RC (RIGHT\_READ\_CONTROL):** The right to read data from the security descriptor of the object, not including the data in the SACL.

**DE (RIGHT\_DELETE):** The right to delete the object.

**CR (RIGHT\_DS\_CONTROL\_ACCESS):** The right to perform an operation controlled by a **control access right**. The ObjectType member of an ACE can contain a **GUID** that identifies the **control access right**. If ObjectType does not contain a **GUID**, the ACE controls the right to perform all **control access right** controlled operations associated with the object. For a list of **control access rights**, see section [5.1.3.2.1](#).

**LO (RIGHT\_DS\_LIST\_OBJECT):** The right to list a particular object. If the user is not granted this right and the user is not granted the RIGHT\_DS\_LIST\_CONTENTS right on the object's parent, the object is hidden from the user. Note LIST\_OBJECT rights are not enforced by AD by default. In order to enable LIST\_OBJECT enforcement, a [dSHeuristics](#) bit fDoListObject must be set. For more information about this right, see section [7.1.1.2.4.1.2](#).

**DT (RIGHT\_DS\_DELETE\_TREE):** The right to perform Delete-Tree operation on this object. See the Delete operation in section [3.1.1.5](#) for more details.

**WP (RIGHT\_DS\_WRITE\_PROPERTY):** The right to write properties of the object. The ObjectType member of an ACE can contain a **GUID** that identifies a property set or an attribute. If ObjectType does not contain a **GUID**, the ACE controls the right to write all attributes of the object.

**RP (RIGHT\_DS\_READ\_PROPERTY):** The right to read properties of the object. The ObjectType member of an ACE can contain a **GUID** that identifies a property set or an attribute. If ObjectType does not contain a **GUID**, the ACE controls the right to read all attributes of the object.

**VW (RIGHT\_DS\_WRITE\_PROPERTY\_EXTENDED):** The right to perform an operation controlled by a validated write access right. The ObjectType member of an ACE can contain a **GUID** that identifies the validated write. If ObjectType does not contain a **GUID**, the ACE controls the rights to perform all validated write operations associated with the object. For a list of validated write rights, see Modify operation in section [3.1.1.5](#).

**LC (RIGHT\_DS\_LIST\_CONTENTS):** The right to list child objects of this object. For more information about this right, see section [3.1.1.4](#).

**DC (RIGHT\_DS\_DELETE\_CHILD):** The right to delete child objects of the object. The ObjectType member of an ACE can contain a **GUID** that identifies the [objectClass](#) of child object whose deletion is controlled. If ObjectType does not contain a **GUID**, the ACE controls the deletion of all child object classes.

**CC (RIGHT\_DS\_CREATE\_CHILD):** The right to create child objects of the object. The ObjectType member of an ACE can contain a **GUID** that identifies the [objectClass](#) of child object whose creation is controlled. If ObjectType does not contain a **GUID**, the ACE controls the creation of all child object classes allowed by the schema.

**X:** Ignored. These bits are ignored in Active Directory DACLS.

The four generic rights are presented, along with the specific access rights which they represent. The mapping for access to objects in Active Directory is as follows:

**GR** = (RC | LC | RP | LO)

**GW** = (RC | WP | VW)

**GX** = (RC | LC)

**GA** = (DE | RC | WD | WO | CC | DC | DT | RP | WP | LC | LO | CR | VW)

Note that the "GENERIC" **access mask** bits above are never stored in Active Directory security descriptor values. They can be present in an SD value sent by a user in an add or modify request. When the SD value is stored in the database, the GENERIC access bits are mapped according to the GENERIC\_MAPPING structure. See section [7.1.3](#) and [\[MS-DTYP\]](#) section **2.4.3** for more information.

#### 5.1.3.2.1 Control Access Rights

In Active Directory, you can control which users have the right to perform a particular operation on an object or its attributes using standard access rights. However, there are certain operations that have semantics that are not tied to specific properties or where it is desirable to control access in a way not supported by the standard access rights. For example, you can grant users a "Reanimate tombstones" right so that he is able to perform tombstone reanimation on any object in a naming context. Active Directory allows the standard access control mechanism to be extended for controlling access to custom actions or operations using a mechanism called **control access rights**.

A **control access right** is not identified by a specific bit in an **access mask** as the standard access rights are. Instead, each **control access right** is identified by a **globally unique identifier (GUID)**. An ACE that grants or denies a **control access right** specifies the RIGHT\_DS\_CONTROL\_ACCESS (CR) bit in the ACCESS\_MASK field and the **GUID** identifying the particular **control access right** in the ObjectType field of the ACE. If ObjectType field does not contain a **GUID**, the ACE is deemed to control the right to perform all operations associated with the objects that are controlled by **control access rights**. For convenience and easy identification by Active Directory administrative tools facilitating access control, each **control access right** is represented by an object of class [controlAccessRight](#) in the Extended-Rights **container**. Note that these objects are not integral to evaluating access to an operation and hence, their presence is not required for the proper functioning of the access control mechanism. There are a number of predefined **control access rights** in Active Directory, and that list can be extended by application developers by adding [controlAccessRight](#) objects to the Extended-Rights **container**.

The pertinent attributes on the [controlAccessRight](#) object that defines the use of the **control access right** for the administrative tools; their meaning is described below:

- [validAccesses](#): The type of access right bits in the ACCESS\_MASK field of an ACE with which the **control access right** can be associated. The only permitted access right for **control access rights** is RIGHT\_DS\_CONTROL\_ACCESS (CR).
- [rightsGuid](#): The **GUID** used to identify the **control access right** in an ACE. The **GUID** value is placed in the ObjectType field of the ACE.



- [appliesTo](#): This multi-value attribute has a list of object classes that the **control access right** applies to. Each object class in the list is represented by the [schemaIDGUID](#) attribute of the [classSchema](#) object that defines the object class in the Active Directory schema.

The table below summarizes the predefined **control access rights**, and the corresponding **GUID** value identifying each right, that can be specified in an ACE.

Control Access Right symbol	Identifying GUID used in ACE
Abandon-Replication	ee914b82-0a98-11d1-adbb-00c04fd8d5cd
Add-GUID	440820ad-65b4-11d1-a3da-0000f875ae0d
Allocate-Rids	1abd7cf8-0a99-11d1-adbb-00c04fd8d5cd
Allowed-To-Authenticate	68b1d179-0d15-4d4f-ab71-46152e79a7bc
Apply-Group-Policy	edacfd8f-ffb3-11d1-b41d-00a0c968f939
Certificate-Enrollment	0e10c968-78fb-11d2-90d4-00c04f79dc55
Change-Domain-Master	014bf69c-7b3b-11d1-85f6-08002be74fab
Change-Infrastructure-Master	cc17b1fb-33d9-11d2-97d4-00c04fd8d5cd
Change-PDC	bae50096-4752-11d1-9052-00c04fc2d4cf
Change-Rid-Master	d58d5f36-0a98-11d1-adbb-00c04fd8d5cd
Change-Schema-Master	e12b56b6-0a95-11d1-adbb-00c04fd8d5cd
Create-Inbound-Forest-Trust	e2a36dc9-ae17-47c3-b58b-be34c55ba633
Do-Garbage-Collection	fec364e0-0a98-11d1-adbb-00c04fd8d5cd
Domain-Administer-Server	ab721a52-1e2f-11d0-9819-00aa0040529b
DS-Check-Stale-Phantoms	69ae6200-7f46-11d2-b9ad-00c04f79f805
DS-Execute-Intentions-Script	2f16c4a5-b98e-432c-952a-cb388ba33f2e
DS-Install-Replica	9923a32a-3607-11d2-b9be-0000f87a36b2
DS-Query-Self-Quota	4ecc03fe-ffc0-4947-b630-eb672a8a9dbc
DS-Replication-Get-Changes	1131f6aa-9c07-11d1-f79f-00c04fc2dcd2
DS-Replication-Get-Changes-All	1131f6ad-9c07-11d1-f79f-00c04fc2dcd2
DS-Replication-Get-Changes-In-Filtered-Set	89e95b76-444d-4c62-991a-0facbeda640c
DS-Replication-Manage-Topology	1131f6ac-9c07-11d1-f79f-00c04fc2dcd2
DS-Replication-Monitor-Topology	f98340fb-7c5b-4cdb-a00b-2ebdfa115a96
DS-Replication-Synchronize	1131f6ab-9c07-11d1-f79f-00c04fc2dcd2
DS-Replication-Secrets-Synchronize	1131f6ae-9c07-11d1-f79f-00c04fc2dcd2
Enable-Per-User-Reversibly-Encrypted-Password	05c74c5e-4deb-43b4-bd9f-86664c2a7fd5

Control Access Right symbol	Identifying GUID used in ACE
Generate-RSoP-Logging	b7b1b3de-ab09-4242-9e30-9980e5d322f7
Generate-RSoP-Planning	b7b1b3dd-ab09-4242-9e30-9980e5d322f7
Migrate-SID-History	ba33815a-4f93-4c76-87f3-57574bff8109
msmq-Open-Connector	b4e60130-df3f-11d1-9c86-006008764d0e
msmq-Peek	06bd3201-df3e-11d1-9c86-006008764d0e
msmq-Peek-computer-Journal	4b6e08c3-df3c-11d1-9c86-006008764d0e
msmq-Peek-Dead-Letter	4b6e08c1-df3c-11d1-9c86-006008764d0e
msmq-Receive	06bd3200-df3e-11d1-9c86-006008764d0e
msmq-Receive-computer-Journal	4b6e08c2-df3c-11d1-9c86-006008764d0e
msmq-Receive-Dead-Letter	4b6e08c0-df3c-11d1-9c86-006008764d0e
msmq-Receive-journal	06bd3203-df3e-11d1-9c86-006008764d0e
msmq-Send	06bd3202-df3e-11d1-9c86-006008764d0e
Open-Address-Book	a1990816-4298-11d1-ade2-00c04fd8d5cd
Read-Only-Replication-Secret-Synchronization	1131f6ae-9c07-11d1-f79f-00c04fc2dcd2
Reanimate-Tombstones	45ec5156-db7e-47bb-b53f-dbeb2d03c40f
Recalculate-Hierarchy	0bc1554e-0a99-11d1-adbb-00c04fd8d5cd
Recalculate-Security-Inheritance	62dd28a8-7f46-11d2-b9ad-00c04f79f805
Receive-As	ab721a56-1e2f-11d0-9819-00aa0040529b
Refresh-Group-Cache	9432c620-033c-4db7-8b58-14ef6d0bf477
Reload-SSL-Certificate	1a60ea8d-58a6-4b20-bcdc-fb71eb8a9ff8
SAM-Enumerate-Entire-Domain	91d67418-0135-4acc-8d79-c08e857cfbec
Send-As	ab721a54-1e2f-11d0-9819-00aa0040529b
Send-To	ab721a55-1e2f-11d0-9819-00aa0040529b
Unexpire-Password	ccc2dc7d-a6ad-4a7a-8846-c04e3cc53501
Update-Password-Not-Required-Bit	280f369c-67c7-438e-ae98-1d46f3c6f541
Update-Schema-Cache	be2bb760-7f46-11d2-b9ad-00c04f79f805
User-Change-Password	ab721a53-1e2f-11d0-9819-00aa0040529b
User-Force-Change-Password	00299570-246d-11d0-a768-00aa006e0529

### 5.1.3.2.2 Validated Writes

In Active Directory, write access to an object's attributes is controlled by using the DS\_RIGHT\_WRITE\_PROPERTY (WP) access right. However, that would allow any value that is permissible by the attribute schema, to be written to the attribute with no value checking performed. There are cases where validation of the attribute values being written, beyond that required by the schema, is necessary before writing them to an object in order to maintain integrity constraints. Active Directory extends the standard access control mechanism to allow such additional validation semantics to be incorporated using a mechanism called *validated write rights*.

A validated write right is not identified by a specific bit in an **access mask** as the standard access rights are. Instead, each validated write right is identified by a **GUID**. This **GUID** is the value of the [schemaIDGUID](#) attribute from the [attributeSchema](#) object of the attribute where the validated write is defined. An ACE that grants or denies a validated write right specifies the RIGHT\_DS\_WRITE\_PROPERTY\_EXTENDED (VW) bit in the ACCESS\_MASK field and the **GUID** identifying the particular validated write right in the ObjectType field of the ACE. If ObjectType field does not contain a **GUID**, the ACE is deemed to control the right to perform all validated write operations associated with the object. As with **control access rights**, each validated write right is represented by an object of class [controlAccessRight](#) in the Extended-Rights **container** for convenience and easy identification by Active Directory administrative tools. Note that these objects are not integral to evaluating access to an update operation and, hence, their presence is not required for the proper functioning of the access control mechanism. The predefined list of validated write rights in Active Directory cannot be extended by application developers.

The table below summarizes the validated write rights, and the corresponding **GUID** value identifying each right, that can be specified in an ACE.

Validated Write Right symbol	Identifying GUID used in ACE
Self-Membership	bf9679c0-0de6-11d0-a285-00aa003049e2 (member attribute)
Validated-DNS-Host-Name	72e39547-7b18-11d1-adeb-00c04fd8d5cd ( <a href="#">dNSHostName</a> attribute)
Validated-SPN	f3a64788-5306-11d1-a9c5-0000f80367c1 ( <a href="#">servicePrincipalName</a> attribute)

### 5.1.3.3 Checking Access

Before performing a requested access on an object in Active Directory, the DC performs an **access check** to confirm that the security context of the requestor is authorized for the type of access requested. This determination is made by using the following information:

- The requestor's security context
- The requestor's desired **access mask**
- An appropriate security descriptor (The security descriptor used for the **access check** is typically the security descriptor of the object itself, but for some types of access the security descriptor of the object's parent and/or other objects in the directory may be used).

Note that a special principal called "Principal Self," and identified by the fixed SID value of S-1-5-10, may appear in the SID field of an ACE in the security descriptor of an object. This fixed SID value represents the object itself in an ACE on a security principal object. For example, when an ACE on a [user](#) object grants certain access rights to Principal Self, it essentially grants those access rights to the user represented by that object. During an **access check** for object O, if O![nTSecurityDescriptor](#)

contains any ACEs with the fixed SID for Principal Self the server replaces them with O![objectSid](#) before proceeding with the access check.

For the access checking behavior described in the following sections, it is assumed that any security descriptor used as input to that process has already undergone the SID substitution for Principal Self (as described above), if necessary.

#### 5.1.3.3.1 Null vs. Empty DACLs

The presence of a NULL DACL in the [nTSecurityDescriptor](#) attribute of an object grants full access to the object to any principal that requests it; normal access checks are not performed with respect to the object.

An empty DACL, on the other hand, is a properly allocated and initialized DACL containing no access-control entries (ACEs). An empty DACL in the [nTSecurityDescriptor](#) attribute of an object grants no access to the object. Note that even with empty DACL, some rights are implied. For example, the current OWNER of an object is implicitly granted READ\_CONTROL and WRITE\_DAC access. If the user possesses the SE\_TAKE\_OWNERSHIP\_PRIVILEGE, then WRITE\_OWNER access is implied.

#### 5.1.3.3.2 Checking Simple Access

When evaluating standard access rights specified in simple ACEs for an Active Directory object, the security descriptor of the object is used. Let G and D denote the access rights that are granted and denied, respectively, on the object. Set both to a value of 0 initially.

The following rules are used to determine the authorization for the requestor's security context:

1. If the security descriptor has no DACL or its "DACL Present" (DP) bit is not set, then grant the requestor all possible access rights on the object.
2. If the DACL does not have any ACE, then grant the requestor no access rights on the object.
3. If the SID in the Owner field of the object's security descriptor matches any SID in the requestor's security context, then add the bits "Read Control" (RC), "Write DACL" (WD) and "Write Owner" (WO) to G.
4. Evaluate the DACL by examining each ACE in sequence, starting with the first ACE. Perform the following sequence of actions for each ACE in the order as shown. Let the ACCESS\_MASK field of the ACE have a value M.
  1. If the "Inherit Only" (IO) flag is set in the ACE, skip the ACE.
  2. If the SID in the ACE does not match any SID in the requestor's security context, skip the ACE.
  3. If the ACE type is "Access Denied" and the access rights in M are not in G, then add the rights in M to D.
  4. If the ACE type is "Access Allowed" and the access rights in M are not in D, then add the rights in M to G.
5. When the end of the DACL is reached, the access rights in G is the maximum standard access available to the requestor on the object. Check the requested **access mask** against the access rights granted in G.

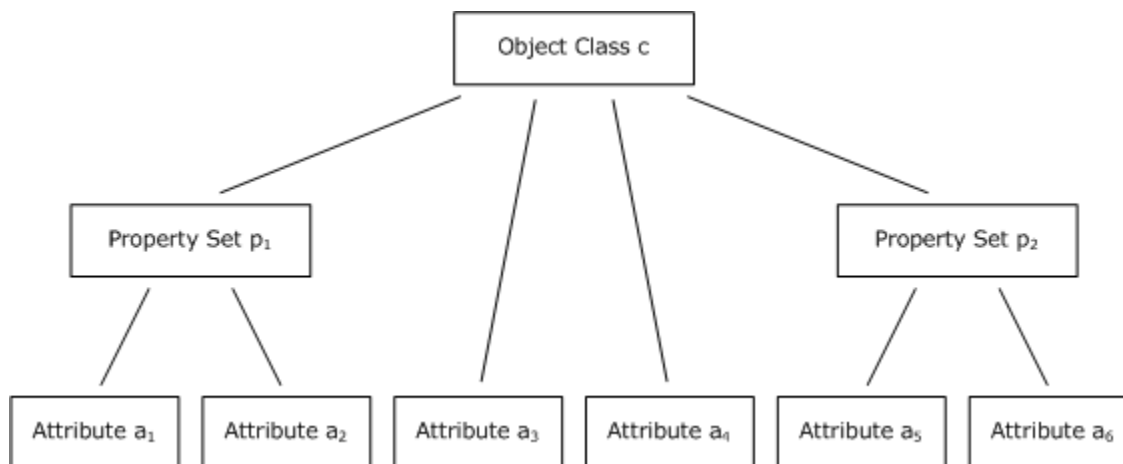
### 5.1.3.3.3 Checking Object Specific Access

This section describes how object specific access rights on Active Directory objects are evaluated with the exception of access rights representing **control access rights** and validated write rights. That is the subject of the subsequent sections.

When evaluating object specific access rights specified in object-specific ACEs for an Active Directory object, the security descriptor of the object (or its parent) is used along with a 3-level "object type tree" associated with that object. For an object O that is the subject of an access check, the object type tree  $T(V, E)$  consists of nodes  $V = \{v_1, v_2, \dots\}$ , edges  $E = \{e_1, e_2, \dots\}$ , and a **GUID**-valued label for each node in  $V$  indicated by  $Guid(v)$ , and is constructed as follows:

- Let O be an object of class c, and let  $A = \{a_1, a_2, \dots\}$  be the set of attributes that instances of class c may contain. For each attribute  $a_i$  that is an element of A, if  $a_i.attributeSecurityGUID \neq \text{NULL}$  then let  $p_i$  denote the property set of which  $a_i$  is a member and let  $Guid(p_i) = a_i.attributeSecurityGUID$  (see Property Set in section 3.1.1.2). Let P be the union of all such sets  $\{p_i\}$ .
- Add c to V as the root node of the tree and set  $Guid(c)$  to  $c!schemaIDGUID$ .
- For every property set  $p_i$  that is an element of P, add a node  $p_i$  to V and  $Guid(p_i)$  is as specified earlier.
- For every attribute  $a_i$  that is an element of A, add a node  $a_i$  to V and set  $Guid(a_i)$  to  $a_i!schemaIDGUID$ .
- For every property set  $p_i$  that is an element of P, add an edge  $(c, p_i)$  to E such that  $p_i$  is a child of c.
- For every attribute  $a_i$  that is an element of A, if there exists a property set  $p_i$  that is an element of P of which  $a_i$  is a member then add an edge  $(p_i, a_i)$  to E such that  $a_i$  is a child of  $p_i$ ; otherwise add an edge  $(c, a_i)$  to E such that  $a_i$  is a child of c.

**Note** The object type tree used during an **access check** can include only a subset of the property set (see Property set in section 3.1.1.2) nodes and a subset of the attribute nodes that the requestor is interested in. An object type tree for an object is illustrated by the figure below.



**Figure 1: An object type tree**

Let  $r$  be the root node of the object type tree  $T$ . Further, label each node  $v$  that is an element of  $V$  with two additional labels called  $\text{Grant}(v)$  and  $\text{Deny}(v)$  indicating the access rights that are granted and denied, respectively, at that node. Set both labels to a value 0 initially for every node.

The following rules are used to determine the authorization for the requestor's security context:

1. If the security descriptor of object  $O$  has no DACL or its "DACL Present" (DP) bit is not set, then grant the requestor all possible access rights on the object.
2. If the DACL does not have any ACE, then grant the requestor no access rights on the object.
3. Evaluate the DACL by examining each ACE in sequence, starting with the first ACE. Perform the following sequence of actions for each ACE in the order as shown. Let the `ACCESS_MASK` field of the ACE have a value  $M$ .

1. If the "Inherit Only" (IO) flag is set in the ACE, skip the ACE.
2. If the SID in the ACE does not match any SID in the requestor's security context, skip the ACE.
3. If the ACE type is "Access Allowed" and the access rights in  $M$  are not in  $\text{Deny}(r)$ , then add the rights in  $M$  to  $\text{Grant}(r)$ . (Where  $r$  denotes the root node of object type tree  $T$  as stated above.) For every descendant node  $u$  of  $r$ , if the rights in  $M$  are not in  $\text{Deny}(u)$ , then add the rights in  $M$  to  $\text{Grant}(u)$ .
4. If the ACE type is "Object Access Allowed" and the `ObjectType` field in the ACE is not present, then treat the ACE type as "Access Allowed" and perform the action in 3.c.
5. If the ACE type is "Object Access Allowed" and the `ObjectType` field in the ACE contains a **GUID** value  $g$ :

If there exists no node  $v$  that is an element of  $V$  such that  $\text{Guid}(v) = g$ , then skip the ACE.

Otherwise, let  $v$  that is an element of  $V$  be the unique node such that  $\text{Guid}(v) = g$ . If the rights in  $M$  are not in  $\text{Deny}(v)$ , then add the rights in  $M$  to  $\text{Grant}(v)$ . For every descendant node  $u$  of  $v$ , if the rights in  $M$  are not in  $\text{Deny}(u)$ , then add the rights in  $M$  to  $\text{Grant}(u)$ .

1. If  $v = r$ , then proceed to the next ACE.
2. If  $\text{Grant}(v) = \text{Grant}(s)$  for every sibling  $s$  of node  $v$ , then add the rights in  $\text{Grant}(v)$  to  $\text{Grant}(p)$  where  $p$  is the parent of node  $v$ . Otherwise, proceed to the next ACE.
3. Set  $v$  to  $p$ , and repeat these three steps.
6. If the ACE type is "Access Denied" and the access rights in  $M$  are not in  $\text{Grant}(r)$ , then add the rights in  $M$  to  $\text{Deny}(r)$ . For every descendant node  $u$  below the root node, if the rights in  $M$  are not in  $\text{Grant}(u)$ , then add the rights in  $M$  to  $\text{Deny}(u)$ .
7. If the ACE type is "Object Access Denied" and the `ObjectType` field in the ACE is not present, then treat the ACE type as "Access Denied" and perform the action in 3.f.
8. If the ACE type is "Object Access Denied" and the `ObjectType` field in the ACE contains a **GUID** value  $g$ :

If there exists no node  $v$  that is an element of  $V$  such that  $\text{Guid}(v) = g$ , then skip the ACE.

Otherwise let  $v$  be the unique node in  $P$  such that  $\text{Guid}(v) = g$  if any such node exists. If no such node exists let  $v$  be the unique node in  $A$  such that  $\text{Guid}(v) = g$ . If the rights in  $M$  are not

in Grant(v), then add the rights in M to Deny(v). For every descendant node u of v, if the rights in M are not in Grant(u), then add the rights in M to Deny(u). For every ancestor node w of v, add the rights in M to Deny(w).

4. When the end of the DACL is reached, the access rights in Grant(r) at the root node of tree T is the maximum access available to the requestor on the object. For each node u below the root node r, the access rights in Grant(u) is the maximum access available to the requestor for that node.

If the requested access is for the entire object, check the requested **access mask** against the access rights granted in Grant(r). If the requested access is for specific properties on the object, check the requested **access mask** against the rights granted in Grant(u) where u is the attribute node in tree T that is the target of the request.

#### 5.1.3.3.4 Checking Control Access Right-Based Access

When evaluating the right to perform an operation controlled by a **control access right** identified by the **GUID** value G, use the following rules to determine the authorization for the requestor's security context:

1. If the security descriptor has no DACL or its "DACL Present" (DP) bit is not set, then grant the requestor the requested **control access right**.
2. If the DACL does not have any ACE, then deny the requestor the requested **control access right**.
3. Evaluate the DACL by examining each ACE in sequence, starting with the first ACE. Perform the following sequence of actions for each ACE in the order as shown. Let the ACCESS\_MASK field of the ACE have a value M.
  1. If the "Inherit Only" (IO) flag is set in the ACE, skip the ACE.
  2. If the SID in the ACE does not match any SID in the requestor's security context, skip the ACE.
  3. If the ACE type is "Object Access Allowed", the access right RIGHT\_DS\_CONTROL\_ACCESS (CR) is present in M, and the ObjectType field in the ACE is not present, then grant the requested **control access right**. Stop any further access checking.
  4. If the ACE type is "Object Access Allowed" the access right RIGHT\_DS\_CONTROL\_ACCESS (CR) is present in M, and the ObjectType field in the ACE contains a **GUID** value equal to G, then grant the requested **control access right**. Stop any further access checking.
  5. If the ACE type is "Object Access Denied", the access right RIGHT\_DS\_CONTROL\_ACCESS (CR) is present in M, and the ObjectType field in the ACE is not present, then deny the requested **control access right**. Stop any further access checking.
  6. If the ACE type is "Object Access Denied" the access right RIGHT\_DS\_CONTROL\_ACCESS (CR) is present in M, and the ObjectType field in the ACE contains a **GUID** value equal to G, then deny the requested **control access right**. Stop any further access checking.

#### 5.1.3.3.5 Checking Validated Write-Based Access

When evaluating the right to perform an operation controlled by a validated write access right identified by the **GUID** value G, use the following rules to determine the authorization for the requestor's security context:

1. If the security descriptor has no DACL or its "DACL Present" (DP) bit is not set, then grant the requestor the requested validated write right.
2. If the DACL does not have any ACE, then deny the requestor the requested validated write right.
3. Evaluate the DACL by examining each ACE in sequence, starting with the first ACE. Perform the following sequence of actions for each ACE in the order as shown. Let the ACCESS\_MASK field of the ACE have a value M.
  1. If the "Inherit Only" (IO) flag is set in the ACE, skip the ACE.
  2. If the SID in the ACE does not match any SID in the requestor's security context, skip the ACE.
  3. If the ACE type is "Object Access Allowed", the access right RIGHT\_DS\_WRITE\_PROPERTY\_EXTENDED (VW) is present in M, and the ObjectType field in the ACE is not present, then grant the requested validated write right. Stop any further access checking.
  4. If the ACE type is "Object Access Allowed" the access right RIGHT\_DS\_WRITE\_PROPERTY\_EXTENDED (VW) is present in M, and the ObjectType field in the ACE contains a **GUID** value equal to G, then grant the requested validated write right. Stop any further access checking.
  5. If the ACE type is "Object Access Denied", the access right RIGHT\_DS\_WRITE\_PROPERTY\_EXTENDED (VW) is present in M, and the ObjectType field in the ACE is not present, then deny the requested validated write right. Stop any further access checking.
  6. If the ACE type is "Object Access Denied" the access right RIGHT\_DS\_WRITE\_PROPERTY\_EXTENDED (VW) is present in M, and the ObjectType field in the ACE contains a **GUID** value equal to G, then deny the requested validated write right. Stop any further access checking.

#### 5.1.3.3.6 Checking Object Visibility

An object in Active Directory is considered to be *visible* to a requestor if the requestor can see the name of the object and thus learn of its existence, even if the requestor can see no other attributes of the object. The default behavior of Active Directory with respect to making objects visible to a requesting principal is as follows:

- If a user is granted the RIGHT\_DS\_LIST\_CONTENTS access right on a **container**, all child objects of that **container** are visible to the user.
- Otherwise (if a user is not granted the RIGHT\_DS\_LIST\_CONTENTS access right on a **container**), no child object of that **container** is visible to the user. This allows the contents of entire **containers** to be hidden.

However, Active Directory can optionally be put into a special mode, called the "List Object" mode. Active Directory is put into the "List Object" mode by setting the third character of [dSHeuristics](#) (section [7.1.1.2.4.1.2](#)) to the value '1'. The mode is disabled by setting the same character to the value '0'. The default setting is '0'.

In "List Object" mode, a requestor is allowed to selectively view specific child objects of a **container** while other child objects remain hidden. In this mode, an object is visible if the user has been granted the RIGHT\_DS\_LIST\_CONTENTS right on the parent object. If, however, the user does not



have that right on the parent, then the object is visible if the user is granted the RIGHT\_DS\_LIST\_OBJECT right on both the object and its parent.

In summary, an object is **not** visible to a requestor if:

- The object is not the root object of a NC replica, and
- The requestor lacks RIGHT\_DS\_LIST\_CONTENTS right on the object's parent, and
- "List Object" mode is not set (as described above) or the requestor lacks RIGHT\_DS\_LIST\_OBJECT right on both the object and its parent.

#### 5.1.3.4 AD/LDS Security Context Construction

The construction of a Windows security context for an authenticated security principal in AD/DS is specified in [MS-PAC] section [4.2.2](#).

After a successful **Authentication** to an AD/LDS DC, the DC constructs a security context for the authenticated security principal as follows:

1. Create an initial security context.
  - If the bind named an AD/LDS user object, the initial security context contains only the [objectSid](#) of that object.
  - If the bind named an AD/LDS bind proxy, or the SID of some Windows account, the initial security context is the context returned by the Windows login.
2. Extend the security context with well-known SIDs.
  - If the bind named an AD/LDS user object or an AD/LDS bind proxy object, add the following SIDs to the security context if not already present:
    1. Authenticated Users (section [7.1.1.2.6.2](#)).
    2. Everyone (section [7.1.1.2.6.9](#)).
    3. Users, for the NC containing the AD/LDS object (section [7.1.1.4.12.3](#)).
    4. Users, for the config NC of the forest containing the AD/LDS object (section [7.1.1.4.12.3](#)).
3. Extend the security context with AD/LDS group memberships.
  - If a SID currently in the security context is a member of an AD/LDS group on this DC, and that group is not already present in the context, add the SID of that group to the context. (The group membership is represented as a reference to an object whose [objectSid](#) equals the SID: either an AD/LDS user, an AD/LDS bind proxy, an AD/LDS [group](#), or a [foreignSecurityPrincipal](#) object.) Repeat until there are no more SIDs to add.

## 6 Index of Version-Specific Behavior

Version-Specific Behavior	Section	Windows 2000	Windows Server 2003	Windows Server 2008
Capability to interpret flag ADS_UF_PARTIAL_SECRETS_ACCOUNT on userAccountControl	<a href="#">2.2.15</a>			X
LinkValueStamp support for link values	<a href="#">3.1.1.1.9</a>		X	X
Support for new schema defunct behavior	<a href="#">3.1.1.2.3</a>		X	X
Capability to generate msDS-IntId when adding a schema attribute	<a href="#">3.1.1.2.3</a>		X	X
Auto-generated linkID support	<a href="#">3.1.1.2.3.1</a>		X	X
Property sets MS-TS-GatewayAccess, Private Information, Terminal Server License Server	<a href="#">3.1.1.2.3.3</a>			X
Dynamically instantiating auxiliary classes capability	<a href="#">3.1.1.2.4.6</a>		X	X
Capability to generate msDS-IntId when adding a schema class	<a href="#">3.1.1.2.4.8</a>		X	X
Support for the new schema Defunct behavior	<a href="#">3.1.1.2.5.2.2</a>		X	X
Attribute attributeTypes, dITContentRules, extendedAttributeInfo, extendedClassInfo, and objectClasses attributes do not contain defunct attributes or classes.	<a href="#">3.1.1.3.1.1.1</a>		X	X
Support for dashed-string form of <b>GUID</b> DN	<a href="#">3.1.1.3.1.2.4</a>		X	X
Support for SDDL SID form in DN	<a href="#">3.1.1.3.1.2.4</a>		X	X
Three-value logic for search filter evaluation	<a href="#">3.1.1.3.1.3.1</a>		X	X
Capability to allow modification of the unicodePwd attribute over an unencrypted connection.	<a href="#">3.1.1.3.1.5.1</a>			X
Support for modification of the unicodePwd attribute on a connection protected by 128-bit (or better) SASL-layer encryption	<a href="#">3.1.1.3.1.5.1</a>		X	X

Version-Specific Behavior	Section	Windows 2000	Windows Server 2003	Windows Server 2008
Support for modifying the userPassword attribute	<a href="#">3.1.1.3.1.5.2</a>		X	X
Support for dynamic objects	<a href="#">3.1.1.3.1.6</a>		X	X
Support for rootDSE attributes: domainControllerFunctionality, domainFunctionality, forestFunctionality, msDS-ReplicationInboundNeighbors, msDS-ReplicationOutboundNeighbors, msDS-ReplicationConnectionFailures, msDS-ReplicationLinkFailures, msDS-ReplicationPendingOps, msDS-ReplicationQueueStatistics, msDS-TopQuotaUsage, supportedConfigurableSettings, supportedExtension, validFSMOs	<a href="#">3.1.1.3.2</a>		X	X
Support for rootDSE attribute dsaVersionString, msDS-PortLDAP, msDS-PortSSL, msDS-PrincipalName, serviceAccountInfo, spnRegistrationResult, tokenGroups, usnAtRifm	<a href="#">3.1.1.3.2</a>			X
Support for rootDSE Modify Operations: doLinkCleanup, doOnlineDefrag, replicateSingleObject, updateCachedMemberships, doGarbageCollection, doGarbageCollectionPhantomsNow	<a href="#">3.1.1.3.3</a>		X	X
Support for rootDSE Modify Operations: invalidateGCCConnection, renewServerCertificate	<a href="#">3.1.1.3.3</a>			X
Support for special values for rootDSE modify operation fixupInheritance	<a href="#">3.1.1.3.3.1.0</a>		X	X
Support for <b>LDAP</b> extended controls: LDAP_CONTROL_VLVREQUEST, LDAP_CONTROL_VLVRESPONSE, LDAP_SERVER_QUOTA_CONTROL_OID, LDAP_SERVER_SHUTDOWN_NOTIFY_OID	<a href="#">3.1.1.3.4.1</a>		X	X
Support for <b>LDAP</b> extended controls: LDAP_SERVER_FORCE_UPDATE_OID, LDAP_SERVER_RANGE_RETRIEVAL_NOERR_OID, LDAP_SERVER_RODC_DCPROMO_OID	<a href="#">3.1.1.3.4.1</a>			X
Support for dir sync flag LDAP_DIRSYNC_OBJECT_SECURITY, LDAP_DIRSYNC_PUBLIC_DATA_ONLY, LDAP_DIRSYNC_INCREMENTAL_VALUES	<a href="#">3.1.1.3.4.1.3</a>		X	X
Accepting controlValue field when using LDAP_SERVER_EXTENDED_DN_OID	<a href="#">3.1.1.3.4.1.5</a>		X	X

Version-Specific Behavior	Section	Windows 2000	Windows Server 2003	Windows Server 2008
Accepting controlValue field when using LDAP_SERVER_GET_STATS_OID	<a href="#">3.1.1.3.4.1.6</a>		X	X
Support to use a base DN that is not a root of a NC in subtree search when using LDAP_SERVER_NOTIFICATION_OID control	<a href="#">3.1.1.3.4.1.9</a>		X	X
Support ording rules when using LDAP_SERVER_SORT_OID and LDAP_SERVER_RESP_SORT_OID controls	<a href="#">3.1.1.3.4.1.13</a>		X	X
Support for phonetic display name sort	<a href="#">3.1.1.3.4.1.13</a>			X
Support for <b>LDAP</b> extended operations: LDAP_SERVER_FAST_BIND_OID, LDAP_SERVER_START_TLS_OID, LDAP_TTL_REFRESH_OID	<a href="#">3.1.1.3.4.2</a>		X	X
Support for <b>LDAP</b> extended operations: LDAP_SERVER_WHO_AM_I_OID	<a href="#">3.1.1.3.4.2</a>			X
Support for <b>LDAP</b> capacities: LDAP_CAP_ACTIVE_DIRECTORY_V51_OID	<a href="#">3.1.1.3.4.3</a>		X	X
Support for <b>LDAP</b> capacities: LDAP_CAP_ACTIVE_DIRECTORY_ADAM_DIGEST , LDAP_CAP_ACTIVE_DIRECTORY_ADAM_OID, LDAP_CAP_ACTIVE_DIRECTORY_PARTIAL_SECRETS_OID, LDAP_CAP_ACTIVE_DIRECTORY_V61_OID	<a href="#">3.1.1.3.4.3</a>			X
Support for LDAP_MATCHING_RULE_TRANSITIVE_EVAL <b>LDAP</b> Matching Rules	<a href="#">3.1.1.3.4.4</a>			X
Support for <b>LDAP</b> SASL mechanisms: EXTERNAL, DIGEST-MD5	<a href="#">3.1.1.3.4.5</a>		X	X
Support for LDAP policies: MaxActiveQueries	<a href="#">3.1.1.3.4.6</a>	X		
Support for LDAP policies: MaxValRange	<a href="#">3.1.1.3.4.6</a>		X	X
Support for <b>LDAP</b> configurable settings: DynamicObjectDefaultTTL, DynamicObjectMinTTL, DisableVLVSupport	<a href="#">3.1.1.3.4.7</a>		X	X
Support for <b>LDAP</b> configurable settings: ADAMAllowADAMSecurityPrincipalsInConfigPartition, ADAMDisableLogonAuditing, ADAMDisablePasswordPolicies, ADAMDisableSPNRegistration, ADAMDisableSSI,	<a href="#">3.1.1.3.4.7</a>			X

Version-Specific Behavior	Section	Windows 2000	Windows Server 2003	Windows Server 2008
ADAMLastLogonTimestampWindow, MaxReferrals, ReferralRefreshInterval, RequireSecureProxyBind, RequireSecureSimpleBind, SelfReferralsOnly				
Attribute msDS-RevealedList	<a href="#">3.1.1.4.5.3.4</a>			X
Attribute msDS-RevealedListBL	<a href="#">3.1.1.4.5.3.5</a>			X
New behavior when reading defunct attributes and classes	<a href="#">3.1.1.4.8</a>		X	X
Modifications of msDS-AdditionalDnsHostName are allowed	<a href="#">3.1.1.5.3.2</a>		X	X
Intra-site connection creation for read-only replica	<a href="#">7.2.2.2</a>			X
Capability to interpret flag NETLOGON_NT_VERSION_WITH_CLOSEST_SITE in NETLOGON_NT_VERSION Options Bits	<a href="#">7.3.1.1</a>			X

## 7 Additional Information

### 7.1 Special Objects and Forest Invariants

This section specifies the objects that are necessary for the proper functioning of the DCs in a forest and the invariants that govern the state of these objects.

#### 7.1.1 Special Objects

##### 7.1.1.1 Naming Contexts

References:

- Special Attributes: Well-known Objects, Other Well-known Objects, Behavior Version
- Forest Invariants
- FSMO Roles
- State Model: NC Naming
- Security: SD Reference Domain

Variables: upToDateVector, DN-Binary, Security Descriptor

Glossary Terms: NC, NC Replica, NC root, DC, Forest root, **Domain** NC, PDC, FSMO

LDAP attributes: [instanceType](#), [subRefs](#), [repsTo](#), [repsFrom](#), [replUpToDateVector](#), [wellKnownObjects](#), [otherWellKnownObjects](#), [name](#), [objectClass](#), [nTSecurityDescriptor](#), [fSMORoleOwner](#), [msDS-Behavior-Version](#), [distinguishedName](#), [systemFlags](#), [nTMixedDomain](#), [domainReplica](#), [msDS-AllowedDNSSuffixes](#), [dNSHostName](#), [msDS-AdditionalDnsHostName](#), [msDS-SDReferenceDomain](#)

LDAP classes: [configuration](#), [dMD](#), [domainDNS](#)

Constants

- Access mask bits, CARs: DS-Replication-Get-Changes, DS-Replication-Get-Changes-All, DS-Replication-Get-Changes-In-Filtered-Set
- [systemFlags](#) bits: FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE

##### 7.1.1.1.1 Any NC Root

The following attributes have constant semantics across all types of NCs.

[instanceType](#): The [instanceType](#) of an NC root is a bitfield, which is presented here in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	G	C	A	W	U	H

**X**: Unused. Must be zero and ignored.

**H (IT\_NC\_HEAD):** This flag is set (value 1) on all NC roots.

**U (IT\_UNINSTANT):** If this flag is set, the NC replica which this root represents does not exist locally.

**W (IT\_WRITE):** This flag is written locally based upon the desired NC replica type. A regular NC replica will have this flag set, and a **partial NC replica** will not have this flag set. The IT\_WRITE flag must be propagated identically to every object in the NC replica.

**A (IT\_NC\_ABOVE):** This flag indicates that the **local DC** holds an instantiated NC replica which is a parent of the NC replica represented by this NC root.

**C (IT\_NC\_COMING):** This flag indicates that the NC replica has not completed its initial replication into the local DC and may not have a full set of objects in the NC represented by this NC root.

**G (IT\_NC\_GOING):** This flag indicates that the NC replica is being removed from the local DC and may not have a full set of objects in the NC represented by this NC root.

Invariants:

- IT\_UNINSTANT can only be set with IT\_NC\_HEAD. The rest of the bits are incompatible with IT\_UNINSTANT.
- IT\_NC\_COMING and IT\_NC\_GOING cannot be set at the same time.
- If IT\_NC\_GOING, then no replication can occur with that NC, either as server or as client.

[subRefs](#): This value references all child objects in this NC replica of this NC root that are, themselves, NC roots. For example, the schema NC is always referenced by this value on the Config NC root object.

[repsTo](#): This attribute contains the abstract attribute [repsTo](#) associated with this DC for this NC replica. This attribute is non-replicated. [MS-DRSR] section [5:repsTo](#) specifies this abstract attribute.

[repsFrom](#): This attribute contains the abstract attribute [repsFrom](#) associated with this DC for this NC replica. This attribute is non-replicated. [MS-DRSR] section [5:repsFrom](#) specifies this abstract attribute.

[replUpToDateVector](#): This attribute contains the abstract attribute [replUpToDateVector](#) associated with this DC for this NC replica. This attribute is non-replicated. [MS-DRSR] section [5:replUpToDateVector](#) specifies this abstract attribute.

#### 7.1.1.1.2 Config NC Root

[name](#): Configuration

[parent](#): For AD/DS, the forest root NC root object. For AD/LDS, no parent.

[objectClass](#): [configuration](#)

[wellKnownObjects](#): This attribute holds DN-Binary values. See section [7.1.4](#) for details.

[instanceType](#): This value can never contain the following flags:

- IT\_NC\_COMING
- IT\_NC\_GOING

- IT\_UNINSTANT

[nTSecurityDescriptor](#):

- Let D1 be a DC that is instructed to host a writable replica of the config NC (see section [7.1.2.3](#) for hosting invariants). In order for D1 to replicate the config NC, D1 must be granted the following rights on the config NC root:
  - DS-Replication-Get-Changes
  - DS-Replication-Get-Changes-All
  - DS-Replication-Get-Changes-In-Filtered-Set
- Let D2 be a DC that is instructed to host a read-only replica of config NC (see section [7.1.2.3](#) for hosting invariants) such that the objects in the NC replica will not contain attributes in the **filtered attribute set**. In order for D2 to replicate the config NC, D2 must be granted the following rights on the config NC root:
  - DS-Replication-Get-Changes

[msDS-ReplAuthenticationMode](#): Present and used on AD/LDS only. Specifies the **Authentication** used for DC-to-DC communication over **RPC** ([\[MS-DRSR\]](#)). The [msDS-ReplAuthenticationMode](#) values 0, 1, and 2 are valid; if absent, the effect is as if the value was 1. See [\[MS-DRSR\] section 2.2.1](#) for the effects of these values.

[objectSid](#): Present and used on AD/LDS only. This attribute contains the SID that is used in generating [objectSid](#) values for new AD/LDS **security principals** residing in the config NC, as specified in section [3.1.1.5.2.4](#). This attribute is not returned by **LDAP** queries.

#### 7.1.1.1.3 Schema NC Root

[name](#): Schema

[parent](#): Config NC root

[objectClass](#): [dMD](#)

[FSMORoleOwner](#): This value refers to the [nTDSDSA](#) object of the DC that owns the Schema Master FSMO. See section [7.1.5](#).

[instanceType](#): This value can never contain the following flags:

- IT\_NC\_COMING
- IT\_NC\_GOING
- IT\_UNINSTANT

[nTSecurityDescriptor](#): Let D be a DC which is instructed to host the schema replica NC (see section [7.1.2.3](#) for hosting invariants). In order for D to replicate the schema NC, D must be granted the following rights on the schema NC root:

- DS-Replication-Get-Changes
- DS-Replication-Get-Changes-All
- DS-Replication-Get-Changes-In-Filtered-Set



#### 7.1.1.1.4 Domain NC Root

[distinguishedName](#): See section [3.1.1.1](#) for more information about **domain** NC naming rules.

[objectClass](#): [domainDNS](#)

[fSMORoleOwner](#): This value refers to the [nTDSDSA](#) object of the DC that owns the PDC FSMO role. See section [7.1.5](#) for more information about the PDC role.

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

[wellKnownObjects](#): This attribute holds DN-Binary values. See section [7.1.4](#) for details.

[otherWellKnownObjects](#): This attribute holds DN-Binary values. See section [7.1.4](#) for details.

[msDS-Behavior-Version](#): This value defines the functional level of the **domain**. See section [7.1.4](#).

[nTMixedDomain](#): This value defines whether NT BDC replication [MS-NRPC] is available in the domain. See section [7.1.1](#).

[domainReplica](#): See section [3.1.1.5](#) for more information.

[msDS-AllowedDNSSuffixes](#): List of DNS suffixes that are allowed in the [dNSHostName](#) and [msDS-AdditionalDnsHostName](#) attributes of [computer](#) objects in this **domain**.

[nTSecurityDescriptor](#):

- Let D1 be a DC that is instructed to host a writable **domain** replica NC (see section [7.1.2.3](#) for hosting invariants). In order for D1 to replicate the **domain** NC, D1 must be granted the following rights on the **domain** NC root:
  - DS-Replication-Get-Changes
  - DS-Replication-Get-Changes-All
  - DS-Replication-Get-Changes-In-Filtered-Set
- Let D2 be a DC that is instructed to host a partial or read-only **domain** replica NC (see section [7.1.2.3](#) for hosting invariants) such that objects in the NC replica can have attributes in the **filtered attribute set**. In order for D2 to replicate the **domain** NC, D2 must be granted the following right on the **domain** NC root:
  - DS-Replication-Get-Changes
  - DS-Replication-Get-Changes-In-Filtered-Set
- Let D3 be a DC that is instructed to host a partial or read-only domain replica NC (see section [7.1.2.3](#) for hosting invariants) such that objects in the NC replica will not have attributes in the **filtered attribute set**. In order for D3 to replicate the domain NC, D3 must be granted the following right on the domain NC root:
  - DS-Replication-Get-Changes

#### 7.1.1.1.5 Application NC Root

[distinguishedName](#): See section [3.1.1.1](#) for more information about **domain** NC naming rules.

[objectClass](#): [domainDNS](#) (AD/DS); any structural or 88 class except [dMD](#) and [configuration](#) (AD/LDS)

[wellKnownObjects](#): This attribute holds DN-Binary values. See section [7.1.4](#) for details.

[otherWellKnownObjects](#): This attribute holds DN-Binary values. See section [7.1.4](#) for details.

[nTSecurityDescriptor](#):

- Let D1 be a DC that is instructed to host a writable application replica NC (see section [7.1.2.3](#) for hosting invariants). In order for D1 to replicate the NC, D1 must be granted the following rights on the NC root:
  - DS-Replication-Get-Changes
  - DS-Replication-Get-Changes-All
  - DS-Replication-Get-Changes-In-Filtered-Set
- Let D2 be a DC which is instructed to host a readonly application replica NC (see section [7.1.2.3](#) for hosting invariants) such that objects in the NC replica will not contain attributes in **filtered attribute set**. In order for D2 to replicate the NC, D2 must be granted the following rights on the NC root:
  - DS-Replication-Get-Changes
- Note that this [nTSecurityDescriptor](#) must be resolved with the **domain** specified on the [msDS-SDReferenceDomain](#) attribute on the [crossRef](#) object representing this NC, see section [5](#) for details.

[objectSid](#): Present and used on AD/LDS only. This attribute contains the SID that is used in generating [objectSid](#) values for new AD/LDS **security principals** residing in this application NC, as specified in section [3.1.1.5.2.4](#). This attribute is not returned by **LDAP** queries.

### 7.1.1.2 Configuration Objects

References:

- FSMO Roles
- LDAP Protocol
- Special Attributes
- Forest Invariants
- Security
- Knowledge Consistency Checker
- Originating Updates

Glossary Terms: NC, NC Replica, NC root, DC, **Domain** NC, FSMO, Forest Functional Level, Application NC, KCC, **ISTG**, Intra-site, Inter-site, Global Catalog, Forest, SMTP, Site, **COM** (**Component Object Model**), **UUID**, MAPI, ANR, **NSPI**

LDAP attributes: [name](#), [objectClass](#), [fsmoRoleOwner](#), [msDS-Behavior-Version](#), [distinguishedName](#), [systemFlags](#), [nTMixedDomain](#), [dnsRoot](#), [nCName](#), [msDS-Replication-Notify-First-DSA-Delay](#), [msDS-](#)

[Replication-Notify-Subsequent-DSA-Delay](#), [nETBIOName](#), [msDS-SDReferenceDomain](#), [options](#), [schedule](#), [interSiteTopologyGenerator](#), [interSiteTopologyFailover](#), [interSiteTopologyRenew](#), [serverReference](#), [dNSHostName](#), [mailAddress](#), [invocationId](#), [hasMasterNCs](#), [hasPartialReplicaNCs](#), [msDS-HasInstantiatedNCs](#), [instanceType](#), [msDS-HasDomainNCs](#), [msDS-hasMasterNCs](#), [msDS-ReplicationEpoch](#), [enabledConnection](#), [fromServer](#), [transportType](#), [mS-DS-ReplicatesNCReason](#), [siteObject](#), [transportDLLName](#), [transportAddressAttribute](#), [cost](#), [siteList](#), [replInterval](#), [siteLinkList](#), [adminPropertyPages](#), [shellPropertyPages](#), [adminContextMenu](#), [shellContextMenu](#), [adminMultiselectPropertyPages](#), [treatAsLeaf](#), [creationWizard](#), [createWizardExt](#), [dSHeuristics](#), [objectGUID](#), [msDS-KeyVersionNumber](#), [tombstoneLifetime](#), [sPNMappings](#), [msDS-Other-Settings](#), [rightsGuid](#), [appliesTo](#), [localizationDisplayId](#), [validAccesses](#), [repsTo](#)

LDAP classes: [crossRefContainer](#), [crossRef](#), [sitesContainer](#), [site](#), [nTDSSiteSettings](#), [nTDSConnection](#), [serversContainer](#), [server](#), [nTDSDSA](#), [subnetContainer](#), [subnet](#), [interSiteTransportContainer](#), [interSiteTransport](#), [siteLink](#), [container](#), [displaySpecifier](#), [nTDSservice](#), [physicalLocation](#), [controlAccessRight](#)

Constants:

- [systemFlags](#) bits: FLAG\_DISALLOW\_DELETE, FLAG\_CR\_NTDS\_NC, FLAG\_CR\_NTDS\_DOMAIN, FLAG\_CR\_NTDS\_NOT\_GC\_REPLICATED, FLAG\_DISALLOW\_MOVE\_ON\_DELETE, FLAG\_CONFIG\_ALLOW\_LIMITED\_MOVE, FLAG\_CONFIG\_ALLOW\_RENAME
- Replication bits: DRS\_SYNC\_FORCE, DRS\_SELECT\_SECRET\_REP
- Extended replication operation: EXOP\_REPL\_SECRETS

#### 7.1.1.2.1 Cross-Ref-Container Container

[name](#): Partitions

[parent](#): Config NC root

[objectClass](#): [crossRefContainer](#)

[fSMORoleOwner](#): This value references the Domain Naming Master FSMO role owner. See section [7.1.5](#).

[systemFlags](#): {FLAG\_DISALLOW\_DELETE}

[msDS-Behavior-Version](#): This value defines the forest functional level. See section [7.1.4](#).

##### 7.1.1.2.1.1 Cross-Ref Objects

The following is the description of the flags and their meaning for [crossRef](#) objects stored in [systemFlags](#). The flags are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	G	D	N
																												C		C	

**X**: Unused. Must be zero and ignored.

**NC (FLAG\_CR\_NTDS\_NC)**: NC exists within the forest (not external).

**D (FLAG\_CR\_NTDS\_DOMAIN):** NC is a **domain**.

**GC (FLAG\_CR\_NTDS\_NOT\_GC\_REPLICATED):** NC must not be replicated to GC servers as a read only replica.

The following attributes and attribute values are common to [crossRef](#) objects representing all NC types:

[parent](#): [crossRefContainer](#) object

[objectClass](#): [crossRef](#)

[Enabled](#): If false, this is a "pre-created" [crossRef](#); that is, the [crossRef](#) exists but the corresponding NC root does not yet exist. See section [3.1.1.5.2.8](#).

[dnsRoot](#): If [Enabled](#) equals false, in AD/DS [dnsRoot](#) holds the DNS name of the DC that will create the root of this NC. If [Enabled](#) equals false, in AD/LDS, [dnsRoot](#) holds the DNS name of the DC that will create the root of this NC followed by a colon (":") followed by the **LDAP** port number used by the DC. If [Enabled](#) is not false, in AD/DS [dnsRoot](#) holds the fully qualified DNS name used for **LDAP** referrals (section [3.1.1.4.6](#)). If [Enabled](#) is not false, in AD/LDS [dnsRoot](#) is absent.

[nCName](#): If [Enabled](#) is not false, a reference to the NC root corresponding to this [crossRef](#).

[msDS-Replication-Notify-First-DSA-Delay](#): Indicates a number of seconds that each DC should delay after receiving updates (originating or replicated) to objects in the NC referred to by [nCName](#) before the DC notifies another DC of updates received according to the DCs local repsTos. See IDL\_DRSReplicaSync in [MS-DRSR] section [4.1.23](#).

[msDS-Replication-Notify-Subsequent-DSA-Delay](#): Indicates a number of seconds that each DC should delay after notifying the first DC of updates received to objects in the NC referred to by [nCName](#) before notifying each additional DC according to the DCs local repsTos. See IDL\_DRSReplicaSync in [MS-DRSR] section [4.1.23](#).

#### 7.1.1.2.1.1.1 Foreign crossRef Objects

A foreign [crossRef](#) object is used to enable referrals for searches that need to return objects from different forests or **LDAP** services. For more information see section [3.1.1.3](#). The following attribute and attribute values are defined for a foreign [crossRef](#):

[systemFlags](#): 0

#### 7.1.1.2.1.1.2 Configuration crossRef Object

[name](#): Enterprise Configuration

[systemFlags](#): { FLAG\_CR\_NTDS\_NC }

[nCName](#): The value must equal the config NC root.

[dnsRoot](#): In AD/DS, the value is the forest root's fully qualified DNS name. Not present in AD/LDS.

#### 7.1.1.2.1.1.3 Schema crossRef Object

[name](#): Enterprise Schema

[systemFlags](#): { FLAG\_CR\_NTDS\_NC }

[nCName](#): The value must equal the schema NC root.

[dnsRoot](#): In AD/DS, the value is the forest root's fully qualified DNS name. Not present in AD/LDS.

#### 7.1.1.2.1.1.4 Domain crossRef Object

The following attribute and attribute values are common to **domain** [crossRef](#) objects:

[name](#): The NetBIOS name of the domain.

[nCName](#): The reference must be to a **domain** NC root.

[nETBIOSName](#): This value is the NetBIOS name of the **domain**.

[nTMixedDomain](#): This value is read-only on this object. It is kept in sync with the same attribute on the NC root of the NC referred to by [nCName](#). See section [7.1.1](#).

[systemFlags](#): { FLAG\_CR\_NTDS\_NC | FLAG\_CR\_NTDS\_DOMAIN }

[msDS-Behavior-Version](#): This value is read-only on this object. It is kept in sync with the same attribute on the NC root of the NC referred to by [nCName](#). See section [7.1.4](#).

#### 7.1.1.2.1.1.5 Application NC crossRef Object

[dnsRoot](#): In AD/DS, the value for [dnsRoot](#) for an application NC [crossRef](#) is derived by syntactically converting the DN portion of the [crossRef](#)'s [nCName](#) into a fully qualified DNS name as specified in section [3.1.1.1.5](#). Not present in AD/LDS.

[systemFlags](#): { FLAG\_CR\_NTDS\_NC | FLAG\_CR\_NTDS\_NOT\_GC\_REPLICATED }

[msDS-NC-Replica-Locations](#): This attribute references the [nTDSDSA](#) objects representing every DC instructed to hold a writable NC replica of this application NC. See Hosting invariants in section [7.1.2.3](#).

[msDS-SDReferenceDomain](#): In AD/DS, the attribute references an NC root object for a **domain**. All security descriptors in this application NC must use the NC represented as the reference **domain** for resolution. See section [5](#) for security descriptor reference **domain** information. Not present in AD/LDS.

[msDS-NC-RO-Replica-Locations](#): This attribute references the [nTDSDSA](#) object representing every DC instructed to hold an read-only NC replica of this application NC. See hosting invariants in section [7.1.2.3](#).

#### 7.1.1.2.2 Sites Container

Each forest contains a Sites **container** in the Config NC. For each site in the forest, a [site](#) object exists in the Sites **container**.

[name](#): Sites

[parent](#): Config NC root object

[objectClass](#): [sitesContainer](#)

[systemFlags](#): { FLAG\_DISALLOW\_DELETE | FLAG\_DISALLOW\_MOVE\_ON\_DELETE }

### 7.1.1.2.2.1 Site Object

A [site](#) object corresponds to a set of one or more IP subnets that have LAN connectivity. Thus, by virtue of their subnet associations, DCs that are in the same site are well connected in terms of speed. Each [site](#) object has a child [nTDSiteSettings](#) object and a Servers **container**.

[parent](#): Sites container

[objectClass](#): [site](#)

[systemFlags](#): { FLAG\_CONFIG\_ALLOW\_RENAME | FLAG\_DISALLOW\_MOVE\_ON\_DELETE }

**Note** The initial AD/DS and AD/LDS configuration contains one initial site object named Default-First-Site-Name, which has no subnet association.

#### 7.1.1.2.2.1.1 NTDS Site Settings Object

NTDS **site settings objects** identify site-wide settings. There is one [nTDSiteSettings](#) object per site.

[name](#): NTDS Site Settings

[parent](#): [site](#) object.

[objectClass](#): [nTDSiteSettings](#)

[options](#): One or more bits from the following diagram. The bits are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	R	S	B	W	F	G	I	D	M	T	A
																					S	H	H	2	W	C	S	S	H	C	T
																					E	E	D	K	B	E	D	D	D	D	D

**X**: Unused. Must be zero and ignored.

**ATD (NTDSSETTINGS\_OPT\_IS\_AUTO\_TOPOLOGY\_DISABLED)**: Automatic topology generation is disabled. See section [7.2](#) for more information.

**TCD (NTDSSETTINGS\_OPT\_IS\_TOPL\_CLEANUP\_DISABLED)**: Automatic topology cleanup is disabled. See section [7.2](#) for more information.

**MHD (NTDSSETTINGS\_OPT\_IS\_TOPL\_MIN\_HOPS\_DISABLED)**: Automatic minimum hops topology is disabled. See section [7.2](#) for more information.

**DSD (NTDSSETTINGS\_OPT\_IS\_TOPL\_DETECT\_STALE\_DISABLED)**: Automatic stale server detection is disabled. See section [7.2](#) for more information.

**ISD (NTDSSETTINGS\_OPT\_IS\_INTER\_SITE\_AUTO\_TOPOLOGY\_DISABLED)**: Automatic inter-site topology generation is disabled. See section [7.2](#) for more information.

**GCE (NTDSSETTINGS\_OPT\_IS\_GROUP\_CACHING\_ENABLED)**: Caching of users' group memberships is enabled for this site. This caching is an implementation-specific behavior. This flag

may be ignored by other implementations, but must not be used in a conflicting way that would affect the performance of Windows DCs.

**FWB (NTDSSETTINGS\_OPT\_FORCE\_KCC\_WHISTLER\_BEHAVIOR)**: Force the KCC to behave in a forest functional level of DS\_BEHAVIOR\_WIN2003 or greater. See section [7.2](#) for more information.

**W2K (NTDSSETTINGS\_OPT\_FORCE\_KCC\_W2K\_ELECTION)**: Force the KCC to use the Windows 2000 ISTG election algorithm. See section [7.2](#) for more information.

**BHD (NTDSSETTINGS\_OPT\_IS\_RAND\_BH\_SELECTION\_DISABLED)**: Prevent the KCC from randomly picking a **bridgehead** when creating a connection. See section [7.2](#) for more information.

**SHE (NTDSSETTINGS\_OPT\_IS\_SCHEDULE\_HASHING\_ENABLED)**: Allow the KCC to use hashing when creating a replication schedule. See section [7.2](#) for more information.

**RSE (NTDSSETTINGS\_OPT\_IS\_REDUNDANT\_SERVER\_TOPOLOGY\_ENABLED)**: Create static failover connections. See section [7.2](#) for more information.

[schedule](#): The default replication schedule (defined as a SCHEDULE structure), that applies to all [nTDSConnection](#) objects for intra-site replication within this site. If this attribute does not contain any value, a schedule of once per hour is applied to replication within this site. See section [7.2](#) for more information.

[interSiteTopologyGenerator](#): A reference to the [nTDSDSA](#) object of the DC which is acting as the ISTG for this site. See section [7.2](#) for more information on the ISTG.

[interSiteTopologyFailover](#): Indicates how much time must transpire since the last keep-alive for the **inter-site topology generator** to be considered dead. See section [7.2](#) for more information.

[interSiteTopologyRenew](#): Indicates how often the inter-site topology generator updates the keep-alive message that is sent to **domain** controllers contained in the same site. See section [7.2](#) for more information.

#### 7.1.1.2.2.1.2 Servers Container

Each site contains a Servers **container** which contains the [server](#) objects for all the DCs that are in that site.

[parent](#): [site](#) object

[objectClass](#): [serversContainer](#)

[systemFlags](#): FLAG\_DISALLOW\_MOVE\_ON\_DELETE

##### 7.1.1.2.2.1.2.1 Server Object

Each DC in a **domain** has a [server](#) object in the config NC. See invariants in section [7.1.2.1](#). A [server](#) object has the following attributes:

[parent](#): The parent of this object is a [serversContainer](#) object.

[objectClass](#): [server](#)

[systemFlags](#): { FLAG\_CONFIG\_ALLOW\_RENAME | FLAG\_CONFIG\_ALLOW\_LIMITED\_MOVE | FLAG\_DISALLOW\_MOVE\_ON\_DELETE }

[serverReference](#): In AD/DS, a reference to the **domain** controller object representing this DC. See invariants in section [7.1.2.1](#). Not present in AD/LDS.

[dNSHostName](#): Fully qualified DNS name of the DC.

[mailAddress](#): To enable the DC to perform inter-site replication via SMTP protocol (see [\[MS-SRPL\]](#)), this attribute must contain the SMTP mail address of the server.

#### 7.1.1.2.2.1.2.1.1 nTDSDSA Object

Each DC in a **domain** has an [nTDSDSA](#) object in the config NC. See invariants in section [7.1.2.1](#). An [nTDSDSA](#) object has the following attributes:

[name](#): NTDS Settings

[parent](#): An object with [objectClass server](#).

[objectClass](#): [nTDSDSA](#)

[dMDLocation](#): dsname of the schema NC root.

[invocationId](#): The invocation ID for this DC (section [3.1.1.1.9](#)).

[options](#): One or more of the following bits presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	D	D	D	D	G
																										S	N	O	I	C		
																										X						

**X**: Unused. Must be zero and ignored.

**GC (NTDSDSA\_OPT\_IS\_GC)**: This DC is, or is becoming, a GC server.

**DI (NTDSDSA\_OPT\_DISABLE\_INBOUND\_REPL)**: This DC does not perform inbound replication unless the DRS\_SYNC\_FORCE flag is passed. See [MS-DRSR] section [4.1.10.4.1](#), ReplicateNCRequestMsg, for the effects of this option.

**DO (NTDSDSA\_OPT\_DISABLE\_OUTBOUND\_REPL)**: This DC does not perform outbound replication unless the DRS\_SYNC\_FORCE flag is passed. See [MS-DRSR] section [4.1.10.5.2](#), GetReplChanges, for the effects of this option.

**DNX (NTDSDSA\_OPT\_DISABLE\_NTDSConn\_XLATE)**: This DC does not translate connection objects into [repsFroms](#). See Section [7.2](#) for more information.

**DS (NTDSDSA\_OPT\_DISABLE\_SPN\_REGISTRATION)**: This DC does not perform SPN registration. Only interpreted by AD/LDS DCs. See [MS-DRSR] sections [2.2.3.2](#) and [2.2.4.2](#), SPN for Target DC in AD/LDS, for the effects of this option.

[systemFlags](#): {FLAG\_DISALLOW\_DELETE}

[msDS-Behavior-Version](#): Indicates the DC Version. See section [7.1.4.2](#) for more information.



[msDS-PortLDAP](#): In AD/LDS, stores the **LDAP** port for this instance. Not present in AD/DS.

[msDS-PortSSL](#): In AD/LDS, stores the SSL port for this instance. Not present in AD/DS.

[msDS-ServiceAccount](#): In AD/LDS, stores the [foreignSecurityPrincipal](#) object representing the service account running this DC. Not present in AD/DS.

[hasMasterNCs](#): Contains the dsnames of the NC root objects representing the schema NC, config NC, and **domain** NC for the default **domain** of the DC. This attribute always contains these three values, and only these three values. This attribute is not present on the [nTDSDSA](#) object of an RODC.

[hasPartialReplicaNCs](#): Contains the dsnames of the root objects of all **domain** NCs within the forest for which the DC hosts a partial NC replica.

[msDS-HasInstantiatedNCs](#): Contains an Object(DN-Binary) value for each NC replica hosted by this DC. The DN field is the DN of the root object of the NC. The Binary field contains the value of the [instanceType](#) attribute on the root object of the NC. This is a binary encoding of attribute [instanceType](#) with little-endian byte ordering.

Invariant: The DN fields of all the values of [msDS-HasInstantiatedNCs](#) must be equal to the set of DNs contained in the values of [msDS-hasMasterNCs](#) and [hasPartialReplicaNCs](#).

[msDS-HasDomainNCs](#): Equals the dsname of the NC root object for which the DC is hosting a regular NC replica. This attribute must have only one value. This NC root is called the default **domain** for the DC.

[msDS-hasMasterNCs](#): Contains the dsnames of the root objects of all writable NC replicas hosted by this DC. Not present on the [nTDSDSA](#) object of an RODC. On a normal (writable) DC, includes the default NC, config NC, schema NC, and all application NC replicas hosted by the DC.

[msDS-hasFullReplicaNCs](#): Contains the dsnames of the root objects of all read-only full NC replicas hosted by this DC. Not present on the [nTDSDSA](#) object of a normal (writable) DC. On an RODC, includes the default NC, config NC, schema NC, and all application NC replicas hosted by the DC.

[msDS-ReplicationEpoch](#): [MS-DRSR] section [4.1.3.1](#) (client behavior of IDL\_DRSSBind) and [MS-DRSR] section [4.1.10.5](#) (server behavior of IDL\_DRSSGetNCChanges) specify the effects of this attribute.

[msDS-DefaultNamingContext](#): In AD/LDS, specifies the NC that should be returned as the default NC by the [defaultNamingContext](#) attribute of the root DSE. If this attribute is not set, AD/LDS does not have a default NC and the [defaultNamingContext](#) attribute of the root DSE is treated by the server as if it does not exist. Not present in AD/DS.

[objectCategory](#): This attribute is a mandatory attribute representing the schema definition of [nTDSDSA](#) object. If the [objectCategory](#) points to the [classSchema](#) object for the [nTDSDSA](#) class then this [nTDSDSA](#) object is for a normal (writable) DC. If the [objectCategory](#) points to the [classSchema](#) object for the [nTDSDSARO](#) class then this [nTDSDSA](#) object is for an RODC.

#### 7.1.1.2.2.1.2.1.2 Connection Object

An [nTDSConnection](#) object represents a path for replication from a source DC to a destination DC. This object is a child of the [nTDSDSA](#) object of the destination DC. See section [7.2](#) for more information about connection objects.

Each [nTDSConnection](#) object has the following attributes:



extended information about a connection object that could be used by administrators, and is a bitfield consisting of one or more of the following bits.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
S	I	I	O	S	M	R	G	X	X	X	X	X	X	R	S	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
F	S	S	C	S	H		C							S	I																
		G												F																	

**X:** Unused. Must be zero and ignored.

**GC (NTDSCONN\_KCC\_GC\_TOPOLOGY, 0x00000001):** Not used.

**R (NTDSCONN\_KCC\_RING\_TOPOLOGY, 0x00000002):** The connection object is created to form a ring topology.

**MH (NTDSCONN\_KCC\_MINIMIZE\_HOPS\_TOPOLOGY, 0x00000004):** The connection object is created to minimize hops between replicating nodes.

**SS (NTDSCONN\_KCC\_STALE\_SERVERS\_TOPOLOGY, 0x00000008):** If KCC finds that the destination server is not responding, then it sets this bit.

**OC (NTDSCONN\_KCC\_OSCILLATING\_CONNECTION\_TOPOLOGY, 0x00000010):** KCC sets this bit if it is considering deleting the connection object. If the connection is considered for deletion for an implementation-specific time *T* (default value is 3) then KCC will delete the connection object. Otherwise, it will set this bit on the connection object.

**ISG (NTDSCONN\_KCC\_INTERSITE\_GC\_TOPOLOGY, 0x00000020):** This connection is to enable replication of partial NC replica between DCs in different sites.

**IS (NTDSCONN\_KCC\_INTERSITE\_TOPOLOGY, 0x00000040):** This connection is to enable replication of a full NC replica between DCs in different sites.

**SF (NTDSCONN\_KCC\_SERVER\_FAILOVER\_TOPOLOGY, 0x00000080):** This connection is a redundant connection between DCs used to failover when other connections between DCs are not functioning.

**SIF (NTDSCONN\_KCC\_SITE\_FAILOVER\_TOPOLOGY, 0x00000100):** This connection is a redundant connection between bridge head DCs in different DCs used to failover when other connections between bridge head DCs connecting two sites are not functioning.

**RS (NTDSCONN\_KCC\_REDUNDANT\_SERVER\_TOPOLOGY, 0x00000200):** Redundant connection object connecting bridge heads in different sites.

See section [7.2](#) for more information about these options.

#### 7.1.1.2.2.2 Subnets Container

Each forest contains a Subnets **container** in the config NC. A network subnet is a segment of a TCP/IP network to which a set of logical IP addresses is assigned. For each subnet in the forest, a [subnet](#) object exists in the Subnets **container**.

[name](#): Subnets

[parent](#): Sites **container**

[objectClass](#): [subnetContainer](#)

[systemFlags](#): FLAG\_DISALLOW\_DELETE

#### 7.1.1.2.2.1 Subnet Object

- [subnet](#) objects define network subnets in the directory. Subnets group computers in a way that identifies their physical proximity on the network. [subnet](#) objects are used to map computers to sites.
- [name](#): The name of the [subnet](#) object identifies the set of IP addresses that fall in this subnet. An IP address that falls in this subnet is considered to be in the site specified by the [siteObject](#) attribute of this object.

A valid subnet name must satisfy the following constraints:

Let *s* be the subnet name.

Let *l* be the length of the subnet name.

Let BitMask[] = {0x00000000, 0x00000080, 0x000000C0, 0x000000E0, 0x000000F0, 0x000000F8, 0x000000FC, 0x000000FE, 0x000000FF, 0x000080FF, 0x0000C0FF, 0x0000E0FF, 0x0000F0FF, 0x0000F8FF, 0x0000FCFF, 0x0000FEFF, 0x0000FFFF, 0x0080FFFF, 0x00C0FFFF, 0x00E0FFFF, 0x00F0FFFF, 0x00F8FFFF, 0x00FCFFFF, 0x00FEFFFF, 0x00FFFFFF, 0x80FFFFFF, 0xC0FFFFFF, 0xE0FFFFFF, 0xF0FFFFFF, 0xF8FFFFFF, 0xFCFFFFFF, 0xFEFFFFFF, 0xFFFFFFFF};

*s* is a valid subnet name if:

1. There is only one occurrence of the character '/' in *s*. Let *i* be the index of the character '/' in *s*.
2. The substring *s*[0, *i*-1] does not have any leading zeros and is either a valid IPv4 address in dotted decimal notation (as specified in [RFC1166](#)) or a valid IPv6 address in colon-hexadecimal form or compressed form (as specified in [RFC2373](#)).

Let *b* be the binary representation of the address.

3. The substring *s*[*i*+1, *l*-1] does not have any leading zeros and can be converted to an unsigned integer *n*.
4. When the address is in IPv4 format, 0 < *n* ≤ 32. When the address is in IPv6 format, 0 < *n* ≤ 128.
5. When the address is in IPv4 format, *b* & (~BitMask[*n*]) = 0.
6. When the address is in IPv4 format, *b* ≠ BitMask[*n*].

- [parent](#): Subnets **container**
- [objectClass](#): [subnet](#)
- [systemFlags](#): FLAG\_CONFIG\_ALLOW\_RENAME
- [siteObject](#): A reference to the dsname of the [site](#) object for the site that covers this subnet.

### 7.1.1.2.2.3 Inter-Site Transports Container

The Inter-Site Transports **container** provides the means for specifying the transport or wire protocol to be used for replication between sites. Inter-site replication can use either the **RPC** protocol over IP (see [\[MS-DRSR\]](#)), or the SMTP protocol (see [\[MS-SRPL\]](#)).

[name](#): Inter-Site Transports

[parent](#): Sites **container**

[objectClass](#): [interSiteTransportContainer](#)

[systemFlags](#): FLAG\_DISALLOW\_DELETE

#### 7.1.1.2.2.3.1 IP Transport Container

The IP transport **container** contains all the [siteLink](#) and [siteLinkBridge](#) objects that connect two or more sites for inter-site replication using **RPC** over IP protocol.

[parent](#): Inter-Site Transports **container**

[objectClass](#): [interSiteTransport](#)

[transportDLLName](#): This attribute identifies the component that implements the Inter-Site Messaging interface for this transport. For the IP transport, the component is "ismip.dll".

[transportAddressAttribute](#): Identifies which attribute on the [server](#) object of a DC should be used as the network address of the DC for replication using this transport. For the IP transport, the attribute is [DNSHostName](#).

[options](#): A set of the following bit flags presented in big-endian byte order. For IP transport the initial value is none present ([options](#) value 0x0).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	B	I
																													R	S	

**X**: Unused. Must be zero and ignored.

**IS (NTDSTRANSOPT\_OPT\_IGNORE\_SCHEDULES)**: If present, values of the [schedule](#) attribute for [siteLink](#) objects associated with this transport are ignored. In this case the schedule is assumed to be always "on", that is, that the transport is always available to send and receive messages.

**BR (NTDSTRANSOPT\_OPT\_BRIDGES\_REQUIRED)**: If present, transitive connectivity between [siteLink](#) objects associated with this transport is assumed only if the [siteLink](#) objects are in the [siteLinkList](#) of the same [siteLinkBridge](#) object. If absent, the system behaves as if all [siteLink](#) objects associated with this transport were in the [siteLinkList](#) of a common [siteLinkBridge](#) object associated with this transport.

#### 7.1.1.2.2.3.2 SMTP Transport Container

In AD/DS, the SMTP transport **container** contains all the [siteLink](#) and [siteLinkBridge](#) objects that connect two or more sites for inter-site replication using the SMTP protocol. Not present in AD/LDS.

[parent](#): Inter-Site Transports **container**

[objectClass](#): [interSiteTransport](#)

[transportDLLName](#): This attribute identifies the component that implements the Inter-Site Messaging interface for this transport. For the SMTP transport, the component is "ismsmtp.dll".

[transportAddressAttribute](#): Identifies which attribute on the [server](#) object of a DC should be used as the network address of the DC for replication using this transport. For the SMTP transport, the attribute is [mailAddress](#).

[options](#): A set of bit flags as defined for [options](#) in section [7.1.1.2.2.3.1](#). For SMTP transport the initial value is **NTDSTRANSPORT\_OPT\_IGNORE\_SCHEDULES** present ([options](#) value 0x1).

### 7.1.1.2.2.3.3 Site Link Object

In order for a DC in one site to replicate directly with a DC in a different site, a [siteLink](#) object, or a series of [siteLink](#) objects, must connect the two sites specified. A [siteLink](#) object identifies the transport (wire protocol) to be used for replication between the sites. If the transport is IP, the [siteLink](#) object is child of the IP Transport **container**. If the transport is SMTP, the [siteLink](#) object is a child of the SMTP Transport **container**. Any single [siteLink](#) object may encompass two or more sites. If a [siteLink](#) object contains two sites, then those two sites are considered to be directly connected. If a [siteLink](#) object contains more than two sites, then all of the sites listed in the [siteLink](#) are considered to be connected in a mesh of point-to-point links.

[parent](#): Either IP Transport **container** or SMTP Transport **container**.

[objectClass](#): [siteLink](#)

[systemFlags](#): FLAG\_CONFIG\_ALLOW\_RENAME

[cost](#): An administrator-defined cost value associated with that replication path.

[siteList](#): Contains the dsname of the [site](#) objects for the sites that are connected using this site link.

[replInterval](#): An interval that determines how frequently replication occurs over this site link during the times when the schedule allows replication.

[schedule](#): Replication schedule of type SCHEDULE that specifies the time intervals when replication is permitted between the two sites.

[options](#): A set of the following bit flags presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	D	T	U
																												C	S	N	

**X**: Unused. Must be zero and ignored.

**UN (NTDSSITELINK\_OPT\_USE\_NOTIFY)**: If present, enables replications notifications (see Updates, section [3.1.1.5](#)) between DCs in different sites in the [siteList](#).

**TS (NTDSSITELINK\_OPT\_TWOWAY\_SYNC)**: If present, forces a replication cycle in the opposite direction at the end of a replication cycle between DCs in different sites in the [siteList](#).

**DC (NTDSSITELINK\_OPT\_DISABLE\_COMPRESSION)**: If present, disables compression of IDL\_DRSGetNCChanges response messages sent between DCs in different sites in the [siteList](#).

#### 7.1.1.2.2.3.4 Site Link Bridge Object

A [siteLinkBridge](#) object connects two or more [siteLink](#) objects associated with the same transport. The [siteLinkBridge](#) object is a child of the [interSiteTransport](#) object for the transport used by the [siteLink](#) objects being connected.

When **NTDSTRANSPORT\_OPT\_BRIDGES\_REQUIRED** is present in the [options](#) of a transport, replication only assumes transitive communication between sites as specified in the [siteLinkBridge](#) objects for that transport. See the specification of IDL\_QuerySitesByCost [MS-DRSR] section [4.1.16.3](#).

[parent](#): Either IP Transport **container** or SMTP Transport **container**.

[objectClass](#): [siteLinkBridge](#)

[systemFlags](#): FLAG\_CONFIG\_ALLOW\_RENAME

[siteLinkList](#): Contains the dsnames of the [siteLink](#) objects for the site links being connected by this site link bridge.

#### 7.1.1.2.3 Display Specifiers Container

The display specifier objects are installed in the directory for use by the administrative applications of the directory. Each supported locale (i.e., language and location) for the administrative application is assigned a number, called a Locale ID (LCID), and each of the children of the Display Specifier **container** is named with that number's hexadecimal character representation (ex 1033 is name '409'). Section [2.2.1](#) contains a table associating each locale with an LCID. Some locales do not have Display Specifier Objects installed by default.

[name](#): DisplaySpecifiers

[parent](#): Config NC root

[objectClass](#): [container](#)

[systemFlags](#): FLAG\_DISALLOW\_DELETE

##### 7.1.1.2.3.1 Display Specifier Object

[name](#): The name of each Display Specifier is a hexadecimal number in **Unicode** characters that represents a locale.

[parent](#): Display Specifiers **container**

[objectClass](#): [displaySpecifier](#)

The children of the Display Specifier object describe an implementation component of the administrative application. These objects are not interpreted by the DC.

The attributes of the children of the Display Specifier object are:

[parent](#): Display Specifier object

[objectClass](#): [displaySpecifier](#)

[adminPropertyPages](#): Each administrative application component for an object of class [displaySpecifier](#) associates a **Component Object Model** (COM) object represented by **UUID** called a property page in this attribute. Each value in this multivalued attribute describes a single COM object. The description of a COM object is as a string with the following format:

<order-number>,<UUID>,[optional data]

where

- The order-number determines a desired ordering in the application for each COM object represented in the value.
- The **UUID** is a string representation of a **UUID** representing a COM object enclosed in curly braces.
- The optional data is passed to the COM object by the implementation.

[shellPropertyPages](#): This attribute has the same semantics as [adminPropertyPages](#).

[adminContextMenu](#): This attribute can store values, where each value describes either a single COM object representation or a single application representation. For a COM object, this attribute has the same semantics as [adminPropertyPage](#). For an application representation, the description is stored as a string with the following format:

<order-number>,<context menu name>,<program name>

where

- The order-number determines a desired ordering for each COM object represented in the value.
- The context menu name is the text of the menu item for the administrative application interface.
- The program name is the application that is executed when the application references this [adminContextMenu](#) attribute. Either the full path must be specified or the application must be in the search path.

[shellContextMenu](#): This attribute has the same semantics as [adminContextMenu](#).

[adminMultiselectPropertyPages](#): This attribute has the same semantics as [adminPropertyPages](#).

[treatAsLeaf](#): This attribute is a Boolean that instructs the administrative application to ignore any child objects of this object, whether they exist or not.

[creationWizard](#): The [creationWizard](#) attribute identifies primary object creation COM objects to replace the existing or native object creation wizard in Active Directory administrative applications. The COM objects in this value are represented by **UUID**.

[createWizardExt](#): The [createWizardExt](#) attribute identifies secondary object creation COM objects for the administrative applications, if needed. This attribute is multi-valued and requires the following format:

<order number>,<UUID>

where

- The order-number determines a desired ordering for each COM object represented in the value.
- The **UUID** is a string representation of a **UUID** representing a COM object.



[iconPath](#): The [iconPath](#) attribute can be specified in one of two ways:

1. "<state>,<icon file name>" or
2. "<state>,<module file name>,<resource ID>"

In these examples, the "<state>" is an integer with a value between 0 and 15. The value 0 is defined to be the default or closed state of the icon. The value 1 is defined to be the open state of the icon. The value 2 is the disabled state. All other values are application-defined.

The "<icon file name>" is the path and file name of an icon file that contains the icon image.

The "<module file name>" is the path and file name of a module, such as an EXE or DLL, that contains the icon image in a resource. The "<resource ID>" is an integer that specifies the resource identifier of the icon resource within the module.

#### 7.1.1.2.4 Services

[name](#): Services

[parent](#): Config NC root object

[objectClass](#): [container](#)

[systemFlags](#): { FLAG\_DISALLOW\_DELETE }

##### 7.1.1.2.4.1 Windows NT

[name](#): Windows NT

[parent](#): Services

[objectClass](#): [container](#)

##### 7.1.1.2.4.1.1 Directory Service

[name](#): Directory Service

[parent](#): Windows NT (section [7.1.1.2.4.1](#))

[objectClass](#): [nTDSservice](#)

[tombstoneLifetime](#): The number of days a deleted object exists before it is garbage collected. See [3.1.1](#) for more information.

[sPNMappings](#): In AD/DS, a set of SPN mappings as specified in [MS-DRSR] section [4.1.4.2.19](#) (MapSPN). Not present in AD/LDS.

[msDS-Other-Settings](#): A multi-valued string where each string value encodes a name-value pair. In the encoding the name and value are separated by an "=". For example, the encoding of the name "DisableVLVSupport" with value "0" is "DisableVLVSupport=0". Each name is the name of an **LDAP** configurable setting, and the value is the value of that setting. The **LDAP** configurable settings and their effects are specified in section [3.1.1.3.4.7](#).

[dSHeuristics](#): See section [7.1.1.2.4.1.2](#).

### 7.1.1.2.4.1.2 dSHeuristics

dSHeuristics is a **Unicode** string attribute with the following meaning assigned to each character. Unless otherwise specified, the default value of each character is '0'; for instance, if dSHeuristics is not present or has length zero, then fSupFirstLastANR=false.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
fSupFirstLastANR																fSupLastFirstANR															
fDoListObject																fDoNickRes															
fLDAPUsePermMod																ulHideDSID															
fLDAPBlockAnonOps																fAllowAnonNSPI															
fUserPwdSupport																tenthChar															
fSpecifyGUIDOnAdd																fDontStandardizeSDs															
fAllowPasswordOperationsOverNonSecureConnection																fDontPropagateOnNoChangeUpdate															
fComputeANRStats																dwAdminSDExMask															
fKVNOEmuW2K																fLDAPBypassUpperBoundsOnLimits															

**fSupFirstLastANR:** If this field equals character '0', then fSupFirstLastANR=false, else fSupFirstLastANR=true.

Section [3.1.1.3.1.3.3](#) specifies the effects of this heuristic.

**fSupLastFirstANR:** If this field equals character '0', then fSupLastFirstANR=false, else fSupLastFirstANR=true.

Section [3.1.1.3.1.3.3](#) specifies the effects of this heuristic.

**fDoListObject:** If this field equals character '1', then fDoListObject=true, else fDoListObject=false.

Section [5.1.3.2](#) specifies the effects of this heuristic.

**fDoNickRes:** If this field equals character '0' then fDoNickRes=false, else fDoNickRes=true.

If fDoNickRes is true, then the MAPI ResolveNames call attempts an exact match against the MAPI nickname attribute (the attribute with [mAPIID](#) equal to 0x3A00) before performing an ANR search. If there is a single entry whose MAPI nickname attribute exactly matches the requested name, ResolveNames returns that single entry; otherwise, ResolveNames performs a normal ANR search and returns all entries resulting from that search.

**fLDAPUsePermMod:** If this field equals character '0', then fLDAPUsePerMod=false, else fLDAPUsePerMod=true.

If fLDAPUsePerMod is true, then all **LDAP** Modify operations behave as if the LDAP\_SERVER\_PERMISSIVE\_MODIFY\_OID control was passed. Section [3.1.1.3.4.1.8](#) specifies the effects of the LDAP\_SERVER\_PERMISSIVE\_MODIFY\_OID control.

**ulHideDSID:** ulHideDSID is the numeric value of the character in the field: 0 = '0', 1 = '1', etc.

Controls when DSIDs are returned in the **LDAP** extended error string when an operation encounters an error. If this value is character '0', then DSIDs will be returned at all times. If this value is character '1', then DSIDs will be returned as long as the error is not a name error where different DSIDs may reveal the existence of an object that is not visible to the client. If this value is anything but '0' or '1', then DSIDs will not be returned at all.

A DSID consists of the string "DSID-" followed by an implementation-specific 32-bit integer expressed in hexadecimal. The integer identifies the execution point at which an error occurred.

**fLDAPBlockAnonOps:** If this field equals character '2', then fLDAPBlockAnonOps=false, else fLDAPBlockAnonOps=true. If this field is not present, it defaults to '2' when the DC functional level is less than DS\_BEHAVIOR\_WIN2003, and to '0' otherwise.

Section [5.1.3](#) specifies the effects of this heuristic.

**fAllowAnonNSPI:** If this field equals character '0', then fAllowAnonNSPI=false, else fAllowAnonNSPI=true.

If fAllowAnonNSPI is true, allow anonymous calls to the NSPI **RPC** bind method. Otherwise, only allow authenticated clients.

**fUserPwdSupport:** If this field equals the character '0', fUserPwdSupport=false for AD/DS, and fUserPwdSupport=true for AD/LDS. If this field equals character '2', then fUserPwdSupport=false, else fUserPwdSupport=true.

Sections [3.1.1.3.1.5.2](#) and [3.1.1.4.4](#) specify the effects of this heuristic.

**tenthChar:** When setting [dSHeuristics](#) to a value 10 or more **Unicode** characters long, if the value of tenthChar is not '1', the server rejects the update. See section [3.1.1.5.3.2](#).

**fSpecifyGUIDOnAdd:** If this field equals character '0', then fSpecifyGUIDOnAdd=false, else fSpecifyGUIDOnAdd=true.

The fSpecifyGUIDOnAdd heuristic applies only to AD/DS. AD/LDS always treats this heuristic as if the field equals character '0', that is, fSpecifyGUIDOnAdd = false.

Section [3.1.1.5.2.2](#) specifies the effects of this heuristic.

**fDontStandardizeSDs:** If this field equals character '0', then fDontStandardizeSDs=false, else fDontStandardizeSDs=true.

Section [7.1.3](#) specifies the effects of this heuristic.

**fAllowPasswordOperationsOverNonSecureConnection:** If this field equals character '0', then fAllowPasswordOperationsOverNonSecureConnection=false, else fAllowPasswordOperationsOverNonSecureConnection=true.

The fAllowPasswordOperationsOverNonSecureConnection heuristic applies only to AD/LDS.

Sections [3.1.1.5.2.2](#) and [3.1.1.5.3.2](#) specify the effects of this heuristic.

**fDontPropagateOnNoChangeUpdate:** If this field equals character '0', then fDontPropagateOnNoChangeUpdate=false, else fDontPropagateOnNoChangeUpdate=true.

If fDontPropagateOnNoChangeUpdate is true, when the [nTSecurityDescriptor](#) attribute of an object is set to a value that is bitwise identical to the current value, no work item is enqueued for the task that updates the security descriptors on the children of a modified object in order to propagate inherited ACEs (section [7.1.3](#)). If fDontPropagateOnNoChangeUpdate is false, a work item is always enqueued when the [nTSecurityDescriptor](#) attribute is modified.

The fDontPropagateOnNoChangeUpdate heuristic applies to Windows Server 2008. Previous versions of Active Directory behave as if fDontPropagateOnNoChangeUpdate is false.

**fComputeANRStats:** If this field equals character '0' then fComputeANRStats=false, else fComputeANRStats=true.

The effects of the fComputeANRStats field are outside the state model. If fComputeANRStats is true, ANR searches (section [3.1.1.3.1.3.3](#)) are optimized using cardinality estimates like all other searches.

**dwAdminSDExMask:** The valid values of this field are characters from the set '0'-'9', and 'a'-'f'. dwAdminSDExMask equals the character, interpreted as a hex digit and converted into a 4-bit value (i.e., '1'=0x1, 'f'=0xF).

Section [3.1.1.6.1](#) specifies the effects of this heuristic.

**fKVNOEmuW2K:** If this field equals character '0', then fKVNOEmuW2K=false, else fKVNOEmuW2K=true.

Section [3.1.1.4.5.16](#) specifies the effects of this heuristic.

**fLDAPBypassUpperBoundsOnLimits:** If this field equals character '0', then fLDAPBypassUpperBoundsOnLimits=false, else fLDAPBypassUpperBoundsOnLimits=true.

If fLDAPBypassUpperBoundsOnLimits is false, DCs impose implementation-dependent limits when interpreting values of the **LDAP** policies specified in section [3.1.1.3.4.6](#). If the configured policy value exceeds the limit, the DC ignores the policy value and instead uses the implementation-dependent limit.

This heuristic applies to Windows Server 2008. Previous versions of Active Directory do not impose any such limits.

### 7.1.1.2.4.1.3 Query-Policies

**Container** to store the default [queryPolicy](#) object. Can also contain [queryPolicy](#) objects created by administrators. See section [3.1.1.3.4.6](#) for the effects of [queryPolicy](#) objects.

[name](#): Query-Policies

[parent](#): Directory Service

[objectClass](#): [container](#)

#### 7.1.1.2.4.1.3.1 Default Query Policy

Stores the default **LDAP** query policies. See section [3.1.1.3.4.6](#) for the effects of the default [queryPolicy](#) object.

[name](#): Default Query Policy

[parent](#): Query-Policies

[objectClass](#): [queryPolicy](#)

[LDAPAdminLimits](#): encoding of the **LDAP** policies as specified in section [3.1.1.3.4.6](#).

#### 7.1.1.2.4.1.4 SCP Publication Service Object

This object is present only in AD/LDS. It is system created but can be removed and re-created by the administrator if desired. This object stores forest-wide configuration used to control the creation of [serviceConnectionPoint](#) objects by an AD/LDS DC running on a computer joined to a AD/DS **domain**. Section [7.3.8](#) specifies the effects of this object.

[name](#): SCP Publication Service

[parent](#): Directory Service

[objectClass](#): [msDS-ServiceConnectionPointPublicationService](#)

[Enabled](#): A Boolean value. If false, no DC in this forest will create a [serviceConnectionPoint](#) object.

[msDS-DisableForInstances](#): A set of references to [nTDSDSA](#) objects in this forest. A DC in this set will not create a [serviceConnectionPoint](#) object.

[msDS-SCPContainer](#): If present, is a reference to a AD/DS object (a reference to an object outside this AD/LDS forest). The parent of any [serviceConnectionPoint](#) object created by a DC in this forest is [msDS-SCPContainer](#). If an AD/LDS DC in this forest is joined to **domain** D, then a DC of **domain** D must be capable of generating a referral to a DC containing a writable replica of the NC containing [msDS-SCPContainer](#).

[keywords](#): A set of strings. The [keywords](#) attribute of any [serviceConnectionPoint](#) object created by a DC in this forest contains all of these strings.

#### 7.1.1.2.5 Physical Locations

This object is not present on AD/LDS.

[name](#): Physical Locations

[parent](#): config NC root object

[objectClass](#): [physicalLocation](#)

#### 7.1.1.2.6 WellKnown Security Principals

This object is not present on AD/LDS.

[name](#): WellKnown Security Principals

[parent](#): config NC root object

[objectClass](#): [container](#)

[systemFlags](#): { FLAG\_DISALLOW\_DELETE}

#### **7.1.1.2.6.1 Anonymous Logon**

[name](#): Anonymous Logon

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-7

#### **7.1.1.2.6.2 Authenticated Users**

[name](#): Authenticated Users

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-11

#### **7.1.1.2.6.3 Batch**

[name](#): Batch

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-3

#### **7.1.1.2.6.4 Creator Group**

[name](#): Creator Group

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-3-1

#### **7.1.1.2.6.5 Creator Owner**

[name](#): Creator Owner

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-3-0

#### **7.1.1.2.6.6 Dialup**

[name](#): Dialup

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-1

#### **7.1.1.2.6.7 Digest Authentication**

[name](#): Digest Authentication

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-64-21

#### **7.1.1.2.6.8 Enterprise Domain Controllers**

[name](#): Enterprise Domain Controllers

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-9

By default all normal (writable) DCs in the forest belong to this group.

#### **7.1.1.2.6.9 Everyone**

[name](#): Everyone

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-1-0

#### **7.1.1.2.6.10 Interactive**

[name](#): Interactive

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-4

#### **7.1.1.2.6.11 IUSR**

[name](#): IUSR

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-17

#### **7.1.1.2.6.12 Local Service**

[name](#): Local Service

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-19

#### **7.1.1.2.6.13 Network**

[name](#): Network

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-2

#### **7.1.1.2.6.14 Network Service**

[name](#): Network Service

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-20

#### **7.1.1.2.6.15 NTLM Authentication**

[name](#): NTLM Authentication

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-64-10

#### **7.1.1.2.6.16 Other Organization**

[name](#): Other Organization

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-1000

#### **7.1.1.2.6.17 Owner Rights**

[name](#): Owner Rights

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-3-4

#### **7.1.1.2.6.18 Proxy**

[name](#): Proxy

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-8

#### **7.1.1.2.6.19 Remote Interactive Logon**

[name](#): Remote Interactive Logon

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-14

#### **7.1.1.2.6.20 Restricted**

[name](#): Restricted

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-12

#### **7.1.1.2.6.21 SChannel Authentication**

[name](#): SChannel Authentication

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-64-14



#### **7.1.1.2.6.22 Self**

[name](#): Self

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-10

#### **7.1.1.2.6.23 Service**

[name](#): Service

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-6

#### **7.1.1.2.6.24 System**

[name](#): System

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-18

#### **7.1.1.2.6.25 Terminal Server User**

[name](#): Terminal Server User

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-13

#### **7.1.1.2.6.26 This Organization**

[name](#): This Organization

[parent](#): WellKnown Security Principals

[objectSid](#): S-1-5-15

#### **7.1.1.2.7 Extended Rights**

[name](#): Extended Rights

[parent](#): Config NC root object

[objectClass](#): [container](#)

[systemFlags](#): { FLAG\_DISALLOW\_DELETE }

#### **7.1.1.2.7.1 controlAccessRight objects**

All [controlAccessRight](#) objects have:

[objectClass](#): [controlAccessRight](#)

[rightsGuid](#): This value is the identifier of the **control access right** used for security descriptors and SDDL.

[appliesTo](#): Each value in this attribute is a **GUID**, with each **GUID** equaling an attribute [schemaIDGUID](#) on a schema object defining a class in the schema NC. This class defines the objects that the **control access right** can be in a security descriptor for.

[localizationDisplayId](#): This is implementation-specific information for the administrative application.

[validAccesses](#): This is implementation-specific information for the administrative application.

#### **7.1.1.2.7.2 Change-Rid-Master**

[name](#): Change-Rid-Master

[rightsGuid](#): d58d5f36-0a98-11d1-adbb-00c04fd8d5cd

[appliesTo](#): 6617188d-8f3c-11d0-afda-00c04fd930c9

#### **7.1.1.2.7.3 Do-Garbage-Collection**

[name](#): Do-Garbage-Collection

[rightsGuid](#): fec364e0-0a98-11d1-adbb-00c04fd8d5cd

[appliesTo](#): f0f8ffab-1191-11d0-a060-00aa006c33ed

#### **7.1.1.2.7.4 Recalculate-Hierarchy**

[name](#): Recalculate-Hierarchy

[rightsGuid](#): 0bc1554e-0a99-11d1-adbb-00c04fd8d5cd

[appliesTo](#): f0f8ffab-1191-11d0-a060-00aa006c33ed

#### **7.1.1.2.7.5 Allocate-Rids**

[name](#): Allocate-Rids

[rightsGuid](#): 1abd7cf8-0a99-11d1-adbb-00c04fd8d5cd

[appliesTo](#): f0f8ffab-1191-11d0-a060-00aa006c33ed

#### **7.1.1.2.7.6 Change-PDC**

[name](#): Change-PDC

[rightsGuid](#): bae50096-4752-11d1-9052-00c04fc2d4cf

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.7 Add-GUID**

[name](#): Add-GUID

[rightsGuid](#): 440820ad-65b4-11d1-a3da-0000f875ae0d

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.8 Change-Domain-Master**

[name](#): Change-Domain-Master

[rightsGuid](#): 014bf69c-7b3b-11d1-85f6-08002be74fab

[appliesTo](#): ef9e60e0-56f7-11d1-a9c6-0000f80367c1

#### **7.1.1.2.7.9 Public-Information**

[name](#): Public-Information

[rightsGuid](#): e48d0154-bcf8-11d1-8702-00c04fb96050

[appliesTo](#): 4828CC14-1437-45bc-9B07-AD6F015E5F28, bf967a86-0de6-11d0-a285-00aa003049e2, and bf967aba-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.10 msmq-Receive-Dead-Letter**

[name](#): msmq-Receive-Dead-Letter

[rightsGuid](#): 4b6e08c0-df3c-11d1-9c86-006008764d0e

[appliesTo](#): 9a0dc344-c100-11d1-bbc5-0080c76670c0

#### **7.1.1.2.7.11 msmq-Peek-Dead-Letter**

[name](#): msmq-Peek-Dead-Letter

[rightsGuid](#): 4b6e08c1-df3c-11d1-9c86-006008764d0e

[appliesTo](#): 9a0dc344-c100-11d1-bbc5-0080c76670c0

#### **7.1.1.2.7.12 msmq-Receive-computer-Journal**

[name](#): msmq-Receive-computer-Journal

[rightsGuid](#): 4b6e08c2-df3c-11d1-9c86-006008764d0e

[appliesTo](#): 9a0dc344-c100-11d1-bbc5-0080c76670c0

#### **7.1.1.2.7.13 msmq-Peek-computer-Journal**

[name](#): msmq-Peek-computer-Journal

[rightsGuid](#): 4b6e08c3-df3c-11d1-9c86-006008764d0e

[appliesTo](#): 9a0dc344-c100-11d1-bbc5-0080c76670c0

#### **7.1.1.2.7.14 msmq-Receive**

[name](#): msmq-Receive

[rightsGuid](#): 06bd3200-df3e-11d1-9c86-006008764d0e

[appliesTo](#): 9a0dc343-c100-11d1-bbc5-0080c76670c0

#### **7.1.1.2.7.15 msmq-Peek**

[name](#): msmq-Peek

[rightsGuid](#): 06bd3201-df3e-11d1-9c86-006008764d0e

[appliesTo](#): 9a0dc343-c100-11d1-bbc5-0080c76670c0

#### **7.1.1.2.7.16 msmq-Send**

[name](#): msmq-Send

[rightsGuid](#): 06bd3202-df3e-11d1-9c86-006008764d0e

[appliesTo](#): 46b27aac-aafa-4ffb-b773-e5bf621ee87b and 9a0dc343-c100-11d1-bbc5-0080c76670c0

#### **7.1.1.2.7.17 msmq-Receive-journal**

[name](#): msmq-Receive-journal

[rightsGuid](#): 06bd3203-df3e-11d1-9c86-006008764d0e

[appliesTo](#): 9a0dc343-c100-11d1-bbc5-0080c76670c0

#### **7.1.1.2.7.18 msmq-Open-Connector**

[name](#): msmq-Open-Connector

[rightsGuid](#): b4e60130-df3f-11d1-9c86-006008764d0e

[appliesTo](#): bf967ab3-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.19 Apply-Group-Policy**

[name](#): Apply-Group-Policy

[rightsGuid](#): edacfd8f-ffb3-11d1-b41d-00a0c968f939

[appliesTo](#): f30e3bc2-9ff0-11d1-b603-0000f80367c1

#### **7.1.1.2.7.20 RAS-Information**

[name](#): RAS-Information

[rightsGuid](#): 037088f8-0ae1-11d2-b422-00a0c968f939

[appliesTo](#): 4828CC14-1437-45bc-9B07-AD6F015E5F28 and bf967aba-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.21 DS-Install-Replica**

[name](#): DS-Install-Replica

[rightsGuid](#): 9923a32a-3607-11d2-b9be-0000f87a36b2

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.22 Change-Infrastructure-Master**

[name](#): Change-Infrastructure-Master

[rightsGuid](#): cc17b1fb-33d9-11d2-97d4-00c04fd8d5cd

[appliesTo](#): 2df90d89-009f-11d2-aa4c-00c04fd7d83a

#### **7.1.1.2.7.23 Update-Schema-Cache**

[name](#): Update-Schema-Cache

[rightsGuid](#): be2bb760-7f46-11d2-b9ad-00c04f79f805

[appliesTo](#): bf967a8f-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.24 Recalculate-Security-Inheritance**

[name](#): Recalculate-Security-Inheritance

[rightsGuid](#): 62dd28a8-7f46-11d2-b9ad-00c04f79f805

[appliesTo](#): f0f8ffab-1191-11d0-a060-00aa006c33ed

#### **7.1.1.2.7.25 DS-Check-Stale-Phantoms**

[name](#): DS-Check-Stale-Phantoms

[rightsGuid](#): 69ae6200-7f46-11d2-b9ad-00c04f79f805

[appliesTo](#): f0f8ffab-1191-11d0-a060-00aa006c33ed

#### **7.1.1.2.7.26 Certificate-Enrollment**

[name](#): Certificate-Enrollment

[rightsGuid](#): 0e10c968-78fb-11d2-90d4-00c04f79dc55

[appliesTo](#): e5209ca2-3bba-11d2-90cc-00c04fd91ab1

#### **7.1.1.2.7.27 Self-Membership**

[name](#): Self-Membership

[rightsGuid](#): bf9679c0-0de6-11d0-a285-00aa003049e2

[appliesTo](#): bf967a9c-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.28 Validated-DNS-Host-Name**

[name](#): Validated-DNS-Host-Name

[rightsGuid](#): 72e39547-7b18-11d1-adeb-00c04fd8d5cd

[appliesTo](#): bf967a86-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.29 Validated-SPN**

[name](#): Validated-SPN

[rightsGuid](#): f3a64788-5306-11d1-a9c5-0000f80367c1

[appliesTo](#): bf967a86-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.30 Generate-RSoP-Planning**

[name](#): Generate-RSoP-Planning

[rightsGuid](#): b7b1b3dd-ab09-4242-9e30-9980e5d322f7

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9 and bf967aa5-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.31 Refresh-Group-Cache**

[name](#): Refresh-Group-Cache

[rightsGuid](#): 9432c620-033c-4db7-8b58-14ef6d0bf477

[appliesTo](#): f0f8ffab-1191-11d0-a060-00aa006c33ed

#### **7.1.1.2.7.32 Reload-SSL-Certificate**

[name](#): Reload-SSL-Certificate

[rightsGuid](#): 1a60ea8d-58a6-4b20-bcdc-fb71eb8a9ff8

[appliesTo](#): f0f8ffab-1191-11d0-a060-00aa006c33ed

#### **7.1.1.2.7.33 SAM-Enumerate-Entire-Domain**

[name](#): SAM-Enumerate-Entire-Domain

[rightsGuid](#): 91d67418-0135-4acc-8d79-c08e857cfbec

[appliesTo](#): bf967aad-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.34 Generate-RSoP-Logging**

[name](#): Generate-RSoP-Logging

[rightsGuid](#): b7b1b3de-ab09-4242-9e30-9980e5d322f7

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9 and bf967aa5-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.35 Domain-Other-Parameters**

[name](#): Domain-Other-Parameters

[rightsGuid](#): b8119fd0-04f6-4762-ab7a-4986c76b3f9a

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.36 DNS-Host-Name-Attributes**

[name](#): DNS-Host-Name-Attributes

[rightsGuid](#): 72e39547-7b18-11d1-adeb-00c04fd8d5cd

[appliesTo](#): bf967a86-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.37 Create-Inbound-Forest-Trust**

[name](#): Create-Inbound-Forest-Trust

[rightsGuid](#): e2a36dc9-ae17-47c3-b58b-be34c55ba633

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.38 DS-Replication-Get-Changes-All**

[name](#): DS-Replication-Get-Changes-All

[rightsGuid](#): 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2

[appliesTo](#): bf967a8f-0de6-11d0-a285-00aa003049e2, bf967a87-0de6-11d0-a285-00aa003049e2, and 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.39 Migrate-SID-History**

[name](#): Migrate-SID-History

[rightsGuid](#): BA33815A-4F93-4c76-87F3-57574BFF8109

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.40 Reanimate-Tombstones**

[name](#): Reanimate-Tombstones

[rightsGuid](#): 45EC5156-DB7E-47bb-B53F-DBEB2D03C40F

[appliesTo](#): bf967a8f-0de6-11d0-a285-00aa003049e2, bf967a87-0de6-11d0-a285-00aa003049e2, and 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.41 Allowed-To-Authenticate**

[name](#): Allowed-To-Authenticate

[rightsGuid](#): 68B1D179-0D15-4d4f-AB71-46152E79A7BC

[appliesTo](#): 4828cc14-1437-45bc-9b07-ad6f015e5f28, bf967aba-0de6-11d0-a285-00aa003049e2, and bf967a86-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.42 DS-Execute-Intentions-Script**

[name](#): DS-Execute-Intentions-Script

[rightsGuid](#): 2f16c4a5-b98e-432c-952a-cb388ba33f2e

[appliesTo](#): ef9e60e0-56f7-11d1-a9c6-0000f80367c1

#### **7.1.1.2.7.43 DS-Replication-Monitor-Topology**

[name](#): DS-Replication-Monitor-Topology

[rightsGuid](#): f98340fb-7c5b-4cdb-a00b-2ebdfa115a96

[appliesTo](#): bf967a8f-0de6-11d0-a285-00aa003049e2, bf967a87-0de6-11d0-a285-00aa003049e2, and 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.44 Update-Password-Not-Required-Bit**

[name](#): Update-Password-Not-Required-Bit

[rightsGuid](#): 280f369c-67c7-438e-ae98-1d46f3c6f541

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.45 Unexpire-Password**

[name](#): Unexpire-Password

[rightsGuid](#): ccc2dc7d-a6ad-4a7a-8846-c04e3cc53501

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.46 Enable-Per-User-Reversibly-Encrypted-Password**

[name](#): Enable-Per-User-Reversibly-Encrypted-Password

[rightsGuid](#): 05c74c5e-4deb-43b4-bd9f-86664c2a7fd5

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.47 DS-Query-Self-Quota**

[name](#): DS-Query-Self-Quota

[rightsGuid](#): 4ecc03fe-ffc0-4947-b630-eb672a8a9dbc

[appliesTo](#): da83fc4f-076f-4aea-b4dc-8f4dab9b5993

#### **7.1.1.2.7.48 Private-Information**

[name](#): Private-Information

[rightsGuid](#): 91e647de-d96f-4b70-9557-d63ff4f3ccd8

[appliesTo](#): bf967aba-0de6-11d0-a285-00aa003049e2 and 4828cc14-1437-45bc-9b07-ad6f015e5f28

#### **7.1.1.2.7.49 MS-TS-GatewayAccess**

[name](#): MS-TS-GatewayAccess

[rightsGuid](#): ffa6f046-ca4b-4feb-b40d-04dfef722543



[appliesTo](#): bf967a86-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.50 Terminal-Server-License-Server**

[name](#): Terminal-Server-License-Server

[rightsGuid](#): 5805bc62-bdc9-4428-a5e2-856a0f4c185e

[appliesTo](#): bf967aba-0de6-11d0-a285-00aa003049e2 and 4828cc14-1437-45bc-9b07-ad6f015e5f28

#### **7.1.1.2.7.51 Domain-Administer-Server**

[name](#): Domain-Administer-Server

[rightsGuid](#): ab721a52-1e2f-11d0-9819-00aa0040529b

[appliesTo](#): bf967aad-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.52 User-Change-Password**

[name](#): User-Change-Password

[rightsGuid](#): ab721a53-1e2f-11d0-9819-00aa0040529b

[appliesTo](#): 4828CC14-1437-45bc-9B07-AD6F015E5F28, bf967a86-0de6-11d0-a285-00aa003049e2, and bf967aba-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.53 User-Force-Change-Password**

[name](#): User-Force-Change-Password

[rightsGuid](#): 00299570-246d-11d0-a768-00aa006e0529

[appliesTo](#): 4828CC14-1437-45bc-9B07-AD6F015E5F28, bf967a86-0de6-11d0-a285-00aa003049e2, and bf967aba-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.54 Send-As**

[name](#): Send-As

[rightsGuid](#): ab721a54-1e2f-11d0-9819-00aa0040529b

[appliesTo](#): 4828CC14-1437-45bc-9B07-AD6F015E5F28, bf967a86-0de6-11d0-a285-00aa003049e2, and bf967aba-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.55 Receive-As**

[name](#): Receive-As

[rightsGuid](#): ab721a56-1e2f-11d0-9819-00aa0040529b

[appliesTo](#): 4828CC14-1437-45bc-9B07-AD6F015E5F28, bf967a86-0de6-11d0-a285-00aa003049e2, and bf967aba-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.56 Send-To**

[name](#): Send-To

[rightsGuid](#): ab721a55-1e2f-11d0-9819-00aa0040529b

[appliesTo](#): bf967a9c-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.57 Domain-Password**

[name](#): Domain-Password

[rightsGuid](#): c7407360-20bf-11d0-a768-00aa006e0529

[appliesTo](#): 19195a5b-6da0-11d0-afd3-00c04fd930c9 and 19195a5a-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.58 General-Information**

[name](#): General-Information

[rightsGuid](#): 59ba2f42-79a2-11d0-9020-00c04fc2d3cf

[appliesTo](#): 4828CC14-1437-45bc-9B07-AD6F015E5F28 and bf967aba-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.59 User-Account-Restrictions**

[name](#): User-Account-Restrictions

[rightsGuid](#): 4c164200-20c0-11d0-a768-00aa006e0529

[appliesTo](#): 4828CC14-1437-45bc-9B07-AD6F015E5F28, bf967a86-0de6-11d0-a285-00aa003049e2, and bf967aba-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.60 User-Logon**

[name](#): User-Logon

[rightsGuid](#): 5f202010-79a5-11d0-9020-00c04fc2d4cf

[appliesTo](#): 4828CC14-1437-45bc-9B07-AD6F015E5F28 and bf967aba-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.61 Membership**

[name](#): Membership

[rightsGuid](#): bc0ac240-79a9-11d0-9020-00c04fc2d4cf

[appliesTo](#): 4828CC14-1437-45bc-9B07-AD6F015E5F28 and bf967aba-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.62 Open-Address-Book**

[name](#): Open-Address-Book

[rightsGuid](#): a1990816-4298-11d1-ade2-00c04fd8d5cd

[appliesTo](#): 3e74f60f-3e73-11d1-a9c0-0000f80367c1

#### 7.1.1.2.7.63 Personal-Information

[name](#): Personal-Information

[rightsGuid](#): 77B5B886-944A-11d1-AEBD-0000F80367C1

[appliesTo](#):

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- bf967a86-0de6-11d0-a285-00aa003049e2
- 5cb41ed0-0e4c-11d0-a286-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2

#### 7.1.1.2.7.64 Email-Information

[name](#): Email-Information

[rightsGuid](#): E45795B2-9455-11d1-AEBD-0000F80367C1

[appliesTo](#):

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- bf967a9c-0de6-11d0-a285-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2

#### 7.1.1.2.7.65 Web-Information

[name](#): Web-Information

[rightsGuid](#): E45795B3-9455-11d1-AEBD-0000F80367C1

[appliesTo](#):

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- 5cb41ed0-0e4c-11d0-a286-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2

#### 7.1.1.2.7.66 DS-Replication-Get-Changes

[name](#): DS-Replication-Get-Changes

[rightsGuid](#): 1131f6aa-9c07-11d1-f79f-00c04fc2dcd2

[appliesTo](#):

- bf967a8f-0de6-11d0-a285-00aa003049e2
- bf967a87-0de6-11d0-a285-00aa003049e2
- 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.67 DS-Replication-Synchronize**

[name](#): DS-Replication-Synchronize

[rightsGuid](#): 1131f6ab-9c07-11d1-f79f-00c04fc2dcd2

[appliesTo](#):

- bf967a8f-0de6-11d0-a285-00aa003049e2
- bf967a87-0de6-11d0-a285-00aa003049e2
- 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.68 DS-Replication-Manage-Topology**

[name](#): DS-Replication-Manage-Topology

[rightsGuid](#): 1131f6ac-9c07-11d1-f79f-00c04fc2dcd2

[appliesTo](#):

- bf967a8f-0de6-11d0-a285-00aa003049e2
- bf967a87-0de6-11d0-a285-00aa003049e2
- 19195a5b-6da0-11d0-afd3-00c04fd930c9

#### **7.1.1.2.7.69 Change-Schema-Master**

[name](#): Change-Schema-Master

[rightsGuid](#): e12b56b6-0a95-11d1-adbb-00c04fd8d5cd

[appliesTo](#): bf967a8f-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.7.70 DS-Replication-Get-Changes-In-Filtered-Set**

[name](#): DS-Replication-Get-Changes-In-Filtered-Set

[rightsGuid](#): 89e95b76-444d-4c62-991a-0facbeda640c

[appliesTo](#):

- 19195a5b-6da0-11d0-afd3-00c04fd930c9
- bf967a87-0de6-11d0-a285-00aa003049e2
- bf967a8f-0de6-11d0-a285-00aa003049e2

#### **7.1.1.2.8 Forest Updates Container**

[parent](#): Config NC root object

[name](#): ForestUpdates

[objectClass](#): [container](#)

#### 7.1.1.2.8.1 Operations Container

[parent](#): Forest Updates container

[name](#): Operations

[objectClass](#): [container](#)

#### 7.1.1.2.8.2 Windows2003Update Container

[parent](#): Forest Updates container

[name](#): Windows2003Update

[objectClass](#): [container](#)

[revision](#): Number of times updates have been applied.

#### 7.1.1.2.8.3 ActiveDirectoryUpdate Container

[parent](#): Forest Updates container

[name](#): ActiveDirectoryUpdate

[objectClass](#): [container](#)

[revision](#): Number of times updates have been applied.

**Note** The ActiveDirectoryUpdate container is applicable only to Windows Server 2008.

#### 7.1.1.2.8.4 ActiveDirectoryRodcUpdate Container

[parent](#): Forest Updates container

[name](#): ActiveDirectoryRodcUpdate

[objectClass](#): [container](#)

[revision](#): Number of times updates have been applied.

**Note** The ActiveDirectoryRodcUpdate container is applicable only to Windows Server 2008.

#### 7.1.1.3 Critical Domain Objects

References:

- FSMO Roles
- Forest Invariants
- Security
- Originating Updates
- LDAP Protocol

Attribute Syntaxes: DN-Binary

Glossary Terms: NC, NC Replica, NC root, DC, **Domain** NC, FSMO, Forest, **UUID**, SPN, PDC, RID

LDAP attributes: [name](#), [objectClass](#), [distinguishedName](#), [systemFlags](#), [primaryGroupID](#), [servicePrincipalName](#), [dNSHostName](#), [msDS-AdditionalDnsHostName](#), [wellKnownObjects](#), [isDeleted](#), [revision](#)

LDAP classes: [computer](#), [container](#), [msDS-QuotaContainer](#), [infrastructureUpdate](#), [organizationalUnit](#), [domainPolicy](#), [samServer](#)

WKGuids: GUID\_USERS\_CONTAINER\_W, GUID\_COMPUTERS\_CONTAINER\_W, GUID\_SYSTEMS\_CONTAINER\_W, GUID\_DOMAIN\_CONTROLLERS\_CONTAINER\_W, GUID\_INFRASTRUCTURE\_CONTAINER\_W, GUID\_DELETED\_OBJECTS\_CONTAINER\_W, GUID\_LOSTANDFOUND\_CONTAINER\_W, GUID\_FOREIGNSECURITYPRINCIPALS\_CONTAINER\_W, GUID\_PROGRAM\_DATA\_CONTAINER\_W, GUID\_NTDS\_QUOTAS\_CONTAINER\_W

Constants

- [systemFlags](#) bits: FLAG\_DISALLOW\_DELETE, FLAG\_DOMAIN\_DISALLOW\_RENAME, FLAG\_DOMAIN\_DISALLOW\_MOVE
- [userAccountControl](#) bits: ADS\_UF\_SERVER\_TRUST\_ACCOUNT, ADS\_UF\_TRUSTED\_FOR\_DELEGATION
- [groupType](#) bits: GROUP\_TYPE\_RESOURCE\_GROUP, GROUP\_TYPE\_SECURITY\_ENABLED, GROUP\_TYPE\_ACCOUNT\_GROUP

#### 7.1.1.3.1 Domain Controller Object

In AD/DS, each normal (not read-only) DC in a **domain** has a domain controller object in its default NC. The DC's domain controller object is the DC's [computer](#) object (subject to the [computer](#) object constraints specified in [MS-SAMR] sections [3.2.1.6](#) and [3.2.1.8](#)) with additional requirements as described below.

An AD/DS RODC has a read-only domain controller object as specified in section [7.1.1.3.2](#). An AD/LDS DC does not have a domain controller object.

[objectClass](#): [computer](#)

[userAccountControl](#): { ADS\_UF\_SERVER\_TRUST\_ACCOUNT | ADS\_UF\_TRUSTED\_FOR\_DELEGATION }

[primaryGroupID](#): Contains the value 516.

This attribute is populated by the system during creation of the DC corresponding to the DC object. The primary group of an DC object is the **domain** relative well-known Domain Controllers security group. So the [primaryGroupID](#) attribute of a DC object equals the RID of the Domain Controllers security group, 516.

[servicePrincipalName](#): This attribute contains all of the SPNs for a normal (not read-only) DC as specified in [MS-DRSR] section [2.2.2](#).

[dNSHostName](#): Fully qualified DNS name of the DC.

[msDS-AdditionalDnsHostName](#): Additional DNS names by which the DC can be identified.

### 7.1.1.3.2 Read-Only Domain Controller Object

Each RODC in a **domain** has a read-only **domain** controller object in its default NC. The DC's RODC object is the DC's [computer](#) object (subject to the [computer](#) object constraints specified in [MS-SAMR] sections [3.2.1.6](#) and [3.2.1.8](#)) with additional requirements as described below. An RODC object cannot be created on Windows Server 2003 and earlier DCs, and cannot be created until the **Read-Only Domain Controllers** object (section [7.1.1.6.13](#)) exists in the **domain**.

[objectClass](#): [computer](#)

[userAccountControl](#): {ADS\_UF\_PARTIAL\_SECRETS\_ACCOUNT | ADS\_UF\_WORKSTATION\_TRUST\_ACCOUNT}

[primaryGroupID](#): Contains the value 521.

This attribute is populated during creation of the RODC corresponding to the RODC object. The primary group of an RODC object is the **domain** relative well-known **Read-Only Domain Controllers** security group. So the [primaryGroupID](#) attribute of an RODC object equals the RID of the **Read-Only Domain Controllers** security group, 521.

[servicePrincipalName](#): This attribute contains all of the SPNs for the RODC as specified in [MS-DRSR] section [2.2.2](#).

[dNSHostName](#): Fully qualified DNS name of the RODC.

[msDS-AdditionalDnsHostName](#): Additional DNS names by which the RODC can be identified.

[msDS-RevealedUsers](#): Contains information about the [user](#) objects whose secrets are cached at this RODC. This attribute is maintained by the system; see procedure UpdateRevealedList, [MS-DRSR] section [4.1.10.5.8](#). A more usable form of this attribute is the **constructed attribute** [msDS-RevealedList](#), specified in section [3.1.1.4.5.34](#).

[msDS-AuthenticatedToAccountlist](#): Contains a list of [user](#) objects that have attempted to authenticate at this RODC. This attribute is a back link attribute whose corresponding forward link is the [msDS-AuthenticatedAtDC](#) attribute. The [msDS-AuthenticatedAtDC](#) attribute is maintained by the system; see section [7.1.4.6](#).

[msDS-NeverRevealGroup](#): This attribute is maintained by an administrator. It contains a set of [user](#) and security-enabled [group](#) objects. A user in this set, or reachable from this set by traversing any number of [member](#) links from a [group](#) in this set, will not change state from not being cached to being cached at this RODC. If a user is added to this attribute (directly or indirectly) while one its secrets is already cached, the secret remains cached until the secret changes, at which time the caching stops. For the use of this attribute see procedure RevealSecretsForUserAllowed, [MS-DRSR] section [4.1.10.5.12](#).

[msDS-RevealOnDemandGroup](#): This attribute is maintained by an administrator. It contains a set of [user](#) and security-enabled [group](#) objects. A user in this set, or reachable from this set by traversing any number of [member](#) links from a [group](#) in this set, and not excluded by membership in [msDS-NeverRevealGroup](#), can change state from not being cached to being cached at this RODC. For the use of this attribute see procedure RevealSecretsForUserAllowed, [MS-DRSR] section [4.1.10.5.12](#).

[msDS-KrbTgtLink](#): This attribute is populated during creation of the RODC object. It contains a reference to the RODC's secondary Kerberos ticket granting ticket account. See [\[MS-KILE\]](#) section [3.1.5.6](#).

[managedBy](#): If the value of this attribute points to a valid security principal, that security principal will be an implicit member of the administrators group of this RODC, and this applies to this RODC only.

#### 7.1.1.4 Well-Known Objects

Within each NC (excluding the schema NC) there are certain well-known system objects that can be referred to using a well-known **GUID** (see section [3.1.1.3](#) for more information). **Domain** and config NC root objects contain an attribute called [wellKnownObjects](#) that lists the well-known system objects within that NC. Each value in this list is an Object(DN-Binary) value where the Binary portion is the well-known **GUID** in binary form and the DN portion is the DN of the object. The well-known **GUID** can be used in conjunction with the NC DN to refer to the object (for more information see section [3.1.1.3](#)).

The following invariants apply to the [wellKnownObjects](#) attribute on the NC root object and the referred to objects:

- For each of the well-known **GUIDs** listed below for a given NC, the [wellKnownObjects](#) attribute on the NC root object must contain a value such that the binary portion matches the well-known **GUID**. There must be exactly one such value.
- If rename of the referred to object is permitted (based on the value of the [systemFlags](#) attribute on each object), the DN portion of the value is updated.
- The well-known Users **container** and the well-known Computers **container** in the **domain** NC may be redirected, under the following constraints:
  - The modification must be made on a DC owning the PDC FSMO.
  - The modification removes the reference to the existing object and adds a new reference in the same operation.
  - The new object being referred to must not be in the System **container** of the **domain** NC.
  - The new object being referred to must exist, and if different from the currently referred to Users or Computers **containers**, it must not have the following bits in the [systemFlags](#) attribute: FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE
  - As part of the redirection, the following flags are added to the new object being referred to and removed from the old object: FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE

In AD/DS, the following well-known objects exist within each **domain** NC.

RDN	Symbolic name for well-known GUID
Computers	GUID_COMPUTERS_CONTAINER_W
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
Domain Controllers	GUID_DOMAIN_CONTROLLERS_CONTAINER_W
ForeignSecurityPrincipals	GUID_FOREIGNSECURITYPRINCIPALS_CONTAINER_W
Infrastructure	GUID_INFRASTRUCTURE_CONTAINER_W



RDN	Symbolic name for well-known GUID
LostAndFound	GUID_LOSTANDFOUND_CONTAINER_W
Microsoft <sup>Note 1</sup>	GUID_MICROSOFT_PROGRAM_DATA_CONTAINER_W
NTDS Quotas	GUID_NTDS_QUOTAS_CONTAINER_W
Program Data	GUID_PROGRAM_DATA_CONTAINER_W
System	GUID_SYSTEMS_CONTAINER_W
Users	GUID_USERS_CONTAINER_W

Note 1 The Microsoft **container** is a child of the Program Data **container**.

In AD/DS, the following well-known objects exist within each application NC.

RDN	Symbolic name for well-known GUID
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
Infrastructure	GUID_INFRASTRUCTURE_CONTAINER_W
LostAndFound	GUID_LOSTANDFOUND_CONTAINER_W
NTDS Quotas	GUID_NTDS_QUOTAS_CONTAINER_W

In AD/DS, the following well-known objects exist within the config NC.

RDN	Symbolic name for well-known GUID
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
LostAndFoundConfig	GUID_LOSTANDFOUND_CONTAINER_W
NTDS Quotas	GUID_NTDS_QUOTAS_CONTAINER_W

In AD/LDS, the following well-known objects exist within each application NC.

RDN	Symbolic name for well-known GUID
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
ForeignSecurityPrincipals <sup>Note 2</sup>	GUID_FOREIGNSECURITYPRINCIPALS_CONTAINER_W
LostAndFound	GUID_LOSTANDFOUND_CONTAINER_W
NTDS Quotas	GUID_NTDS_QUOTAS_CONTAINER_W
Roles	GUID_USERS_CONTAINER_W

Note 2 The ForeignSecurityPrincipals **container** is created (and the corresponding value created in the [wellKnownObjects](#) attribute) when the first [foreignSecurityPrincipal](#) object is created in the NC.

In AD/LDS, the following well-known objects exist within the config NC.

RDN	Symbolic name for well-known GUID
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
ForeignSecurityPrincipals	GUID_FOREIGNSECURITYPRINCIPALS_CONTAINER_W
LostAndFoundConfig	GUID_LOSTANDFOUND_CONTAINER_W
NTDS Quotas	GUID_NTDS_QUOTAS_CONTAINER_W
Roles	GUID_USERS_CONTAINER_W

The following table gives the **GUID** values for each of the symbolic names of the well-known **GUIDs**.

Symbolic name for well-known GUID	GUID
GUID_COMPUTERS_CONTAINER_W	AA312825768811D1ADED00C04FD8D5CD
GUID_DELETED_OBJECTS_CONTAINER_W	18E2EA80684F11D2B9AA00C04F79F805
GUID_DOMAIN_CONTROLLERS_CONTAINER_W	A361B2FFFFD211D1AA4B00C04FD7D83A
GUID_FOREIGNSECURITYPRINCIPALS_CONTAINER_W	22B70C67D56E4EFB91E9300FCA3DC1AA
GUID_INFRASTRUCTURE_CONTAINER_W	2FBAC1870ADE11D297C400C04FD8D5CD
GUID_LOSTANDFOUND_CONTAINER_W	AB8153B7768811D1ADED00C04FD8D5CD
GUID_MICROSOFT_PROGRAM_DATA_CONTAINER_W	F4BE92A4C777485E878E9421D53087DB
GUID_NTDS_QUOTAS_CONTAINER_W	6227F0AF1FC2410D8E3BB10615BB5B0F
GUID_PROGRAM_DATA_CONTAINER_W	09460C08AE1E4A4EA0F64AEE7DAA1E5A
GUID_SYSTEMS_CONTAINER_W	AB1D30F3768811D1ADED00C04FD8D5CD
GUID_USERS_CONTAINER_W	A9D1CA15768811D1ADED00C04FD8D5CD

#### 7.1.1.4.1 Lost and Found Container

Each **domain** NC, application NC, and config NC contains a lost and found **container** for objects that are orphaned as a result of add and delete operations that originated on different DCs.

[objectClass](#): [lostAndFound](#)

[systemFlags](#): On **domain** and application NCs: {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

On Config NC: {FLAG\_DISALLOW\_DELETE}

#### 7.1.1.4.2 Deleted Objects Container

Each **domain** NC, application NC and the config NC contain a Deleted Objects **container**. Objects within the **domain** NC that are deleted are stored in this **container** (unless indicated otherwise by the object's [systemFlags](#)) until an amount of time equal to the tombstone lifetime has passed after which they are permanently deleted. To ensure that this **container** does not get garbage collected,

the replication metadata for the [isDeleted](#) attribute must show that the time at which the [isDeleted](#) attribute was set to true is 9999-12-29. See section [3.1.1.5.5](#) for more information about the tombstone lifetime and the Deleted Objects **container**.

[objectClass](#): [container](#)

[isDeleted](#): true

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

#### 7.1.1.4.3 NTDS Quotas Container

Each **domain** NC, application NC, and the config NC contain a NTDS Quotas **container** which contains quotas restricting the number of objects that can be created by a specified security principal.

[objectClass](#): [msDS-QuotaContainer](#)

[systemFlags](#): FLAG\_DISALLOW\_DELETE

#### 7.1.1.4.4 Infrastructure Object

In AD/DS, each **domain** and application NC has an infrastructure object that maintains a reference to the current Infrastructure role owner. This object is not present in AD/LDS.

[objectClass](#): [infrastructureUpdate](#)

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

[fSMORoleOwner](#): This value refers to the [nTDSDSA](#) object of the DC that owns the Infrastructure FSMO role.

#### 7.1.1.4.5 Domain Controllers OU

This is a well-known **container** within the **domain** NC containing the [computer](#) objects for **domain** controllers within this **domain**.

[objectClass](#): [organizationalUnit](#)

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

#### 7.1.1.4.6 Users Container

Each **domain** NC contains a well-known default users **container**.

[objectClass](#): [container](#)

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

#### 7.1.1.4.7 Computers Container

Each **domain** NC contains a well-known default computers **container**.

[objectClass](#): [container](#)

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

#### 7.1.1.4.8 Program Data Container

[objectClass](#): [container](#)

[systemFlags](#): 0

#### 7.1.1.4.9 Foreign Security Principals Container

In AD/DS, each **domain** NC contains a well-known foreign security principals **container**. This **container** holds objects of class [foreignSecurityPrincipal](#). These objects represent **security principals** from trusted **domains** external to the forest, and allow foreign **security principals** to become members of groups within the **domain**.

In AD/LDS, the config NC contains a well-known foreign **security principals container**. It stores foreign **security principals** from outside of the AD/LDS forest.

In an AD/LDS application NC, a foreign **security principals container** is created (and the corresponding value created in the [wellKnownObjects](#) attribute) when the first [foreignSecurityPrincipal](#) object is created in the application NC.

The automatic creation of [foreignSecurityPrincipal](#) objects is specified in sections [3.1.1.5.2.4](#) and [3.1.1.5.3.3](#)).

[name](#): ForeignSecurityPrincipals

[parent](#): **domain** NC root on AD/DS; config NC root on AD/LDS.

[objectClass](#): [container](#)

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

#### 7.1.1.4.10 System Container

[name](#): System

[parent](#): **Domain** NC root object

[objectClass](#): [container](#)

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

##### 7.1.1.4.10.1 Password Settings Container

In AD/DS, each domain NC contains a well-known Password Settings container. This container is initially empty, but is designed to contain objects of class [msDS-PasswordSettings](#). These objects represent password settings for a group of users in the domain. For more information, see [\[MS-SAMR\]](#) section 3.2.1.5.

[name](#): Password Settings container

[parent](#): System container

[objectClass](#): [msDS-PasswordSettings](#)

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

#### 7.1.1.4.11 Builtin Container

In AD/DS, each **domain** NC contains this **container**. Its children are described below. This **container** is not present in AD/LDS.

[name](#): Builtin

[parent](#): **domain** NC root

[objectClass](#): [builtinDomain](#)

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE }

The children of the Builtin Container are well-known **security principals** from the built-in domain.

Each child of the Builtin Container is a group with the following attributes:

[parent](#): Builtin **container**

[objectClass](#): [group](#)

[objectSid](#): The **domain** portion is the built-in domain SID (S-1-5-32). The RID portion is specified per-object below. For instance the Account Operators RID is 548, so the Account Operators [objectSid](#) is S-1-5-32-548.

[systemFlags](#): { DISALLOW\_DELETE | DOMAIN\_DISALLOW\_RENAME | DOMAIN\_DISALLOW\_MOVE }

[groupType](#): { BUILTIN\_LOCAL\_GROUP | RESOURCE\_GROUP | SECURITY\_ENABLED }

Unless otherwise noted below, the initial membership of each group is empty. After initialization, the administrator controls the membership of each group.

##### 7.1.1.4.11.1 Account Operators Group Object

[name](#): Account Operators

**RID**: 548

##### 7.1.1.4.11.2 Administrators Group Object

[name](#): Administrators

**RID**: 544

[member](#): Administrator (section [7.1.1.6.1](#)), Domain Administrators (section [7.1.1.6.5](#)), Enterprise Administrators (section [7.1.1.6.10](#)).

##### 7.1.1.4.11.3 Backup Operators Group Object

[name](#): Backup Operators

**RID:** 551

#### **7.1.1.4.11.4 Certificate Service DCOM Access Group Object**

[name](#): Certificate Service DCOM Access

**RID:** 574

#### **7.1.1.4.11.5 Cryptographic Operators Group Object**

[name](#): Cryptographic Operators

**RID:** 569

#### **7.1.1.4.11.6 Distributed COM Users Group Object**

[name](#): Distributed COM Users

**RID:** 562

#### **7.1.1.4.11.7 Event Log Readers Group Object**

[name](#): Event Log Readers

**RID:** 573

#### **7.1.1.4.11.8 Guests Group Object**

[name](#): Guests

**RID:** 546

[member](#): Guest (section [7.1.1.6.2](#)), Domain Guests (section [7.1.1.6.8](#))

#### **7.1.1.4.11.9 IIS\_IUSRS Group Object**

[name](#): IIS\_IUSRS

**RID:** 568

[member](#): NT AUTHORITY\IUSR well-known security principal (SID S-1-5-17).

#### **7.1.1.4.11.10 Incoming Forest Trust Builders Group Object**

[name](#): Incoming Forest Trust Builders

**RID:** 557

#### **7.1.1.4.11.11 Network Configuration Operators Group Object**

[name](#): Network Configuration Operators

**RID:** 556

#### **7.1.1.4.11.12 Performance Log Users Group Object**

[name](#): Performance Log Users

**RID**: 559

#### **7.1.1.4.11.13 Performance Monitor Users Group Object**

[name](#): Performance Monitor Users

**RID**: 558

#### **7.1.1.4.11.14 Pre-Windows 2000 Compatible Access Group Object**

[name](#): Pre-Windows 2000 Compatible Access

**RID**: 554

[member](#): The initial membership of this group depends on the version of Windows running on the first DC of the **domain** and on the administrator's choice between "Pre-Windows 2000 Compatible Permissions mode" and "Windows 2000-only permissions mode". In Windows 2000 Server, in the Pre-Windows 2000 Compatible Permissions mode, Everyone (S-1-1-0) is a member, and in the Windows 2000-only permissions mode, the membership is empty. In Windows Server 2003, in the Pre-Windows 2000 Compatible Permissions mode, Everyone (S-1-1-0) and Anonymous (S-1-5-7) are members, and in the Windows 2000-only permissions mode, Authenticated Users (S-1-5-11) are members.

#### **7.1.1.4.11.15 Print Operators Group Object**

[name](#): Print Operators

**RID**: 550

#### **7.1.1.4.11.16 Remote Desktop Users Group Object**

[name](#): Remote Desktop Users

**RID**: 555

#### **7.1.1.4.11.17 Replicator Group Object**

[name](#): Replicator

**RID**: 552

#### **7.1.1.4.11.18 Server Operators Group Object**

[name](#): Server Operators

**RID**: 549

#### **7.1.1.4.11.19 Terminal Server License Servers Group Object**

[name](#): Terminal Server License Servers

**RID**: 561

#### 7.1.1.4.11.20 Users Group Object

[name](#): Users

**RID**: 545

[member](#): Domain Users group (section [7.1.1.6.9](#)), NT AUTHORITY\Authenticated Users well-known security principal (SID S-1-5-11), NT AUTHORITY\INTERACTIVE well-known security principal (SID S-1-5-4).

#### 7.1.1.4.11.21 Windows Authorization Access Group Group Object

[name](#): Windows Authorization Access Group

**RID**: 560

[member](#): NT AUTHORITY\ENTERPRISE DOMAIN CONTROLLERS well-known security principal (SID S-1-5-9).

#### 7.1.1.4.12 Roles Container

In AD/LDS, each application NC and the config NC contain this **container**. It stores the well-known AD/LDS groups for this NC. This **container** is not present in AD/DS, nor are any of its child objects specified below.

[name](#): Roles

[parent](#): Application NC root or config NC root.

[objectClass](#): [container](#)

[systemFlags](#): {FLAG\_DISALLOW\_DELETE }

Each child of the Roles Container is a group with the following attributes:

[parent](#): Roles Container

[objectClass](#): [group](#)

[objectSid](#): A SID with two subauthorities, consisting of the [objectSid](#) of the NC root followed by the RID specified for each child below.

[groupType](#): { ACCOUNT\_GROUP | SECURITY\_ENABLED }

[member](#): Unless otherwise noted in the following sections, the initial membership of each group is empty. After initialization the administrator can modify the membership of each group.

#### 7.1.1.4.12.1 Administrators Group Object

[name](#): Administrators

**RID**: 519 (in the config NC) or 512 (in an application NC).

[member](#): At least one [foreignSecurityPrincipal](#) is configured into this group by the administrator when creating a forest.



#### 7.1.1.4.12.2 Readers Group Object

[name](#): Readers

**RID**: 514

#### 7.1.1.4.12.3 Users Group Object

This group is used in constructing an AD/LDS security context as specified in section [5.1.3.4](#).

[name](#): Users

**RID**: 513

#### 7.1.1.4.12.4 Instances Group Object

In AD/LDS, every DC's service account belongs to this group. The system attempts to maintain this group, although an administrator can still modify the membership. This group is only present in the Roles **container** of the config NC.

[name](#): Instances

**RID**: 518

[member](#): An AD/LDS DC ensures that its service account is a member of this group. If an AD/LDS DC's service account is Network Service or Local System, it also ensures that its [computer](#) object is a member of this group.

#### 7.1.1.5 Other Well-Known Objects

Each NC root object may contain a list of well-defined objects in addition to those defined by the [wellKnownObjects](#) attribute. This permits retrieving an object after it has been moved by using just the **GUID** and the **domain** name. This list is contained in the [otherWellKnownObjects](#) attribute on each NC root object. The constraints that exist around the values in the [wellKnownObjects](#) attribute do not exist for this attribute.

##### 7.1.1.5.1 AdminSDHolder Object

[parent](#): System Container

[name](#): AdminSDHolder

[objectClass](#): **container**

[systemFlags](#): {FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE}

##### 7.1.1.5.2 Default Domain Policy Container

[parent](#): System Container

[name](#): Default Domain Policy

[objectClass](#): [domainPolicy](#)

### 7.1.1.5.3 Sam Server Object

[parent](#): System Container

[name](#): Server

[objectClass](#): [samServer](#)

[systemFlags](#): FLAG\_DISALLOW\_DELETE | FLAG\_DOMAIN\_DISALLOW\_RENAME | FLAG\_DOMAIN\_DISALLOW\_MOVE

### 7.1.1.5.4 Domain Updates Container

[parent](#): System Container

[name](#): DomainUpdates

[objectClass](#): [container](#)

#### 7.1.1.5.4.1 Operations Container

[parent](#): Domain Updates Container

[name](#): Operations

[objectClass](#): [container](#)

#### 7.1.1.5.4.2 Windows2003Updates Container

[parent](#): Domain Updates Container

[name](#): Windows2003Updates

[objectClass](#): [container](#)

[revision](#): Number of times updates have been applied.

#### 7.1.1.5.4.3 ActiveDirectoryUpdate Container

[parent](#): Domain Updates container

[name](#): ActiveDirectoryUpdate

[objectClass](#): [container](#)

[revision](#): Number of times updates have been applied.

**Note** The ActiveDirectoryUpdate container is applicable only to Windows Server 2008.

### 7.1.1.6 Well-Known Domain-Relative Security Principals

In each **domain** NC, there are certain well-known **security principals**. These well-known **security principals** are given default privileges in the **domain**. For more information, see section [5](#) and also see [MS-SAMR] section [3.2.4.2](#).

Each of these objects has the following attributes:

[parent](#): Users container (section [7.1.1.4.6](#)).

[objectSid](#): a SID consisting of the [objectSid](#) of the **domain** NC root followed by the RID specified for each child below.

The objects of class [user](#) have the following attribute:

[primaryGroupID](#): This value is a RID, which refers to another Well-Known **domain** relative security principal, by RID.

#### **7.1.1.6.1 Administrator**

[name](#): Administrator

[objectClass](#): [user](#)

**RID**: 500

[primaryGroupID](#): 513 (Domain Users)

#### **7.1.1.6.2 Guest**

[name](#): Guest

[objectClass](#): [user](#)

**RID**: 501

[primaryGroupID](#): 514 (Domain Guests)

#### **7.1.1.6.3 Key Distribution Center Service Account**

[name](#): krbtgt

[objectClass](#): [user](#)

**RID**: 502

[primaryGroupID](#): 513 (Domain Users)

#### **7.1.1.6.4 Cert Publishers**

[name](#): Cert Publishers

[objectClass](#): [group](#)

**RID**: 517

[groupType](#): {GROUP\_TYPE\_RESOURCE\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED}

#### **7.1.1.6.5 Domain Administrators**

[name](#): Domain Admins

[objectClass](#): [group](#)

**RID**: 512

[groupType](#): { GROUP\_TYPE\_ACCOUNT\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

#### 7.1.1.6.6 Domain Computers

[name](#): Domain Computers

[objectClass](#): [group](#)

**RID**: 515

[groupType](#): { GROUP\_TYPE\_ACCOUNT\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

#### 7.1.1.6.7 Domain Controllers

[name](#): Domain Controllers

[objectClass](#): [group](#)

**RID**: 516

[groupType](#): { GROUP\_TYPE\_ACCOUNT\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

#### 7.1.1.6.8 Domain Guests

[name](#): Domain Guests

[objectClass](#): [group](#)

**RID**: 514

[groupType](#): { GROUP\_TYPE\_ACCOUNT\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

#### 7.1.1.6.9 Domain Users

[name](#): Domain Users

[objectClass](#): [group](#)

**RID**: 513

[groupType](#): { GROUP\_TYPE\_ACCOUNT\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

#### 7.1.1.6.10 Enterprise Administrators

This group exists only in the forest root **domain**.

[name](#): Enterprise Admins

[objectClass](#): [group](#)

**RID**: 519

[groupType](#):

- if the forest root **domain** is mixed (section [7.1.4.1](#)):
  - { GROUP\_TYPE\_ACCOUNT\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

- if the forest root **domain** is not mixed:
  - { GROUP\_TYPE\_UNIVERSAL\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

#### 7.1.1.6.11 Group Policy Creator Owners

[name](#): Group Policy Creator Owners

[objectClass](#): [group](#)

**RID**: 520

[groupType](#): { GROUP\_TYPE\_ACCOUNT\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

#### 7.1.1.6.12 RAS and IAS Servers

[name](#): RAS and IAS Servers

[objectClass](#): [group](#)

**RID**: 553

[groupType](#): { GROUP\_TYPE\_RESOURCE\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

#### 7.1.1.6.13 Read-Only Domain Controllers

[name](#): **Read-Only Domain Controllers**

[objectClass](#): [group](#)

**RID**: 521

[groupType](#): { GROUP\_TYPE\_ACCOUNT\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

This group is created in a **domain** by the PDC, the first time a Windows Server 2008 DC holds the PDC FSMO.

#### 7.1.1.6.14 Enterprise Read-Only Domain Controllers

**name**: Enterprise **Read-Only Domain Controllers**

**objectClass**: [group](#)

**RID**: 498

[groupType](#): { GROUP\_TYPE\_UNIVERSAL\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

This group is created in the root **domain** by the root **domain** PDC, the first time a Windows Server 2008 DC holds the root **domain** PDC FSMO.

#### 7.1.1.6.15 Schema Admins

This group exists only in the forest root **domain**.

[name](#): Schema Admins

[objectClass](#): [group](#)

**RID:** 518

[groupType](#):

- if the forest root **domain** is mixed (section [7.1.4.1](#)):
  - { GROUP\_TYPE\_ACCOUNT\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }
- if the forest root **domain** is not mixed:
  - { GROUP\_TYPE\_UNIVERSAL\_GROUP | GROUP\_TYPE\_SECURITY\_ENABLED }

### 7.1.2 Forest Invariants

References: [nTDSDSA](#) object, [server](#) object, Domain Controller object, SPN construction, [crossRef](#) object, NC root object

Glossary Terms: DC, NC, NC Replica

LDAP Attributes: [serverReference](#), [dNSHostName](#), [servicePrincipalName](#), [nCName](#), [msDS-NC-Replica-Locations](#), [msDS-hasMasterNCs](#), [msDS-HasInstantiatedNCs](#), [hasPartialReplicaNCs](#)

LDAP Classes: [nTDSDSA](#), [server](#), [crossRef](#)

Constants: NTDS\_DS\_OPT\_IS\_GC

#### 7.1.2.1 DC Existence

For any DC in the forest, the following objects must exist:

- [nTDSDSA](#): See section [7.1.1](#).
- [server](#) object: See section [7.1.1](#).
- Domain Controller object (in AD/DS, not AD/LDS): See section [7.1.1](#).

For the purposes of this section, a Read-Only Domain Controller object is a Domain Controller object.

Any one of these objects can be said to "represent" the DC.

Relationships:

- The [server](#) object is the parent of the [nTDSDSA](#) object. On AD/DS, the name of the [server](#) object is the computer name of the DC; On AD/LDS, the name of the [server](#) object is the computer name, followed by "\$", followed by the instance name of the DC.
- On AD/DS, the attribute [serverReference](#) on the [server](#) object must equal the dsname of the **domain** controller object.
- On AD/DS, the [dNSHostName](#) attribute of the **domain** controller object must equal the [dNSHostName](#) attribute of the [server](#) object.
- The [dNSHostName](#) attribute of the [server](#) object must equal the DNS Hostname of the computer that is physically the DC.
- On AD/DS, every value of the [servicePrincipalName](#) attribute of the **domain** controller object, which has a DNS Hostname as the instance name (see section [5.1.1.4](#), "Mutual Authentication,"

for SPN construction), should have instance name equal to the [dNSHostName](#) of the **domain** controller object.

### 7.1.2.2 NC Existence

For any NC in the forest, the following objects must exist:

- [crossRef](#): see section [7.1.1](#).
- NC root: see section [7.1.1](#).

Either of these objects can be said to "represent" the NC.

Relationships:

- The [nCName](#) attribute of the [crossRef](#) object must equal the dsname of the NC root object.

### 7.1.2.3 Hosting Invariants

#### 7.1.2.3.1 DC and Application NC Replica

A DC is instructed to host an application NC replica if:

- The attribute [msDS-NC-Replica-Locations](#) on the [crossRef](#) representing the NC contains the dsname of the [nTDSDSA](#) object representing the DC.

A DC is hosting an application NC replica when the following:

- The attribute [msDS-hasMasterNCs](#) on the [nTDSDSA](#) object representing the DC contains the dsname of the NC root representing the NC.
- The attribute [msDS-HasInstantiatedNCs](#) on the [nTDSDSA](#) object representing the DC contains an Object(DN-Binary) value such that the DN field is the dsname of the NC root representing the NC, and the Data field contains the value of the [instanceType](#) attribute on the NC root object on the DC.

#### 7.1.2.3.2 DC and Regular Domain NC Replica

A DC is instructed to host a regular **domain** NC replica if:

- The **domain** controller object representing the DC is in the **domain** NC.

A DC is hosting a regular **domain** NC replica when the following:

- The attribute [msDS-hasMasterNCs](#) and attribute [hasMasterNCs](#) on the [nTDSDSA](#) object representing the DC contain the dsname of the NC root representing the **domain** NC.
- The attribute [msDS-HasInstantiatedNCs](#) on the [nTDSDSA](#) object representing the DC contains an Object(DN-Binary) value such that the DN field is the dsname of the **domain** NC root representing the **domain** NC, and the Data field contains the value of the [instanceType](#) attribute on the **domain** NC root object on the DC.
- The attribute [msDS-HasDomainNCs](#) on the [nTDSDSA](#) object representing the DC equals the dsname of the **domain** NC root. A DC hosts only one full **domain** NC replica.

### 7.1.2.3.3 DC and Schema/Config NC Replicas

Every DC is instructed to host the Schema and Config NC replicas.

A DC is hosting the schema and Config NC replicas when the following:

- The attribute [msDS-hasMasterNCs](#) and attribute [hasMasterNCs](#) on the [nTDSDSA](#) object representing the DC contain the dsname of both the NC roots representing the Schema and Config NCs.
- The attribute [msDS-HasInstantiatedNCs](#) on the [nTDSDSA](#) object representing the DC contains two Object(DN-Binary) values such that the DN fields are the dsname of the NC root representing the config and schema NCs, and the binary fields contain the values of the [instanceType](#) attribute on the config and schema NC root objects on the DC.

### 7.1.2.3.4 DC and Partial Replica NCs Replicas

A DC is instructed to host a partial NC replica of every **domain** NC in the forest if:

- The [options](#) attribute of the [nTDSDSA](#) object representing that DC has the following flag: NTDSOPT\_IS\_GC

A DC hosts a partial NC replica of a **domain** NC when the following:

- The attribute [hasPartialReplicaNCs](#) on the [nTDSDSA](#) object representing the DC contains the dsname of the NC roots representing the **domain** NC.
- The attribute [msDS-HasInstantiatedNCs](#) on the [nTDSDSA](#) object representing the DC contains an Object(DN-Binary) value such that the DN field is the dsname of the NC root representing the NC, and the Data field contains the value of the [instanceType](#) attribute on the NC root object on the DC.

## 7.1.3 Security Descriptor Invariants

Constants:

- LDAP constants: LDAP\_SERVER\_SD\_FLAGS\_OID
- SD flags: OWNER\_SECURITY\_INFORMATION, GROUP\_SECURITY\_INFORMATION, DACL\_SECURITY\_INFORMATION, SACL\_SECURITY\_INFORMATION, SECURITY\_PRIVATE\_OBJECT
- Security **access mask** bits and privileges: SE\_RESTORE\_PRIVILEGE, WRITE\_DAC, WRITE\_OWNER, ACCESS\_SYSTEM\_SECURITY.
- Security Descriptor values stored in Active Directory are in SECURITY\_DESCRIPTOR format (see [\[MS-DTYP\]](#) section 2.4.6). In addition to the defined fields, the RM Control (Resource Manager Control) field is used. It is stored in the Reserved1 byte of the SECURITY\_DESCRIPTOR structure. The following bit may be present in the field:
  - 0x01 == SECURITY\_PRIVATE\_OBJECT.
- Error codes: ERROR\_INVALID\_OWNER

The following security descriptor invariants are maintained by a DC:

1. Each object's SD retains the set of explicit (non-inherited) ACEs stamped in its DACL and SACL (if present). It also retains the owner and group SID values as well as various SD flags (see SD



- reference [\[MS-DTYP\]](#) section **2.4.6**). The owner SID may not be NULL, while the group SID may be NULL.
2. The SD also includes the set of inheritable ACEs from its parent object. It includes both applicable and non-applicable inheritable ACEs. The following exceptions apply to the above rule:
    1. The object is the root of an NC. In this case, the SD does not include any inherited ACEs.
    2. If the ACL (either DACL or SACL) has the "protected from inheritance" flag set. In this case, the ACL does not include inheritable ACEs from the parent object's SD.
    3. The object is deleted. In this case, the set of inheritable ACEs that were obtained from the parent object's SD at the time of object deletion is retained.
  3. When the forest version is W2k3 or above and fDontStandardizeSDs = false, then the ACEs in the ACLs are sorted according to ACE ordering rules (see ACE ordering rules section below). Otherwise, if the forest version is less than W2k3, the order of explicit ACEs supplied by the client is preserved.
  4. The ACEs with inheritedObjectType field present are marked as effective or ineffective (INHERIT\_ONLY flag) according to SD merge rules (see the CreateSecurityDescriptor algorithm in [\[MS-DTYP\]](#) section 2.5.2.3), based on the current value of object's [objectClass](#) attribute. Specifically, the following [objectClass](#) values are considered when processing inheritable ACEs from the parent's SD: the most specific structural [objectClass](#) value, as well as all dynamic auxiliary class values. The static auxiliary classes and non-most specific object classes are not considered. For example, in AD schema, [computer](#) objects have the following [objectClass](#) values: [top](#), [person](#), [organizationalPerson](#), [user](#), and [computer](#). In this case, only the [computer](#) class has to be considered for inheritance processing. For inheritance processing, each effective [objectClass](#) value is converted to the **GUID** (as per schema mapping object classes to **GUIDs**, see Schema section), and supplied as an input to the SD merge routine.
  5. In order to compute the resultant SD value for an object, the CreateSecurityDescriptor algorithm ([\[MS-DTYP\]](#) section 2.5.2.3) is invoked with the following input parameters:
    1. *ParentDescriptor*: If the object is NC root, then NULL; otherwise, the SD value of the parent object.
    2. *CreatorDescriptor*: The current SD value stamped on the object.
    3. *IsContainerObject*: true is always passed.
    4. *AutoInheritFlags*: DACL\_AUTO\_INHERIT | SACL\_AUTO\_INHERIT.
    5. *Token*: When processing an originating SD write, the security information of the requestor is used. Otherwise, SYSTEM security information is used, note, in the case of auto-propagation into children, the information from the token is never used, because all required SD parts are always present and there is nothing that needs to be defaulted.
  6. *GenericMapping*: The following mapping table is used for all AD SD operations:
    - GENERIC\_READ\_MAPPING = READ\_CONTROL | RIGHT\_DS\_LIST\_CONTENTS | RIGHT\_DS\_READ\_PROPERTY | RIGHT\_DS\_LIST\_OBJECT
    - GENERIC\_WRITE\_MAPPING = READ\_CONTROL | RIGHT\_DS\_WRITE\_PROPERTY\_EXTENDED | RIGHT\_DS\_WRITE\_PROPERTY
    - GENERIC\_EXECUTE\_MAPPING = READ\_CONTROL | RIGHT\_DS\_LIST\_CONTENTS

- `GENERIC_ALL_MAPPING = RIGHT_DELETE | READ_CONTROL | WRITE_DAC |  
WRITE_OWNER | RIGHT_DS_CREATE_CHILD | RIGHT_DS_DELETE_CHILD |  
RIGHT_DS_DELETE_TREE | RIGHT_DS_READ_PROPERTY | RIGHT_DS_WRITE_PROPERTY |  
RIGHT_DS_LIST_CONTENTS | RIGHT_DS_LIST_OBJECT | RIGHT_DS_CONTROL_ACCESS |  
RIGHT_DS_WRITE_PROPERTY_EXTENDED`

6. Any CREATOR/OWNER ineffective ACEs has a matching effective ACE granted to the current owner of the object (as obtained from the SD OWNER field).

7. NULL DACLs are disallowed.

The security descriptor invariant maintenance task is implemented using multiple transactions.

### 7.1.3.1 ACE Ordering Rules

ACE ordering rules only apply to ACLs in canonical form, and only when forest version is W2k3 or above. The following rules are applied, in the following order. The next rule is only applied if the previous rule(s) give inconclusive result:

1. Explicit ACEs come before inherited ACEs
2. Deny ACEs come before Allow ACEs
3. Regular ACEs come before object ACEs
4. Within each group, the ACEs are ordered lexicographically (that is, based on octet string comparison rules).

Rules 3 and 4 above are only enforced when forest functionality level is `DS_BEHAVIOR_WIN2003` or above. Otherwise, the order of ACEs within each group defined by rules 1 and 2 is retained as supplied by the user or replication partner.

### 7.1.3.2 SD flags control

When performing an **LDAP** operation (add, modify or search), the client may supply an SD flags control `LDAP_SERVER_SD_FLAGS_OID` with the operation. The value of the control is an integer, which is used to identify which SD parts the client intends to read or modify. When the control is not specified, then the default value of 15 (0x0000000F) is used.

The SD parts are identified using the following bit values: `OWNER_SECURITY_INFORMATION`, `GROUP_SECURITY_INFORMATION`, `DACL_SECURITY_INFORMATION`, `SACL_SECURITY_INFORMATION`, which correspond to OWNER, GROUP, DACL and SACL SD fields, respectively.

For update operations, the bits identify which SD parts are affected by the operation. Note, the client may supply values for other (or all) SD fields. However, the server only updates the fields that are identified by the SD control. The remaining fields are ignored.

### 7.1.3.3 Processing Specifics

1. The clients may send in SD values that include both explicit and inherited ACEs (during add or modify operations). Only the set of explicit ACEs is considered authoritative data. Any inherited ACEs that are included in the SD value are ignored. Instead, the set of inherited ACEs is computed per the rules above, and set on the object.

2. During an add operation, the DC makes sure that the object's security descriptor value is consistent with the parent's SD value (per above rules), at the moment when the add operation is committed.
3. During a move operation, the DC makes sure that the moved object's security descriptor value is consistent with the new parent's SD value (per above rules), at the moment when the move operation is committed. If the moved object has descendant objects (i.e., a tree move was performed), then the SD values of the children objects are updated outside of the move transaction (see Modify DN, section [3.1.1.5.4](#)).
4. During an SD modify operation, the DC ensures that the updated object's security descriptor value is consistent with the parent's SD value (per above rules), at the moment when the modify operation is committed. If the updated object has descendant objects, then the SD values of the children objects are updated outside of the modify transaction.
5. When processing inbound replication containing SD updates, the SD invariant rules are enforced (i.e., it is not guaranteed that the SD value sent by replication partner is consistent with the parent's SD value). It is the responsibility of the DC performing the inbound replication to ensure that the set of inherited ACEs present in the SD is consistent in the subtree rooted at the affected object (per rules above). One exception to this rule is processing inbound replication of a deleted object. In this case, the DC retains the SD value (including both explicit and inherited ACEs) as it is supplied by the replication partner, in cases when it is supplied by the replication partner. If the SD value is not supplied by the replication partner, then the existing SD value is retained.
6. When an originating add operation is processed, the client may or may not supply an SD value. If the SD value is not supplied, then the DACL and SACL on the newly created object are defaulted according to the SD defaulting rules (see SD defaulting section below). If the SD value is present, then the DACL and SACL are obtained from this value. If the DACL is not present in the supplied value, then the add operation is failed with *unwillingToPerform* (per constraint above). If the SACL is not present in the supplied value, then NULL value is written in place of this SACL.
7. If RM control field is present in the supplied SD value, then its value is reset to contain SECURITY\_PRIVATE\_OBJECT bit, and nothing else.
8. AD/LDS imposes a restriction on the **security principals** that can be used in an AD/LDS security descriptor (owner, group, and SID values within ACEs). The SID of a security principal within an AD/LDS application NC can appear in a security descriptor within that application NC, but cannot appear in a security descriptor within any other NC of the same forest. Other SIDs are not restricted, so for instance a Windows security principal is allowed in any AD/LDS security descriptor, as is a security principal from another AD/LDS forest, as is a security principal from the config NC of the same AD/LDS forest.

#### 7.1.3.4 Security Considerations

When an add operation is processed, the client is allowed to specify any SD value, subject to some constraints to the OWNER field, specified below.

When a modify operation is processed, the following security checks are applied to the requestor's security context. If the requestor does not pass the check, then *accessDenied* is returned.

1. If the DACL value is written (according to SD flags), then one of the following requirements must be satisfied:
  1. WRITE\_DAC is granted to the requestor on the object;

2. The OWNER SID in the SD value is one of the SIDs in requestor's token (either as user SID or group SID).
2. If the OWNER and/or GROUP value is written (according to SD flags), then one of the following requirements must be satisfied:
  1. WRITE\_OWNER is granted to the requestor on the object;
  2. The requestor possesses the SE\_TAKE\_OWNERSHIP\_PRIVILEGE.
3. If the SACL value is written (according to SD flags), then the following requirement must be satisfied:
  - The requestor possesses the SE\_SECURITY\_PRIVILEGE.
4. If the object being modified is in config NC or schema NC, and the RM control of the SD is present and contains SECURITY\_PRIVATE\_OBJECT bit, then additional requirements on the DC performing the operation must be enforced:
  1. The DC must be a member of the root **domain** in the forest, or
  2. The DC must be a member of the same **domain** to which the current object owner belongs.

When the OWNER value is being written (via SD flags control, either in add or modify operation), then the following constraint must be satisfied. The value of the OWNER field must be one of the following SIDs:

1. The SID of the user performing the operation;
2. The SID of the "default administrators group" (DAG, see definition below), only when DAG is defined, and the user is a member of this group;
3. Any SID, when the user possesses the SE\_RESTORE\_PRIVILEGE.

If the owner SID does not satisfy the rules above, then the server fails the operation, returning *unwillingToPerform* / *ERROR\_INVALID\_OWNER* error.

If the owner SID is written on an object in config NC or schema NC, then additional requirements on the DC performing the operation are enforced:

- The DC must be a member of the root **domain** in the forest, or
- The DC must be a member of the same **domain** where the current object owner belongs.

### 7.1.3.5 DACL and SACL Defaulting Rules

When an add operation is performed and the client does not supply an SD value, then the SD value is defaulted as follows:

1. The DACL and SACL are determined from the [defaultSecurityDescriptor](#) value obtained from the [classSchema](#) object corresponding to the most specific structural [objectClass](#) of the object being created. The value of [defaultSecurityDescriptor](#) is an SDDL string. The string is converted to the binary SD value in the context of **domain** SID (used to resolve **domain** SID references, such as Domain Administrators alias) and root **domain** SID (used to resolve forest SID references, such as Enterprise Administrators alias). See [\[MS-DTYP\]](#) section 2.5.1, for more details.

2. When the object is created in an application NC, then the value or sdReferenceDomain from the [crossRef](#) corresponding to the NC is used to determine the **domain** SID used as context in SDDL conversion process.

### 7.1.3.6 Owner and Group Defaulting Rules

The OWNER and GROUP fields are defaulted in following scenarios:

- An add operation is performed and no SD value is provided.
- An add operation is performed and SD value is provided, but the SD flags do not include OWNER bit.
- An add or modify operation is performed and SD value is provided, and SD flags include the OWNER bit, but the OWNER field in the supplied value is NULL.

In the above cases, the OWNER field is defaulted as follows:

- If the user performing the operation is a member of the DAG for the object (when it is defined), the SID of this group is written into the OWNER field of the SD.
- Otherwise, if the requestor's security context contains TokenOwner field, then the SID contained in this field is written into the OWNER field of the SD.
- Otherwise, the requestor's user SID is written into the OWNER field of the SD.

The GROUP field is defaulted as follows:

- If the DAG was used as the default OWNER field value, then the same SID is written into the GROUP field.
- Otherwise, the GROUP field is defaulted to NULL.

### 7.1.3.7 Default Administrators Group

The "default administrators group" (DAG), which is used for OWNER/GROUP defaulting and also in OWNER write access checks, is computed based on two inputs: the contents of the requestor's token and the location of the object whose SD is being written. The following rules are applied (in order):

1. When the object belongs to a **domain** NC:
  1. If the user is a member of Domain Admins for this **domain**, then Domain Admins is designated as the DAG.
  2. If the user is a member of Enterprise Admins for the forest, then Enterprise Admins is designated as the DAG.
  3. Otherwise, the DAG is undefined.
2. When the object belongs to config NC:
  1. If the user is a member of Enterprise Admins, then Enterprise Admins is designated as the DAG.
  2. If the user is a member of Domain Admins (for the **domain** that the current DC belongs to), then this Domain Admins group is designated as the DAG.
  3. Otherwise, the DAG is undefined.

3. When the object belongs to schema NC:
  1. If the user is a member of Schema Admins, then Schema Admins is designated as the DAG.
  2. If the user is a member of Enterprise Admins, then Enterprise Admins is designated as the DAG.
  3. If the user is a member of Domain Admins (for the **domain** that the current DC belongs to), then this Domain Admins group is designated as the DAG.
  4. Otherwise, the DAG is undefined.
4. When the object belongs to an application NC:
  1. If the user is a member of Domain Admins for the **domain** that is designated as sdReferenceDomain for this application NC, then this Domain Admins group is designated as the DAG
  2. If the user is a member of Enterprise Admins, then Enterprise Admins is designated as the DAG.
  3. Otherwise, the DAG is undefined.

#### 7.1.4 Special Attributes

Glossary Terms: FSMO Role, PDC FSMO Role Owner

LDAP Attributes: [nTMixedDomain](#), [msDS-Behavior-Version](#)

LDAP Classes: [nTDSDSA](#), [crossRef](#)

Constants: [crossRefContainer](#)

##### 7.1.4.1 ntMixedDomain

The attribute [nTMixedDomain](#) is present on each domain NC root object. The value 1 indicates a domain that is in **mixed mode** and that supports replication to NT backup domain controllers ([\[MS-NRPC\]](#)). The value 0 indicates a domain that does not support such replication.

If the value of [nTMixedDomain](#) is 0, it cannot be changed.

The attribute [nTMixedDomain](#) on a [crossRef](#) object is read-only and equals the attribute [nTMixedDomain](#) on the corresponding domain NC root object.

If there are Windows Server 2008 DCs in the domain, [nTMixedDomain](#) MUST be 0. This implies that Windows Server 2008 DCs cannot be used in a domain that is in mixed mode.

##### 7.1.4.2 msDS-Behavior-Version: DC Functional Level

The [msDS-Behavior-Version](#) attribute is written on the [nTDSDSA](#) object representing a DC. The value is the highest **domain** or forest functional level the DC is capable of supporting. A DC supports any **domain** or forest functional level less than or equal to its [msDS-Behavior-Version](#).

The value of the [msDS-Behavior-Version](#) attribute on an [nTDSDSA](#) object changes during OS upgrade of that DC. The value of the [msDS-Behavior-Version](#) attribute never decreases.

The [msDS-Behavior-Version](#) attribute being absent on an [nTDSDSA](#) object is equivalent to the [msDS-Behavior-Version](#) attribute on that object having the value zero.

The following values are defined:

Identifier	Value
DS_BEHAVIOR_WIN2000	0
DS_BEHAVIOR_WIN2003	2
DS_BEHAVIOR_WIN2008	3

#### 7.1.4.3 msDS-Behavior-Version: Domain NC Functional Level

The [msDS-Behavior-Version](#) for **domains** is written on both the **domain** NC root object and also the [crossRef](#) representing the **domain**. The attribute on the [crossRef](#) is read-only and is kept in sync with the attribute on the **domain** NC root object. Only the PDC FSMO role owner accepts originating updates to the attribute on the **domain** NC root.

Invariants: The functional level of a **domain** is never larger than any **domain** DCs functional level that hosts or is instructed to host (see section [7.1.2.3](#)) the **domain** NC. When the functional level of a **domain** is DS\_BEHAVIOR\_WIN2003 or greater, the attribute [nTMixedDomain](#) on the **domain** NC root is 0 (see section [7.1.4.1](#)). The functional level of a **domain** never decreases.

The [msDS-Behavior-Version](#) attribute being absent on a **domain** NC root object is equivalent to the [msDS-Behavior-Version](#) attribute on that object having the value zero.

The following values are defined:

Identifier	Value
DS_BEHAVIOR_WIN2000	0
DS_BEHAVIOR_WIN2003_WITH_MIXED_DOMAINS	1
DS_BEHAVIOR_WIN2003	2
DS_BEHAVIOR_WIN2008	3

#### 7.1.4.4 msDS-Behavior-Version: Forest Functional Level

The [msDS-Behavior-Version](#) for the forest is written on the [crossRefContainer](#) object, see section [7.1.1.2.1](#). Only the Partition Naming Master role FSMO owner accepts updates to this attribute.

Invariants: The value of [msDS-Behavior-Version](#) for the forest is never larger than any functional level of any **domain** NC in the forest. This value can never decrease.

The [msDS-Behavior-Version](#) attribute being absent on a [crossRefContainer](#) object is equivalent to the [msDS-Behavior-Version](#) attribute on that object having the value zero.

The following values are defined:

Identifier	Value
DS_BEHAVIOR_WIN2000	0

Identifier	Value
DS_BEHAVIOR_WIN2003_WITH_MIXED_DOMAINS	1
DS_BEHAVIOR_WIN2003	2
DS_BEHAVIOR_WIN2008	3

### 7.1.4.5 Replication Schedule Structures

#### 7.1.4.5.1 SCHEDULE\_HEADER Structure

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type																															
Offset																															

**Type:** This must be SCHEDULE\_INTERVAL (0).

**Offset:** An offset into the Data field of the SCHEDULE structure. The offset represents the start of the replication schedule data.

#### 7.1.4.5.2 SCHEDULE Structure

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Size																															
Bandwidth																															
NumberOfSchedules																															
Schedules																															
...																															
Data (variable)																															
...																															

**Size:** Size of the entire replication schedule structure.

**Bandwidth:** Not used; this field is ignored.

**NumberOfSchedules:** Number of elements in Schedules.



This value is always 1.

**Schedules:** Array of [SCHEDULE\\_HEADER](#) structures.

There is only one SCHEDULE\_HEADER element in the array.

**Data:** This is a sequence of bytes specifying the time slots when replication is permitted between the source and destination DC. Each schedule header specifies an offset into the data field. The replication schedule data for that schedule is the next 168 bytes. Each byte represents an hour in the week ( $24 * 7 = 168$ ). The lower 4 bits of each byte represent 15 minute intervals in the hour. The first bit corresponds to the first 15 minutes in the hour, the second bit corresponds to the next 15 minutes and so on. If one of these bits is set, it indicates that replication is permitted in that 15 minute time interval within that hour.

The offset field of the SCHEDULE\_HEADER structure points to the beginning of the Data field and the Data field is exactly 168 bytes since there is only one schedule.

#### 7.1.4.5.3 REPS\_FROM

Specified in [MS-DRSR] section [5:REPS\\_FROM](#).

#### 7.1.4.5.4 REPS\_TO

Specified in [MS-DRSR] section [5:REPS\\_TO](#).

#### 7.1.4.5.5 MTX\_ADDRESS Structure

Specified in [MS-DRSR] section [5:MTX\\_ADDRESS](#).

#### 7.1.4.5.6 REPLTIMES Structure

Specified in [MS-DRSR] section [5:REPLTIMES](#).

#### 7.1.4.5.7 PAS\_DATA Structure

Specified in [MS-DRSR] section [5:PAS\\_DATA](#).

#### 7.1.4.6 msDS-AuthenticatedAtDC

This attribute is maintained by the DC on [user](#) and [computer](#) objects. The attribute contains a list of [computer](#) objects, corresponding to the RODCs at which the [user](#) or [computer](#) has authenticated. This attribute is a forward link attribute whose corresponding back link is the [msDS-AuthenticatedToAccountlist](#) attribute (see section [7.1.1.3.2](#)). When a writable DC authenticates a [user](#) or [computer](#) to an RODC, that writable DC adds the DN of the RODC's [computer](#) object to the list in the [msDS-AuthenticatedAtDC](#) attribute of the [user](#) or [computer](#) that was authenticated.

DCs prior to Windows Server 2008 do not maintain this attribute.

### 7.1.5 FSMO Roles

References: SID, RID, RID Allocation, RID Master role in inter-domain move, PDC Emulator role, Infrastructure role

Functions: RoleObject, GetRoleScope

Glossary Terms: FSMO Role, NC Replica, DC, SID

Ldap Attributes: [fSMORoleOwner](#)

Ldap Classes: [nTDSDSA](#)

An FSMO role is defined as a set of objects that may be updated in only one NC replica at any given time. The DC that hosts this NC replica is the Owner for that FSMO role.

Each FSMO role is represented by an object in the directory. The function RoleObject (section [3.1.1.5.1.7](#)) specifies the object for a given FSMO role type and NC. This object is an element of the FSMO role and contains the [fSMORoleOwner](#) attribute, which references the [nTDSDSA](#) object of the DC that owns the role. The function GetRoleScope defined in [MS-DRSR] section [4.1.10.5.13](#) identifies the set of objects that comprise each FSMO role. These objects must be updated only on the DC currently owning the FSMO role.

The following FSMO role exists in the directory:

### Schema Master FSMO Role

The Schema Master FSMO role owner is the DC responsible for performing updates to the directory schema. This DC is the only one that can process updates to the directory schema. Once the Schema update is complete, it is replicated from the Schema Master FSMO role owner to all other DCs in the directory. There is only one Schema Master FSMO role per forest.

#### 7.1.5.1 Domain Naming Master FSMO Role

The Domain Naming Master FSMO role owner is the DC responsible for making changes to the forest-wide **domain** name space of the directory in the partitions **container**. This DC is the only one that can add or remove a **domain** or application NC from the directory. It can also add or remove cross references to **domains** in external directories. Only the Domain Naming Master FSMO role owner can write to the partitions **container** or its children. There is only one Domain Naming Master FSMO role per forest.

#### 7.1.5.2 RID Master FSMO Role

The RID Master FSMO role owner is the single DC responsible for processing RID Pool requests from all DCs within a given **domain**. It is also responsible for moving an object from one **domain** to another during an interdomain object move.

When a DC creates a security principal object such as a user or group, it attaches a unique Security ID (SID) to the object. This SID consists of a **domain** SID (the same for all SIDs created in a **domain**), and a relative ID (RID) that is unique for each security principal SID created in a **domain**.

Each DC in a **domain** is allocated a pool of RIDs that it is allowed to assign to the **security principals** it creates. When a DC's allocated RID pool falls below a threshold, that DC issues a request for additional RIDs to the **domain's** RID Master FSMO role owner (see [MS-DRSR] section [4.1.10.4.3](#), PerformExtendedOpRequestMsg with ulExtendedOp = EXOP\_FSMO\_RID\_REQ\_ROLE). The RID Master FSMO role owner responds to the request by retrieving RIDs from the **domain's** unallocated RID pool and assigns them to the pool of the requesting DC (see [MS-DRSR] section [4.1.10.5.11](#), ProcessFsmoRoleRequest with ulExtendedOp = EXOP\_FSMO\_RID\_REQ\_ROLE). There is one RID Master FSMO role per **domain** in a directory.

See section [3.1.1.5](#) for more information about the RID Master's role in interdomain object move operations.

### 7.1.5.3 PDC Emulator FSMO Role

The PDC Emulator FSMO role owner performs the following functions:

- Password changes performed by other DCs in the **domain** are replicated preferentially to the PDC emulator.
- Authentication failures that occur at a given DC in a **domain** because of an incorrect password are forwarded to the PDC emulator before a bad password failure message is reported to the user.
- Account lockout is processed on the PDC emulator.
- The PDC emulator performs all of the functionality that a Microsoft Windows NT 4.0 Server-based PDC or earlier PDC performs for Windows NT 4.0-based or earlier clients. Therefore, all functionality required from Windows NT 4.0-based or earlier clients must be performed by the PDC emulator, and not by any other DC.

There is one PDC Emulator FSMO role per **domain** in a directory. See [3.1.1.7](#) for more information about the PDC Emulator FSMO role.

### 7.1.5.4 Infrastructure FSMO Role

When an object in one **domain** is referenced by another object in another **domain**, it represents the reference as a dsname. The Infrastructure FSMO role owner is the DC responsible for updating a cross-domain object reference in the event that the referenced object is moved, renamed or deleted. There is one Infrastructure FSMO role per **domain** and application NC in a directory.

The Infrastructure Master (IM) role should be held by a domain controller that is not a GC server. If the Infrastructure Master runs on a GC server it will not update object information because it does not contain any references to objects that it does not hold. This is because a GC server holds a partial replica of every object in the forest.

If all the domain controllers in a **domain** also host the GC, all the domain controllers have the current data, and it is not important which domain controller owns the infrastructure master role. See [3.1.1.5](#) for more information about the infrastructure master.

## 7.1.6 Trust Objects

### 7.1.6.1 Overview (Synopsis)

Active Directory (AD) **domains** rarely live in isolation. Many AD deployments in customer sites consist of two or more **domains** that represent boundaries between different geographical, managerial, organizational or administrative layouts. For example, when company "A" acquires company "B," it quickly becomes necessary for preexisting **domains** for each to start trusting each other. Alternately, in some deployments, servers having a specific role (such as a mail server) may be members of a "resource domain", easing the management burden by combining like roles under one administrative domain.

Enabling communication between disparate **domains**, especially secure communication involving **authentication** and authorization, requires that some stateful knowledge be shared between the peer **domains** in order for them to trust one another. Some of this knowledge is sensitive, forming the cryptographic basis of trust mechanisms used in protocols such as Kerberos and **Netlogon RPC**. Other state is public knowledge, such as the NetBIOS name of a peer **domain**, or which security identifiers are owned by the peer **domain**. Information like this plays a crucial role when performing

name lookups, which are essential for authorization, locating user accounts, or simply displaying information in some sort of User Interface.

Active Directory stores trust information in **trusted domain objects (TDOs)**. This section of the document details the contents of TDOs, focusing on analysis of the properties that are specific to TDOs, and are essential for proper interdomain functionality.

### 7.1.6.2 Relationship to Other Protocols

#### 7.1.6.2.1 TDO Replication over DRS

Once they are created, TDOs are replicated along with other objects over replication protocols ([\[MS-DRSR\]](#) and [\[MS-SRPL\]](#)). In this manner, they are no different than any other Directory Service object.

#### 7.1.6.2.2 TDO Roles in Authentication Protocols over Domain Boundaries

For most network **authentication** protocols, if a client wishes to securely authenticate to a service residing in a foreign **domain**, it becomes necessary for the client and service **domains** to have some form of trust. Most trust systems in use today rely upon some form of key for trust validation.

TDOs play an important part in the storage and distribution of keys used for trust validation between **domains**. Commonly used Windows network **authentication** mechanisms such as Kerberos ([\[RFC4120\]](#) section 1.1) retrieve keys from TDOs which have been established between the client and service **domains**. Additionally, services using other protocols such as NTLM, **Digest**, and SSL Certificate Mapping use the Generic Passthrough Mechanism over the Netlogon Secure Channel Protocol [\[MS-NRPC\]](#) to authenticate users from foreign **domains**. Establishing the Netlogon Secure Channel requires the use of key information contained in TDOs. The format and storage locations for these keys will be discussed below (section [7.1.6.8.1](#)), including information on the usage for relevant **authentication** protocols.

#### 7.1.6.2.3 TDO Roles in Authorization over Domain Boundaries

In some configurations authorization data from a trusted **domain**, such as a SID ([\[MS-DTYP\]](#) section **2.4.2**) or a client name in a Kerberos Cross Realm Ticket Granting Ticket ([\[RFC4120\]](#) section 5.3), must be scrutinized to protect against attempts in the foreign **domain** from claiming identities from within the local **domain**. For example, if the foreign DC were to become compromised by an attacker, without these protections it would be possible to inject the SID of the local domain administrator into the transferred TGT. This would have the end result of granting the attacker domain administrator rights in the local **domain**.

To protect against these attacks TDOs contain name spaces and SID spaces which legitimately belong to the foreign **domain**. When enabled, **authentication** protocols will use this information to verify that authorization data passed through the protocol is valid for the trust. If a SID or name within the authorization data does not correspond to those claimed within the **TDO**, the request is rejected. This can cause network logon attempts to fail, or may alternately cause Kerberos ticket requests to fail, as discussed in [\[MS-PAC\]](#) section [4.2.3](#).

#### 7.1.6.2.4 Roles in Name Lookup and Routing Decisions

Both the Kerberos and Netlogon Generic Passthrough protocols use TDOs to locate foreign realms as part of their normal processing. For example, if a Kerberos service ticket request is destined for realm "example.com", then a lookup is performed on the name attributes from the registered TDOs to determine which trust should be traversed to reach "example.com." Netlogon Generic

Passthrough requests also contain information about the destination **domain**. Again, proper routing is achieved through the use of TDOs.

### 7.1.6.3 Prerequisites/Preconditions

TDOs are only used for storing trust information on Windows 2000 and beyond operating systems.

### 7.1.6.4 Versioning and Capability Negotiation

- Building TDOs representing **cross forest trusts** require that both the **domain** and the forest are running in a **domain** and forest functional level of DS\_BEHAVIOR\_WIN2003 or greater.
- An **uplevel trust**, by definition, is one in which both trusting **domains** are running all Windows 2000 or newer DCs.
- A **downlevel trust** is one in which either of the trusting **domains** are running Windows NT 4.0 DCs.

### 7.1.6.5 Vendor-Extensible Fields

- It is possible to store provider specific values in the trustAuthInfoOutgoing and the trustAuthInfoIncoming attributes [\[MS-ADA3\]](#) on a **TDO**. See the sections on **TDO** keys (section [7.1.6.8.1](#)), and trustAuthInfoIncoming (section [7.1.6.7.10](#)) for details on the range of extensible values.

### 7.1.6.6 Transport

TDOs are replicated along with other DS objects, as described in [\[MS-DRSR\]](#) and [\[MS-SRPL\]](#).

### 7.1.6.7 Essential Attributes of a Trusted Domain Object

TDOs are stored in the System **container**, with a CN representing the FQDN of the trusted **domain**. For example, if a.example.com trusts b.example.com, an object would be created in the System **container** with a CN of b.example.com. The system **container** can be found by using the function GetWellknownObject( NC, default NC, GUID\_SYSTEM\_CONTAINER\_W). For more information, see section [3.1.1.1](#).

The contents of TDOs are described by the [trustedDomain](#) schema object [\[MS-ADSC\]](#). The table below details those attributes that are essential to a well functioning interdomain trust, with links to specific sections detailing their relevance and format when these attributes are present.

Attribute Name	Reference
<a href="#">flatName</a>	<a href="#">MS-ADA1</a>
<a href="#">isCriticalSystemObject</a>	<a href="#">MS-ADA1</a>
<a href="#">msDS-SupportedEncryptionTypes</a>	<a href="#">MS-ADA2</a> , <a href="#">7.1.6.8.1</a>
<a href="#">msDS-TrustForestTrustInfo</a>	<a href="#">MS-ADA2</a> , <a href="#">7.1.6.8.2</a>
<a href="#">nTSecurityDescriptor</a>	<a href="#">MS-ADA3</a>

Attribute Name	Reference
<a href="#">objectCategory</a>	<a href="#">MS-ADA3</a>
<a href="#">objectClass</a>	<a href="#">MS-ADA3</a>
<a href="#">securityIdentifier</a>	<a href="#">MS-ADA3</a>
<a href="#">trustAttributes</a>	<a href="#">MS-ADA3</a>
<a href="#">trustAuthIncoming</a>	<a href="#">MS-ADA3</a> , <a href="#">7.1.6.8.1</a>
<a href="#">trustAuthOutgoing</a>	<a href="#">MS-ADA3</a> , <a href="#">7.1.6.8.1</a>
<a href="#">trustDirection</a>	<a href="#">MS-ADA3</a>
<a href="#">trustPartner</a>	<a href="#">MS-ADA3</a>
<a href="#">trustPosixOffset</a>	<a href="#">MS-ADA3</a> , <a href="#">7.1.6.8.3</a>
<a href="#">trustType</a>	<a href="#">MS-ADA3</a>

#### 7.1.6.7.1 flatName

The [flatName](#) attribute contains the NetBIOS name (as specified in [\[RFC1088\]](#)) of the trusted **domain** in String(Unicode) syntax.

This attribute is unique on all TDOs within the **domain**. The system rejects attempts to create a duplicate value.

#### 7.1.6.7.2 isCriticalSystemObject

Mandatory Boolean attribute. Always set to true for TDOs, which indicates it must be replicated when a new replica is installed.

#### 7.1.6.7.3 msDs-supportedEncryptionTypes

Windows Server 2008 only.

Contains bitmapped values that define the encryption types supported by this trust relationship. One or more of the following flags can be set. Unused flags should be set to 0 when writing the attribute and should be ignored when reading the attribute. The flags are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	A	R	M	C
																										2	1	C	D	R	
																										5	2	4	5	C	
																										6	8				

#### **CRC (KERB\_ENCTYPE\_DES\_CBC\_CRC, 0x00000001)**

Supports CRC32 as described in [\[RFC3961\]](#) page 31.

#### **MD5 (KERB\_ENCTYPE\_DES\_CBC\_MD5, 0x00000002)**

Supports RSA-MD5 as described in [\[RFC3961\]](#) page 31.

#### **RC4 (KERB\_ENCTYPE\_RC4\_HMAC\_MD5, 0x00000004)**

Supports RC4-HMAC-MD5 as described in [RFC-DRAFT-RC4-HMAC].

#### **A128 (KERB\_ENCTYPE\_AES128\_CTS\_HMAC\_SHA1\_96, 0x00000008)**

Supports HMAC-SHA1-96-AES128 as described in [\[RFC3961\]](#) page 31.

#### **A256 (KERB\_ENCTYPE\_AES256\_CTS\_HMAC\_SHA1\_96, 0x00000010)**

Supports HMAC-SHA1-96-AES256 as described in [\[RFC3961\]](#) page 31.

### **7.1.6.7.4 msDS-TrustForestTrustInfo**

Windows Server 2003 Only

The contents of this attribute are fully specified below in section [7.1.6.8.2](#).

### **7.1.6.7.5 nTSecurityDescriptor**

Mandatory object attribute that contains the security descriptor tied to the Active Directory object. The security descriptor mandates access controls to the object. TDOs are sensitive objects, and have tight access controls placed upon them. Stored as the type String(NT-Sec-Desc) in SDDL ([\[MS-DTYP\]](#) section 2.5.1), the default security descriptor for TDOs is as follows:

Platforms	Default Security Descriptor in SDDL Format
W2000	D: (A;;RPWPCRCDCCLCLORCWOWSDDTSW;;;DA) (A;;RPWPCRCDCCLCLORCWOWSDDTSW;;;SY) (A;;RPLCLORC;;;AU)
W2003 W2003R2	D: (A;;RPWPCRCDCCLCLORCWOWSDDTSW;;;DA) (A;;RPWPCRCDCCLCLORCWOWSDDTSW;;;SY) (A;;RPLCLORC;;;AU) (OA;;WP;736e4812-af31-11d2-b7df-00805f48caeb;bf967ab8-0de6-11d0-a285-00aa003049e2;CO) (A;;SD;;;CO)

### 7.1.6.7.6 objectCategory

This is a mandatory attribute representing the schema definition for TDOs. The value is a reference to the [classSchema](#) object for the [trustedDomain](#) class.

### 7.1.6.7.7 objectClass

Mandatory multi-valued attribute representing the classes the target object is derived from. For a **TDO**, this value contains [[top](#), [leaf](#), [trustedDomain](#) ].

### 7.1.6.7.8 securityIdentifier

The [securityIdentifier](#) attribute contains an String(Octet) representation of the SID belonging to the trusted **domain**. This value contains the **domain** relative SID ([\[MS-DTYP\]](#) section **2.4.2**) of identities issued by the trusted **domain**. For example, for "example.com", a trusted **domain**, the value may be S-1-5-2223345-6677. The domain administrator for example.com would have a SID of S-1-5-2223345-6677-512.

This attribute is unique on all TDOs within the **domain**. The system rejects attempts to create a duplicate value.

### 7.1.6.7.9 trustAttributes

The [trustAttributes](#) attribute contains the value of a trust relationship. This value corresponds to the **TrustAttributes** field detailed in the LSAPR\_TRUSTED\_DOMAIN\_INFORMATION\_EX structure ([\[MS-LSAD\]](#) section **2.2.52**). The flags in the following diagram are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	T	T	T	T	T	T	T	T
																							A	A	A	A	A	A	A	A	
																							R	T	W	C	F	Q	U	N	
																							C	E	F	O	T	D	O	T	

Name and Value	Restrictions / Special Notes
TANT (TRUST_ATTRIBUTE_NON_TRANSITIVE) 0x00000001	
TAUO (TRUST_ATTRIBUTE_UPLEVEL_ONLY) 0x00000002	Only present on Windows 2000, Windows Server 2003, R2 trusts.
TAQD (TRUST_ATTRIBUTE_QUARANTINED_DOMAIN) 0x00000004	



Name and Value	Restrictions / Special Notes
TAFT (TRUST_ATTRIBUTE_FOREST_TRANSITIVE) 0x00000008	Only evaluated on Windows Server 2003. Only settable if forest and trusted forest are running in a forest functional level of DS_BEHAVIOR_WIN2003 or greater.
TACO (TRUST_ATTRIBUTE_CROSS_ORGANIZATION) 0x00000010	Only evaluated on Windows Server 2003. Only settable if forest and trusted forest are running in a forest functional level of DS_BEHAVIOR_WIN2003 or greater.
TAWF (TRUST_ATTRIBUTE_WITHIN_FOREST) 0x00000020	
TATE (TRUST_ATTRIBUTE_TREAT_AS_EXTERNAL) 0x00000040	Only evaluated on Windows Server 2003. Only evaluated if SID Filtering is used. Only evaluated on cross forest trusts having TRUST_ATTRIBUTE_FOREST_TRANSITIVE. Only settable if forest and trusted forest are running in a forest functional level of DS_BEHAVIOR_WIN2003 or greater.
TARC (TRUST_ATTRIBUTE_USES_RC4_ENCRYPTION) 0x00000080	Only evaluated on TRUST_TYPE_MIT
Reserved for future use 0x00000100 - 0x00200000	
Deprecated 0x00400000 - 0x00800000	Previously used trust bits, deprecated.
Reserved for use in User Trust Attributes 0x01000000 - 0x80000000	

#### **TRUST\_ATTRIBUTE\_NON\_TRANSITIVE (0x00000001)**

If this bit is set, then the trust cannot be used transitively. For example, if **domain** A trusts **domain** B, which in turn trusts **domain** C, and the A<-->B trust has this attribute set, then a client in **domain** A cannot authenticate to a server in **domain** C over the A<-->B<-->C trust linkage.

#### **TRUST\_ATTRIBUTE\_UPLEVEL\_ONLY (0x00000002)**

If this bit is set in the attribute, then only Windows 2000 and newer clients may use the trust link. This bit is not enforced for **trust objects**.

#### **TRUST\_ATTRIBUTE\_QUARANTINED\_DOMAIN (0x00000004)**

If this bit is set, the peer **domain** is quarantined, and is subject to the rules of SID Filtering as described in [MS-PAC] section [4.2.2](#).

#### **TRUST\_ATTRIBUTE\_FOREST\_TRANSITIVE (0x00000008)**

If this bit is set, the trust link is a cross forest trust ([\[MS-KILE\]](#)) between the root **domains** of two forests, both of which are running in a forest functional level of DS\_BEHAVIOR\_WIN2003 or greater.

#### **TRUST\_ATTRIBUTE\_CROSS\_ORGANIZATION (0x00000010)**

If this bit is set, then the trust is to a **domain** or forest which is not part of the Enterprise. If this bit is present, then **authentication** attempts across the **domain** boundary are subject to the Authentication Firewall ([\[MS-KILE\]](#)).

#### **TRUST\_ATTRIBUTE\_WITHIN\_FOREST (0x00000020)**

If this bit is set, then the trusted peer **domain** is within the same forest.

#### **TRUST\_ATTRIBUTE\_TREAT\_AS\_EXTERNAL (0x00000040)**

If this bit is set, then a cross forest trust to a peer **domain** is to be treated as an external trust for the purposes of SID Filtering. Cross forest trusts are more stringently filtered than external trusts. This attribute relaxes those cross forest trusts to be equivalent to external trusts. For more information on how each trust type is filtered, see [\[MS-PAC\] section 4.2.2](#).

#### **TRUST\_ATTRIBUTE\_USES\_RC4\_ENCRYPTION (0x00000080)**

This bit is set on trusts with the [trustType](#) set to TRUST\_TYPE\_MIT, which are capable of using RC4 keys. Historically, MIT Kerberos distributions supported only DES and 3DES keys ([\[RFC4120\]](#), [\[RFC3961\]](#)). MIT 1.4.1 adopted the RC4HMAC encryption type common to Windows 2000 ([\[MS-KILE\]](#)), so trusted **domains** deploying later versions of the MIT distribution required this bit. For more information see the section on [Keys and Trusts \(section 7.1.6.8.1\)](#).

There are some byte ranges which are not valid for this field.

Range	Description
0x00000100 - 0x00200000	Reserved for future use
0x00400000 - 0x00800000	Previously used, deprecated, and are not used
0x01000000 - 0x80000000	Reserved for vendor-specified trust attributes

### **7.1.6.7.10 trustAuthIncoming**

This is a String(Octet) attribute. This value is used to compute keys used in **inbound trust** validation. For more information on the contents of this attribute, see the section on [Keys and Trusts \(section 7.1.6.8.1\)](#).

This is a Secret Attribute ([\[MS-DRSR\] section 4.1.10.3.12](#), IsSecretAttribute), and is not readable outside of the context of the LSA on a DC.

### **7.1.6.7.11 trustAuthOutgoing**

This is a String(Octet) attribute. This value is used to compute keys used in **outbound trust** validation. For more information on the contents of this attribute, see the section on [Keys and Trusts \(section 7.1.6.8.1\)](#).

This is a Secret Attribute ([\[MS-DRSR\] section 4.1.10.3.12](#), IsSecretAttribute), and is not readable outside of the context of the LSA on a DC.

### 7.1.6.7.12 trustDirection

The [trustDirection](#) attribute dictates in which direction the trust flows. It is stored as an Integer value. There are 4 valid values, corresponding to the **TrustDirection** field in the LSAPR\_TRUSTED\_DOMAIN\_INFORMATION\_EX structure ([\[MS-LSAD\]](#) section **2.2.52**). The flags in the following diagram are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	T	T
																													D	D	
																													O	I	

#### TRUST\_DIRECTION\_DISABLED 0x00000000

Absence of any flags. The trust relationship exists, but has been disabled.

#### TDI (TRUST\_DIRECTION\_INBOUND) 0x00000001

The trusted **domain** trusts the primary **domain** to perform operations such as name lookups and **authentication**. If this flag is set, then the [trustAuthIncoming](#) attribute is present on this object.

#### TDO (TRUST\_DIRECTION\_OUTBOUND) 0x00000002

The primary **domain** trusts the trusted **domain** to perform operations such as name lookups and **authentication**. If this flag is set, then the [trustAuthOutgoing](#) attribute is present on this object.

#### TRUST\_DIRECTION\_BIDIRECTIONAL 0x00000003

ORing of above flags and behaviors representing that both **domains** trust one another for operations such as name lookups and **authentication**.

### 7.1.6.7.13 trustPartner

This String(**Unicode**) attribute contains the DNS Domain Name of the trusted **domain**.

As with the [securityIdentifier](#) attribute, this attribute is unique on all TDOs within the **domain**. The system rejects attempts to create a duplicate value.

### 7.1.6.7.14 trustPosixOffset

This Integer value contains the Portable Operating System Interface (POSIX) offset for the trusted **domain**, as described in [\[MSDN-POSIX-OFFSET\]](#). This value is added to the RID of a SID to give the POSIX user ID or group ID (as specified in [\[IEEE1003.1\]](#)) for that user in the trusted **domain**. The calculation of this value is documented in section [7.1.6.8.3](#).

### 7.1.6.7.15 trustType

The [trustType](#) attribute is an Integer value that dictates what type of trust has been designated for the trusted **domain**. Below are the valid values, corresponding to the **TrustType** field in LSAPR\_TRUSTED\_DOMAIN\_INFORMATION\_EX, as specified in [\[MS-LSAD\]](#) section **2.2.52**. The [trustType](#) contains one of the values below:

### **TTD (TRUST\_TYPE\_DOWNLEVEL) (0x00000001)**

The trusted **domain** is running Windows NT 4.0.

### **TTU (TRUST\_TYPE\_UPLEVEL) (0x00000002)**

The trusted **domain** is running Windows 2000 or newer directory servers.

### **TTM (TRUST\_TYPE\_MIT) (0x00000003)**

The trusted **domain** is running a non-Windows RFC4120 compliant Kerberos distribution. This type of trust is distinguished in that (1) a SID is not required for the **TDO**, and (2) the default key types include the DES-CBC and DES-CRC encryption types (see [\[RFC4120\]](#) section 8.1).

### **TTDCE (TRUST\_TYPE\_DCE) (0x00000004)**

Historical reference - not used in Windows.

## **7.1.6.8 Details**

### **7.1.6.8.1 trustAuthInfo attributes - TDO Keys and Trusts**

Domain peers share a password in order to validate protocol messages flowing between the trusted **domains**. The password is only good in one direction of the trust. Each direction is stored in its own attribute: the [trustAuthIncoming](#) and [trustAuthOutgoing](#) attributes. These are both Secret Attributes ([MS-DRSR] section [4.1.10.3.12](#), IsSecretAttribute), and are not readable outside of the context of the LSA on a DC.

Both the [trustAuthIncoming](#) and the [trustAuthOutgoing](#) is stored as a String(Octet). The storage of this information in a **TDO** is described in the diagrams below:

```
struct LSAPR_TRUST_DOMAIN_AUTH_INFO_HALF {
    ULONG AuthInfos;
    PLSAPR_AUTH_INFORMATION AuthenticationInformation;
    PLSAPR_AUTH_INFORMATION PreviousAuthenticationInformation;
}
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Count of auth infos																															
Byte offset to AuthenticationInformation																															
Byte offset to PreviousAuthenticationInformation																															
AuthenticationInformation (variable)																															
...																															
PreviousAuthenticationInformation (variable)																															
...																															

**Count of auth infos:** A ULONG count of the pairs of LSAPR\_AUTH\_INFORMATION structures. Each current Authentication Information structure is accompanied by a previous Authentication Information structure (even if it is marked as invalid), and the count of the pairs of elements is stored in this field.

**Byte offset to AuthenticationInformation:** The BYTE offset from the base of the structure to the array of LSAPR\_AUTH\_INFORMATION structures representing the current **Authentication** information.

**Byte offset to PreviousAuthenticationInformation:** The BYTE offset from the base of the structure to the array of LSAPR\_AUTH\_INFORMATION structures representing the previous **Authentication** information.

**AuthenticationInformation:** Array of LSAPR\_AUTH\_INFORMATION [1...Count].

Following the byte offset to PreviousAuthenticationInformation is an array of [LSAPR\\_AUTH\\_INFORMATION](#) structures representing the current **authentication** information.

**PreviousAuthenticationInformation:** Array of LSAPR\_AUTH\_INFORMATION [1...Count].

Following the current **authentication** information is an array of LSAPR\_AUTH\_INFORMATION structures representing the previous **authentication** information. If **authentication** information has not been previously stored, then the previous Authentication Information structure is an exact copy of the current Authentication Information structure.

#### 7.1.6.8.1.1 LSAPR\_AUTH\_INFORMATION

The format of LSAPR\_AUTH\_INFORMATION structure is as follows:

```
struct LSAPR_AUTH_INFORMATION {
    LARGE_INTEGER LastUpdateTime;
    ULONG AuthType;
    ULONG AuthInfoLength;
```

```

    UCHAR *AuthInfo;
}

```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
LastUpdateTime																															
...																															
AuthType																															
AuthInfoLength																															
AuthInfo (variable)																															
...																															
Padding (variable)																															
...																															

**LastUpdateTime:** This LARGE\_INTEGER value represents the last time that the **authentication** information was set, in FILETIME format, as specified in [MS-DTYP] section [2.3](#).

**AuthType:** This ULONG value dictates the type of AuthInfo which is being stored. There are four values that are recognized by Windows

Possible Values	Meaning
TRUST_AUTH_TYPE_NONE 0	AuthInfo byte field is invalid / not relevant
TRUST_AUTH_TYPE_NT4OWF 1	AuthInfo byte field contains an RC4 Key [RFC4757]
TRUST_AUTH_TYPE_CLEAR 2	AuthInfo byte field contains a cleartext password, encoded as a <b>Unicode</b> string.
TRUST_AUTH_TYPE_VERSION 3	AuthInfo byte field contains a version number, used by Netlogon for versioning interdomain <b>trust secrets</b> .

**AuthInfoLength:** A ULONG count of bytes in AuthInfo.

**AuthInfo:** A BYTE field containing **authentication** data. Its size is [1...AuthInfoLength].

**Padding:** Some number of bytes used to align the end of the LSAPR\_AUTH\_INFORMATION structure to a ULONG boundary. This padding is not included in the AuthInfoLength and consists of zeros.

#### 7.1.6.8.1.2 Kerberos Usages of trustAuthInfo Attributes

Microsoft's implementation of Kerberos ([\[RFC4120\]](#), [\[MS-KILE\]](#)) uses TDOs to retrieve cross **domain** passwords when building cross realm Ticket Granting Tickets (TGTs). The KDC supports the following AuthTypes:

##### 1. TRUST\_AUTH\_TYPE\_CLEAR

This flag indicates that the information stored in the attribute is a **Unicode** plaintext password. If this auth type is present, Kerberos can then use this password to derive additional key types needed to encrypt and decrypt cross realm TGTs:

- DES-CBC [\[RFC4120\]](#) section 8.1
- DES-CRC [\[RFC4120\]](#)
- RC4HMAC [\[RFC4757\]](#)

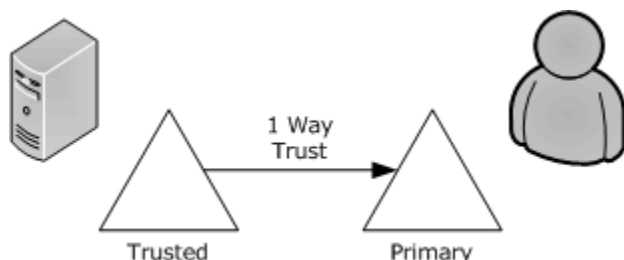
Other derivations of the plaintext password are possible using string to key functionality defined in [\[RFC3961\]](#). It is important to note that if the [trustType](#) is set to TRUST\_TYPE\_MIT, then RC4HMAC keys will not be derived for the trust *unless* the corresponding **TDO's** trustAttribute includes the TRUST\_ATTRIBUTE\_USES\_RC4\_ENCRYPTION bit flag.

In Windows Server 2008, if KERB\_ENCTYPE\_RC4\_HMAC\_MD5 (4) is set in the msDs-supportedEncryptionTypes attribute, then the MIT realm supports RC4.

##### 2. TRUST\_AUTH\_TYPE\_NT4OWF

This flag indicates that the key is stored as a raw RC4HMAC key [\[RFC4757\]](#). Since the key was precomputed with this auth type, it is not possible to derive alternate key types for the **TDO**.

Kerberos' usage of the **TDO** keys is somewhat counter-intuitive. Consider the following scenario involving two trusting Active Directory **domains**, where a user in a primary **domain** wishes to authenticate to a service in the trusted **domain** using Kerberos. The primary **domain** issues a referral TGT to the trusted **domain** containing the service.



**Figure 2: Kerberos protocol usage of keys**

There is a one-way trust in place. The referral TGT issued by the primary **domain** is encrypted based on the key in [trustAuthIncoming](#), not [trustAuthOutgoing](#). This is non-intuitive, but it fits the definition of an inbound trust. This direction of trust allows Kerberos to build a TGT for the trusted **domain** in the primary **domain**, fulfilling the definition of an inbound trust.

#### 7.1.6.8.1.3 Netlogon Usages of trustAuth Attributes

Netlogon also uses both of the **TDO** key formats to bootstrap the Secure Channel as described in [\[MS-NRPC\]](#) sections [3.1.1](#) and [3.1.5.3](#). As with Kerberos, if presented with

TRUST\_AUTH\_TYPE\_CLEAR, it can derive the necessary RC4HMAC keys. Alternately, it can use AuthInfo elements containing TRUST\_AUTH\_TYPE\_NT4OWF, and skip the key derivation entirely.

Additionally, Netlogon uses TRUST\_AUTH\_TYPE\_VERSION to version the AuthInfo stored in trust secrets. This is a ULONG value that is incremented by 1 each time the trust secret is changed ([MS-NRPC] sections 3.1.1, [3.4.5.2.5](#), and [3.5.5.2.5](#)).

**7.1.6.8.2 msDS-TrustForestTrustInfo Attribute**

Information about trust relationships with other forests is stored in objects of class [trustedDomain](#) in the **domain** NC replica of the forest root **domain**. Specifically, the [msDS-TrustForestTrustInfo](#) attribute on such objects contains information about the trusted forest or realm. The structure of the information contained in this attribute is represented in the following manner.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Version																															
RecordCount																															
Records (variable)																															
...																															

**Version:** Version of the data structure. The only supported version of the data structure is 1.

**RecordCount:** Number of records present in the data structure.

**Records:** Variable length records each containing a specific type of data about the forest trust relationship.

**IMPORTANT NOTE:** Records are not aligned to 32-bit boundaries. Each record starts at the next byte after the previous record ends.

Each record is represented as described in section [7.1.6.8.2.1](#).

**Note** All fields have little-endian byte ordering.

**7.1.6.8.2.1 Record**

Each Record is represented in the following manner:



0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
RecordLen																															
Flags																															
Timestamp																															
...																															
RecordType										ForestTrustData (variable)																					
...																															

**RecordLen:** Length, in bytes, of the entire record.

**Flags:** Individual bit flags that control how the forest trust information in this record can be used.

If RecordType = 0 or 1, the Flags field, represented here in little-endian byte order, can have one or more of the following bits:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	T	T	T	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
					D	D	D																								
					C	A	N																								

**X:** Unused. Must be zero and ignored.

**TDN (LSA\_TLN\_DISABLED\_NEW, 0x00000001):** Entry is not yet enabled.

**TDA (LSA\_TLN\_DISABLED\_ADMIN, 0x00000002):** Entry is disabled by administrator.

**TDC (LSA\_TLN\_DISABLED\_CONFLICT, 0x00000004):** Entry is disabled due to conflict with another trusted domain.

If RecordType = 2, the Flags field, represented here in little-endian byte order, can have one or more of the following bits:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	N	N	S	S	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
				D	D	D	D																								
				C	A	C	A																								

**SDA (LSA\_SID\_DISABLED\_ADMIN, 0x00000001):** Entry is disabled for SID, NetBIOS and DNS name based matches by administrator.

**SDC (LSA\_SID\_DISABLED\_CONFLICT, 0x00000002):** Entry is disabled for SID, NetBIOS and DNS name based matches due to SID or DNS name based conflict with another trusted domain.

**NDA (LSA\_NB\_DISABLED\_ADMIN, 0x00000004):** Entry is disabled for NetBIOS name based matches by administrator.

**NDC (LSA\_NB\_DISABLED\_CONFLICT, 0x00000008):** Entry is disabled for NetBIOS name based matches due to NetBIOS **domain** name conflict with another trusted domain.

For RecordType = 2, NETBIOS\_DISABLED\_MASK is defined as a mask on the lower 4 bits of the Flags field.

For all record types, LSA\_FTRECORD\_DISABLED\_REASONS is defined as a mask on the lower 16 bits of the Flags field. Unused bits covered by the mask are reserved for future use.

**Timestamp:** 64-bit timestamp value indicating when this entry was created, in system time (see the FILETIME structure in [MS-DTYP] section [2.3.1](#)).

**RecordType:** 8-bit value specifying the type of record contained in this specific entry. The structure of the content in the next field depends on this value. The allowed values for this field are specified by the enumerated list below.

Name	Value
ForestTrustTopLevelName	0
ForestTrustTopLevelNameEx	1
ForestTrustDomainInfo	2

**ForestTrustData:** Variable length type-specific record, depending on the RecordType value, containing specific type of data about the forest trust relationship.

**IMPORTANT NOTE:** The type-specific ForestTrustData record is not necessarily aligned to a 32-bit boundary. Each record starts at the byte following the RecordType field.

There are three different type-specific records. Depending on the value of the RecordType field, the structure of the type-specific record differs as described below.

- If RecordType = 0 or RecordType = 1, then the type-specific record is represented in the following manner:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NameLen																															
Name (variable length)...																															

**NameLen:** Length, in bytes, of the Name field below.

**Name:** The **top level name** of the trusted forest in UTF-8 format.

- If RecordType = 2, then the type-specific record is represented in the following manner. Note that the record contains the following structures one after another. It is important to note here that none of the data shown below is necessarily aligned to 32-bit boundaries.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SidLen																															
Sid (variable length)...																															
DnsNameLen																															
DnsName (variable length)...																															
NetbiosNameLen																															
NetbiosName (variable length)...																															

**SidLen:** Length, in bytes, of the Sid field below.

**Sid:** The SID of a **domain** in the trusted forest, specified as a SID structure which is defined in [\[MS-DTYP\]](#) section **2.4.2**.

**DnsNameLen:** Length, in bytes, of the DnsName field below.

**DnsName:** The DNS name of a **domain** in the trusted forest in UTF-8 format.

**NetbiosNameLen:** Length, in bytes, of the NetbiosName field below.

**NetbiosName:** The NetBIOS name of a **domain** in the trusted forest in UTF-8 format.

- If RecordType is not one of the values above, then the type-specific record is represented in the following manner:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
BinaryDataLen																															
BinaryData (variable length)...																															

**BinaryDataLen:** Length, in bytes, of the BinaryData field below.

**BinaryData:** Trusted forest data.

#### 7.1.6.8.2.2 Building Well Formed msDS-TrustForestTrustInfo Messages

The [msDS-TrustForestTrustInfo](#) attribute contains an String(Octet) with the data structures specified above. This attribute contains information about the namespaces served by a given trusted forest. For example, if forest a.com contains the **domains** a.com, b.a.com, and c.a.com, then the [msDS-TrustForestTrustInfo](#) for a.com would contain the FQDN and NetBIOS names for each **domain**, as well as the SID space served by each **domain**. This section details what rules well formed [msDS-TrustForestTrustInfo](#) messages must follow.

The [msDS-TrustForestTrustInfo](#) attribute is written on the PDC for the trusting and trusted **domains**. Both the trusted and trusting forest have forest functional level DS\_BEHAVIOR\_WIN2003 or greater.

Some concepts are necessary to understand the algorithm used when validating this attribute.

##### Namespaces

Namespaces are meant to represent those NetBIOS, FQDN, or SID values that a trusted forest or **domain** claims.

##### Top Level Names (TLNs)

**TLNs** are an important concept when detecting and resolving conflicts in name spaces between different TDOs, and for providing hints about which forest owns a given name space. A **TLN** really corresponds to a forest namespace, and in order to be enabled, the TLN must be unique among all TDOs. For example, the TLN for the forest example.com is example.com. Note that it is possible that the forest example.com could have another **domain** corresponding to an entirely different TLN (for example, mailservers.com), in which case 2 TLNs would need to be registered for the example.com forest. TLNs for a **TDO** are stored in records identified by the ForestTrustTopLevelName Record Type.

TLNs which must be excluded from namespace are identified by the ForestTrustTopLevelNameEx RecordType. Exclusion becomes necessary if the namespaces of two forests collide (for example the forests corp.mycompany.com and the forest hr.corp.mycompany.com). These exclusions are set administratively to ensure proper functioning of the domain.

##### Superior/Subordinate namespaces

When evaluating all forest trusts, TLNs are expressed as FQDNs. Parsing the FQDN allows the concept of superior and subordinate name spaces. For example, for the namespace sample.example.com, the superior namespace (and the TLN) is example.com. Similarly, the sample.example.com is namespace is subordinate to the example.com namespace. This allows routing mechanism to understand that the name sample.example.com is associated with the example.com namespace expressed in the TLN, as it is a subordinate.

## Enabled Records vs. Disabled Records

During validation of the Records stored in the msDS-ForestTrustForestInfo, it is possible to have TLN or namespace conflicts. In these circumstances, the conflicting record is disabled. Namespace conflicts are determined using the Record Flags specified in the above msDS-ForestTrustInfo data format definitions.

### 1. ForestTrustTopLevelName RecordType (0)

If the TDN / TDA / TDC Flags are present, then the name present in the TLN and its subordinate namespaces, as well as all **domains** the FQDNs of which are equal to or subordinate to the TLN is not used for routing names or SIDs.

### 2. ForestTrustTopLevelNameEx RecordType (1)

If the TDN / TDA / TDC Flags are present, then the name present in the exclusion TLN is not used for exclusion purposes, and conflicts will be unresolved. All **domains** whose FQDNs are equal to or subordinate to the exclusion TLN are not used for routing names or SIDs.

### 3. ForestTrustDomainInfo RecordType (2)

If the NDC or NDA Flags are set, then the NetBIOS name is excluded from routing for the NetBIOS name.

If the SDA or SDC flags are set, then the entire **domain** and all **domains** the FQDN name of which are subordinate to the FQDN name of that **domain** are excluded from name routing by SID, FQDN, or NetBIOS names. The entire subtree of the forest rooted at the affected **domain** is effectively not computed in the trust **domain** name mappings.

## msDS-TrustForestTrustInfo Validation

When the **TDO** information for a **domain** is added, changed, or the DC possessing the PDC FSMO role in the root **domain** of the forest is freshly started, every **TDO** with msDS-ForestTrustInfo attributes is validated against all other TDOs. The results of that validation are then rewritten to the DS and replicated to the other DCs in the **domain**. DCs that do not own the PDC FSMO role treat the attribute as READONLY, and internally consistent.

Validation of the matrix of trusted **domains** and trusted forest information stored in msDS-ForestTrustInfo includes a mechanism to prevent name collisions. Manipulations of this attribute ensure that each namespace is only assigned to a single **TDO**. If any of the rules below are violated, the colliding RecordFlag is marked as disabled.

The rules for determining whether namespaces collide for ForestTrustDomainInfo Records are as follows:

1. Each SID corresponding to a **domain** in a trusted forest is unique among all TDOs, and among all of the SIDs listed within the ForestTrustData Records. If not, the Record MUST have the SDC bit in the Record Flags.
2. Each SID for each **domain** in a trusted forest does not equal any SIDs within the **domains** of the local forest. If not, the Record MUST have the SDC bit in the Record Flags.
3. Each FQDN corresponding to a **domain** in a trusted forest is unique among all TDOs, and among all of the FQDNs and TLNs listed within the ForestTrustData Records. If not, the Record MUST have the SDC bit in the Record Flags.

- Each FQDN for each **domain** in the trusted forest does not correspond to any FQDNs within the **domains** from the local forest. If not, the Record MUST have the SDC bit in the Record Flags.
- Each NetBIOS **domain** name corresponding to a **domain** in a trusted forest is unique among all TDOs, and among all of the NetBIOS **domains** listed within the Forest Trust Data records. If not, the Record MUST have the NDC bit in the Record Flags. For conflict resolution, the **TDO** with the alphabetically longest name is disabled.
- Each NetBIOS name for each **domain** in the trusted forest does not equal any NetBIOS **domain** name within the **domains** of the local forest. If not, the Record MUST have the NDC bit in the Record Flags. Local forest NetBIOS names always take precedence over those of trusted forests.

The rules for determining whether namespaces collide for ForestTrustTopLevelName Records are as follows:

- Each TLN corresponding to a **domain** in a trusted forest is unique among all TDOs, and among all of the FQDNs and TLNs listed within the Forest Trust Data records. If not, the conflicting Record has the TDC bit in the Record Flags. For consistency sake, since the two TLNs are equal, the first TLN Record that is read is authoritative, and subsequent conflicting Records are disabled.
- Each TLN for each **domain** in the trusted forest does not correspond to any FQDNs within the **domains** from the local forest. If not, the Record has the TDC bit in the Record Flags.

ForestTrustTopLevelNameEx Records by definition, cannot conflict.

Additionally, additions to [msDS-TrustForestTrustInfo](#) pass namespace consistency checks before the attribute is set. Any failures in the consistency checks cause the attempt to modify the [msDS-TrustForestTrustInfo](#) to fail. The below rules dictate what requirements each trusted forest must match:

- At least one ForestTrustTopLevelName TLN Record is specified for each [msDS-TrustForestTrustInfo](#). It is possible for a forest to have more than one TLN if it contains additional TLNs.
- All **domains** listed in the ForestTrustDomainInfo for a **TDO** are subordinate to the TLNs for that **TDO**.
- All **domains** listed in the ForestTrustDomainInfo are not subordinate or superior to other TLNs unless an exclusion record for that TLN or **domain** is registered.

If all of the above tests pass, then the entry is written in binary format to the msDS-ForestTrustInfo, replicated, and honored by all DCs in the forest.

#### 7.1.6.8.3 Computation of posixOffset

A **domain** computes a posixOffset when a new **TDO** is created. This is done by retrieving all of the existing posixOffset values for each outgoing Windows trusts (both TRUST\_TYPE\_UPLEVEL and TRUST\_TYPE\_DOWNLEVEL). These values are then sorted. Finally, the range of numbers is searched from 1 looking for the next unused valid POSIX offset. The selection process excludes the values below, which are reserved for well-known identities:

Value	Description
0x0800	Reserved for built-in domain
0x4000	Reserved for account domain

Value	Description
0xC000	Reserved for primary domain

The selection process only happens on the DC possessing the PDC FSMO role. If the trust creation happens on another DC, the posixOffset value is set to 0, and is computed using the logic above when the **TDO** replicates to the PDC FSMO role owner. This keeps TDOs from having matching POSIX offsets, which could result in collisions of UIDS and GIDS.

## 7.1.6.8.4 Timers

### 7.1.6.8.4.1 Trust Secret Cycling

The keys used to validate trusts periodically expire (typically every 30 days). This is performed by the Netlogon service, which performs this operation when establishing the Secure Channel. Resetting the secure channel secret is covered under [\[MS-NRPC\]](#) section 3.5.4.3.5.

### 7.1.6.8.5 Initialization

Despite being replicated normally between peer DCs in a **domain**, the process of creating, or manipulating TDOs is specifically restricted to the LSA Policy APIs as detailed in [\[MS-LSAD\]](#) section 3.1.1.5. Unlike other objects in the DS, TDOs may not be created or manipulated by client machines over the LDAPv3 transport.

The following trust manipulation **Remote Procedure Calls** specifically target TDOs, and are responsible for creating the special properties detailed in section [7.1.6.7](#). All are documented in [\[MS-LSAD\]](#) section 3.1.4.

- LsarCreateTrustedDomainEx()
- LsarDeleteTrustedDomain()
- LsarSetTrustedDomainInfoByName()
- LsarSetTrustedDomainInformation()

The above APIs enforce the following restrictions:

Each **TDO** corresponds to exactly one trusted **domain**. The FQDN, SID, and NetBIOS name set on the **TDO** all reference the same domain.

The server verifies that the trust is pointing either to a **domain** within the forest, or a **domain** outside the forest. The check is performed by verifying whether any other **domain** within the forest has the security identifier, DNS name or NetBIOS name matching the information being set. One of two options is legal:

1. SID, DNS name and NetBIOS name all match the same **domain** within the forest.
2. None of SID, DNS name and NetBIOS name match any **domain** within the forest.

Any other alternative (some information pointing inside the forest, and some outside, or information pointing at different **domains** within the forest) is illegal and causes the server to fail the request.

An attempt by requestor to set the TRUST\_ATTRIBUTE\_FOREST\_TRANSITIVE bit in trust attributes of the trusted **domain** object can only succeed if the **domain** is in a forest functional level of

DS\_BEHAVIOR\_WIN2003 or greater and the server is a **domain** controller in the root **domain** of the forest. Failing this, the server rejects the request, and does not create the **TDO**.

An attempt by the requestor to set the TRUST\_ATTRIBUTE\_CROSS\_ORGANIZATION bit in the trust attributes of the trusted **domain** object can only succeed if the **domain** is in a forest functional level of DS\_BEHAVIOR\_WIN2003 or greater. Failing this, the server rejects the request, and does not create the **TDO**.

Neither TRUST\_ATTRIBUTE\_FOREST\_TRANSITIVE nor TRUST\_ATTRIBUTE\_CROSS\_ORGANIZATION bits are compatible with the TRUST\_ATTRIBUTE\_WITHIN\_FOREST bit. The server rejects invalid combinations of trust attributes, and does not create the **TDO**.

Uplevel or **downlevel trusts** that have TRUST\_DIRECTION\_OUTBOUND as one of the direction bits cannot have a SID of NULL. Attempts to set this combination of parameters causes the server to fail the request.

If the TRUST\_ATTRIBUTE\_FOREST\_TRANSITIVE bit is cleared from a **TDO's** [trustAttributes](#) attribute, all of forest trust information in the msDS-Trust on that **TDO** is removed from the **TDO's** [msDS-TrustForestTrustInfo](#) attribute.

### 7.1.6.9 Security Considerations for Implementers

Mechanisms of trust depend on secure initialization. [\[MS-LSAD\]](#) describes the secure trust creation system used by Active Directory. In this system, all creation and manipulation of TDOs takes place over a secure session transport, where the client has been authenticated, and sensitive trust information is not sent in the clear. Keys used for trust secrets are sufficiently strong to disallow brute force attacks on the cryptographic material used in cross **domain** protocols.

### 7.1.7 DynamicObject invariants

Dynamic objects are objects that are automatically deleted after a period of time. They are distinguished from regular objects by the presence of the [dynamicObject](#) auxiliary class among their [objectClass](#) values. The intended time of deletion is specified by [msDS-Entry-Time-To-Die](#) attribute.

The following invariants apply to dynamic objects:

- All of dynamic object's descendants are dynamic objects.
- A dynamic object MUST be automatically deleted when all of the following conditions are true:
  - The current time value is greater than or equal to its [msDS-Entry-Time-To-Die](#) attribute value
  - It has no descendants
- If the [msDS-Entry-Time-To-Die](#) of a [dynamicObject](#) **container** is earlier than [msDS-Entry-Time-To-Die](#) of its descendant, then the DC MUST update the [msDS-Entry-Time-To-Die](#) of the **container** to be greater than the maximum [msDS-Entry-Time-To-Die](#) of its descendants. This update MUST occur before current time reaches its original [msDS-Entry-Time-To-Die](#) value.
- When a dynamic object expires, any linked DN references from other objects to this object MUST be removed.
- When a dynamic object expires, any non-linked DN references pointing to it MUST be retained. Their value is the last known DN of the object.
- The value of the [entryTTL](#) constructed attribute is specified in section [3.1.1.4.5.12](#).



## 7.2 Knowledge Consistency Checker

A server running Active Directory is part of a distributed system that performs replication. The Knowledge Consistency Checker (KCC) is a component that reduces the administrative burden of maintaining a functioning replication topology. Additional background is provided in section [3.1.1.1.13](#).

### 7.2.1 References

- DRS options bits: [MS-DRSR] section [5:DRS\\_OPTIONS](#).
- [instanceType](#) bits: [MS-DRSR] section [5:instanceType bit flags](#).
- [repsFrom](#) abstract attribute: [MS-DRSR] section [5:repsFrom](#).
- [repsTo](#) abstract attribute: [MS-DRSR] section [5:repsTo](#).
- [replUpToDateVector](#) abstract attribute: [MS-DRSR] section [5:replUpToDateVector](#).
- `kCCFailedConnections` and `kCCFailedLinks` variables: [MS-DRSR] sections [5:KCCFailedConnections](#) and [5:KCCFailedLinks](#).
- `IDL_DRSGetNCChanges` method: [MS-DRSR] section [4.1.10](#).
- `IDL_DRSReplicaAdd` method: [MS-DRSR] section [4.1.19](#).
- `IDL_DRSReplicaDel` method: [MS-DRSR] section [4.1.20](#).
- `IDL_DRSReplicaModify` method: [MS-DRSR] section [4.1.22](#).
- `IDL_DRSExecuteKCC` method: [MS-DRSR] section [4.1.6](#).
- `DWORD`, `GUID` types: [MS-DTYP] sections [2.2](#) and [2.3.2](#).
- `AmIRODC` method: [MS-DRSR] section [5:AmIRODC](#).

### 7.2.2 Overview

The Knowledge Consistency Checker (KCC) automates management of the NC replica graph for each NC in the forest. In doing so, it maintains the following invariants:

- There exists a path from each writable replica to every other NC replica (writable, read-only full or read-only partial) of the same NC.
- No path from a writable replica to another writable replica passes through a read-only replica.
- For each **domain** NC, the path from a writable replica to another writable replica utilizes only the **RPC transport** (never SMTP [[MS-SRPL](#)]).
- For each **domain** NC, the path from a writable replica to a read-only full replica utilizes only the **RPC** transport (never SMTP [[MS-SRPL](#)]).
- Replication latency is short between NC replicas on DCs in the same site, at the expense of additional replication traffic within the site.
- Replication traffic between sites is low, at the expense of additional replication latency between sites.

- A state in which one or more DCs are offline or unreachable (temporarily or indefinitely) does not cause the replication latency across the remaining DCs to grow without bound.
- Edges between DCs in different sites constitute a least cost spanning tree for an administrator-defined cost metric.

The KCC performs this work in a sequence of tasks called a *run*. These runs execute periodically and on receipt of an IDL\_DRSExecuteKCC request. The first periodic run of the Windows KCC begins 5 minutes after system startup. Subsequent runs execute such that the interval between the end of one run and the beginning of the next run is 15 minutes.

These tasks utilize the following inputs:

- Config NC objects: [crossRef](#), [interSiteTransport](#), [nTDSDSA](#), [nTDSConnection](#), [site](#), [nTDSSiteSettings](#), [siteLink](#), [siteLinkBridge](#)
- Abstract attributes of NC replicas: [repsFrom](#), [repsTo](#)
- Variables of DCs: kCCFailedConnections, kCCFailedLinks
- Current date/time

and produce or update the following:

- Config NC objects: [nTDSConnection](#)
- Abstract attributes of NC replicas: [repsFrom](#), [repsTo](#)
- Variables of DCs: kCCFailedConnections, kCCFailedLinks

The KCC individual tasks are detailed in the remainder of this section, and are executed in the sequence in which they appear in this document. In summary, these tasks are:

- Refresh kCCFailedLinks and kCCFailedConnections.
- Create intra-site connections.
- Create inter-site connections.
- Remove unnecessary connections.
- Translate connections.
- Remove unnecessary kCCFailedLinks and kCCFailedConnections.

To simplify the task descriptions, the following concepts are used:

- An NC replica that "is present" on a DC. Given NC replica *r* of NC *n* and a DC with [nTDSDSA](#) object *o*, *r* "is present" on the DC if both of the following conditions is true:
  - *o*![hasMasterNCs](#) contains *n* or *o*![msDS-hasFullReplicaNCs](#) contains *n* or *o*![hasPartialReplicaNCs](#) contains *n*.
  - one of the following two conditions is true:
    - *o*![msDS-HasInstantiatedNCs](#) contains no value *v* where the dsname portion of *v* = *n*. (In this case *n* is in the process of being instantiated.)

- *o!*[msDS-HasInstantiatedNCs](#) contains a value *v*, where the dsname part of *v* = *n*, and the binary part of *v* (DWORD in big-endian byte order) is an integer such that the IT\_NC\_GOING bit is clear. (In this case *n* is instantiated, and is not in the process of being uninstantaed.)
- An NC replica that "should be present" on a DC. Given NC replica *r* of NC *n* and a DC with [nTDSDSA](#) object *o*, *r* "should be present" on the DC if *r* is one of the following:
  - A writable replica of the config NC, the schema NC, or the DC's default NC on a writable DC.
  - A full read-only replica of the config NC, the schema NC, or the DC's default NC on a RODC.
  - A writable replica of an application NC for which there exists a [crossRef](#) object *cr* such that *cr!*[nCName](#) = *n* and *cr!*[msDS-NC-Replica-Locations](#) contains a reference to *o*.
  - A full read-only replica of an application NC for which there exists a [crossRef](#) object *cr* such that *cr!*[nCName](#) = *n* and *cr.ms-DS-NC-RO-Replica-Locations* contains a reference to *o*.
  - If the DC is a GC server (that is, if bit NTDSDSA\_OPT\_IS\_GC is set in *o!*[options](#)), a partial replica of a **domain** NC *n* such that *n* ≠ the DC's default NC and there exists a [crossRef](#) object *cr* such that *cr!*[nCName](#) = *n*.
- An [nTDSConnection](#) object "implies" a tuple in the [repsFrom](#) abstract attribute of an NC replica (and a corresponding edge in an NC replica graph). An [nTDSConnection](#) object *cn* implies a tuple in *r!*[repsFrom](#) for NC replica *r* of NC *n* on the DC with [nTDSDSA](#) object *t* if each of the following is true:
  - *cn* is a child of *t*.
  - *cn!*[fromServer](#) references an [nTDSDSA](#) object *s*.
  - An NC replica of *n* "is present" on *s*.
  - *r* "should be present" on *t*.
  - The NC replica on *s* is a full replica or *r* is a partial replica.
  - *n* is not a **domain** NC, or *r* is a partial replica, or *cn!*[transportType](#) has no value, or *cn!*[transportType](#) has an RDN of CN=IP.

### 7.2.2.1 Refresh kCCFailedLinks and kCCFailedConnections

This task refreshes and reconciles the contents of the *kCCFailedLinks* and *kCCFailedConnections* variables.

The KCC updates *kCCFailedLinks* by inspecting the [repsFrom](#) abstract attribute associated with each NC replica on the local DC. It first resets the *FailureCount* of each tuple in *kCCFailedLinks* to 0. Then, for each NC replica *r*, for each tuple *rf* in *r!*[repsFrom](#), if *rf.consecutiveFailures* > 0:

- If a tuple *f* exists in *kCCFailedLinks* such that *f.UUIDSsa* = *rf.uuidSsa* and *f.FailureCount* ≠ 0:
  - Set *f.FailureCount* to MAX(*f.FailureCount*, *rf.consecutiveFailures*)
  - Set *f.TimeFirstFailure* to MIN(*f.TimeFirstFailure*, *rf.timeLastSuccess*)
  - Set *f.LastResult* to *rf.resultLastAttempt*
- If a tuple *f* exists in *kCCFailedLinks* such that *f.UUIDSsa* = *rf.uuidSsa* and *f.FailureCount* = 0:

- Set *f.FailureCount* to *rf.consecutiveFailures*
- Set *f.TimeFirstFailure* to *rf.timeLastSuccess*
- Set *f.LastResult* to *rf.resultLastAttempt*
- If no tuple *f* exists in *kCCFailedLinks* such that *f.UUIDDsa* = *rf.uuidDsa*, add tuple *g* to *kCCFailedLinks* such that
  - *g.UUIDDsa* = *rf.uuidDsa*
  - *g.FailureCount* = *rf.consecutiveFailures*
  - *g.TimeFirstFailure* = *rf.timeLastSuccess*
  - *g.LastResult* = *rf.resultLastAttempt*

For each tuple *k* in *kCCFailedConnections*, the KCC attempts to connect to that DC by calling the IDL\_DRSBind method. If the method call is successful, the KCC removes *k* from *kCCFailedConnections*. Otherwise, it increments *k.FailureCount* by 1.

### 7.2.2.2 Intra-site Connection Creation

This task computes an NC replica graph for each NC replica that "should be present" on the local DC. Then for each edge of the graph directed to an NC replica on the local DC, the KCC reconciles its portion of the NC replica graph by creating an [nTDSConnection](#) object to "imply" that edge if one does not already exist.

If the site of the local DC has a site settings object *o* and the NTDSSETTINGS\_OPT\_IS\_AUTO\_TOPOLOGY\_DISABLED bit is set in *o!options*, the KCC skips this task.

For each NC *x* for which have an NC replica "should be present" on the local DC, the KCC constructs an NC replica graph as follows:

- Let *R* be a sequence containing each writable replica *f* of *x* such that *f* "is present" on a DC *s* satisfying the following criteria:
  - *s* is a writable DC other than the local DC.
  - *s* is in the same site as the local DC.
  - If *x* is a read-only full replica and *x* is a **domain** NC then the DC's functional level is at least DS\_BEHAVIOR\_WIN2008
  - Bit NTDSSETTINGS\_OPT\_IS\_TOPL\_DETECT\_STALE\_DISABLED is set in the [options](#) attribute of the site settings object for the local DC's site, or no tuple *z* exists in the *kCCFailedLinks* or *kCCFailedConnections* variables such that *z.UUIDDsa* is the [objectGUID](#) of the [nTDSDSA](#) object for *s*, *z.FailureCount* > 0, and the current time - *z.TimeFirstFailure* > 2 hours.
- If a partial (not full) replica of *x* "should be present" on the local DC, append to *R* each partial replica *p* of *x* such that *p* "is present" on a DC *s* satisfying the same criteria defined above for full replica DCs.
- Append to *R* the NC replica that "should be present" on the local DC.
- Sort *R* in order of the value of the [objectGUID](#) attribute of the corresponding DC's [nTDSDSA](#) object. Let *r<sub>i</sub>* be the *i*'th NC replica in *R*, where 0 ≤ *i* < |*R*|.

- Add a node for each  $r_i$  to the NC replica graph.
- Add an edge from  $r_i$  to  $r_{i+1}$  for each  $0 \leq i < |R| - 1$  if  $r_i$  is a full replica or  $r_{i+1}$  is a partial replica.
- Add an edge from  $r_{i+1}$  to  $r_i$  for each  $0 \leq i < |R| - 1$  if  $r_{i+1}$  is a full replica or  $r_i$  is a partial replica.
- Add an edge from  $r_{|R|-1}$  to  $r_0$  if  $r_{|R|-1}$  is a full replica or  $r_0$  is a partial replica.
- Add an edge from  $r_0$  to  $r_{|R|-1}$  if  $r_0$  is a full replica or  $r_{|R|-1}$  is a partial replica.

The KCC can create additional edges, but does not create more than 50 edges directed to a single DC. To optimize replication latency in sites with many NC replicas, the Windows KCC determines that each  $r_i$  should have  $n+2$  total edges directed to it such that  $n$  is the smallest non-negative integer satisfying  $|R| \leq 2n^2 + 6n + 7$ . For each existing [nTDSConnection](#) object implying an edge from  $r_j$  of  $R$  to  $r_i$  such that  $j \neq i$ , an edge from  $r_j$  to  $r_i$  is not already in the graph, and the total edges directed to  $r_i$  is less than  $n+2$ , the KCC adds that edge to the graph. The KCC then adds new edges directed to  $r_i$  to bring the total edges to  $n+2$ , where the NC replica  $r_k$  of  $R$  from which the edge is directed is chosen at random such that  $k \neq i$  and an edge from  $r_k$  to  $r_i$  is not already in the graph.

For each edge directed to the NC replica that "should be present" on the local DC, the KCC determines whether an object  $c$  exists such that:

- $c$  is a child of the local DC's [nTSDSA](#) object.
- $c!\text{objectCategory} = \text{nTDSConnection}$
- Given the NC replica  $r_i$  from which the edge is directed,  $c!\text{fromServer}$  is the dsname of the [nTSDSA](#) object of the DC on which  $r_i$  "is present".

If no such object  $c$  exists, the KCC adds an object  $c$  to the local DC's NC replica of the config NC such that it satisfies the above criteria and has the following additional attributes:

- $c!\text{objectClass}$  contains [nTDSConnection](#)
- $c!\text{enabledConnection} = \text{true}$
- $c!\text{options} = \text{NTDSConn\_OPT\_IS\_GENERATED}$
- $c!\text{systemFlags} = \text{FLAG\_CONFIG\_ALLOW\_RENAME} + \text{FLAG\_CONFIG\_ALLOW\_MOVE}$
- $c!\text{schedule} = z : \text{SCHEDULE}$ , such that
  - $z.\text{Size} = 188$
  - $z.\text{Bandwidth} = 0$
  - $z.\text{NumberOfSchedules} = 1$
  - $z.\text{Schedules}[0].\text{Type} = 0$
  - $z.\text{Schedules}[0].\text{Offset} = 20$
  - Byte offset 20 from  $z$  begins a stream of 168 bytes with value 0x01.

If the DC is a GC server, the KCC constructs an additional NC replica graph (and creates [nTDSConnection](#) objects) for the config NC as above, except that only NC replicas that "are present" on GC servers are added to  $R$ .

The DC repeats the NC replica graph computation and [nTDSConnection](#) creation for each of the NC replica graphs above, this time assuming that no DC has failed. It does so by re-executing the steps as if the bit NTDSSETTINGS\_OPT\_IS\_TOPL\_DETECT\_STALE\_DISABLED were set in the [options](#) attribute of the site settings object for the local DC's site.

The net result of each DC executing this distributed algorithm is the following set of overlapping rings:

- For each NC, a ring containing each full replica in the site.
- For each NC, a ring containing each NC replica (full or partial) in the site.
- A ring containing each GC server in the site.
- For each NC, a ring containing each full replica in the site that has not failed.
- For each NC, a ring containing each NC replica (full or partial) in the site that has not failed.
- A ring containing each GC server in the site that has not failed.

### 7.2.2.3 Inter-site Connection Creation

This task computes an NC replica graph for each NC replica that "should be present" on the local DC or "is present" on any DC in the same site as the local DC. For each edge directed to an NC replica on such a DC from an NC replica on a DC in another site, the KCC reconciles its portion of the NC replica graph by creating an [nTDSConnection](#) object to "imply" that edge if one does not already exist.

If the site of the local DC has a site settings object *o* and the NTDSSETTINGS\_OPT\_IS\_INTER\_SITE\_AUTO\_TOPOLOGY\_DISABLED bit is set in *o*! [options](#), the KCC skips this task.

Like intra-site connection, inter-site connection creation utilizes distributed algorithms - algorithms that rely upon each DC in the forest implementing the same algorithm and arriving at the same conclusions given the same inputs. However, the algorithms used for inter-site connection creation are significantly more complex. Sufficient analysis of a given variation of this algorithm may yield that DCs implementing the variation are compatible with Windows DCs, but no such different-yet-compatible algorithm is known. To illustrate this point, consider the following simple example:

Assume a forest *F* that contains three DCs of the same **domain** in three distinct sites - DC1 in Site1, DC2 in Site2, and DC3 in Site3 - where [siteLink](#) objects exist specifying that each site is connected to the other two sites with the same cost. DC1 and DC2 execute one implementation of the KCC and DC3 executes a different implementation.

DC1 and DC2 determine that the 3 sites should be connected by a minimum cost spanning tree rooted at site3: both DC1 and DC2 replicate updates from DC3, assuming that DC3 replicates updates from DC1 and DC2.

DC3, because it is running a different implementation, determines that the 3 sites should be connected by a minimum cost spanning tree rooted at site1: DC3 replicates updates from DC1, assuming that DC2 replicates updates from DC1 and DC1 replicates updates from DC2 and DC3.

The minimum cost spanning trees chosen by all the DCs are equally valid. However, the fact that they did not arrive at the same conclusions results in a violation of the first invariant described in section [7.2.1](#):

- DC1 replicates updates from DC3.

- DC3 replicates updates from DC1.
- DC2 replicates updates from DC3 (and therefore transitively receives updates from DC1).
- Neither DC1 nor DC2 replicates updates from DC2.

Slight variations in algorithms may result in similar failures that appear only when given specific, complex combinations of inputs. For this reason, these algorithms are described to a high level of detail, and implementers must carefully analyze any deviations from them.

### 7.2.2.3.1 ISTG Selection

First, the KCC on a writable DC determines whether it acts as an ISTG for its site.

- Let  $s$  be the object such that  $s!\text{IDAPDisplayName} = \text{nTDSDSA}$  and  $\text{classSchema}$  in  $s!\text{objectClass}$ .
- Let  $D$  be the sequence of objects  $o$  in the site of the local DC such that  $o!\text{objectCategory} = s$ .  $D$  is sorted in ascending order by  $\text{objectGUID}$ .
- Let  $o$  be the site settings object for the site of the local DC, or NULL if no such  $o$  exists.
- Let  $f$  be the duration  $o!\text{interSiteTopologyFailover}$  seconds, or 2 hours if  $o!\text{interSiteTopologyFailover}$  is 0 or has no value.
- If  $o \neq \text{NULL}$  and  $o!\text{interSiteTopologyGenerator}$  is not the  $\text{nTDSDSA}$  object for the local DC and  $o!\text{interSiteTopologyGenerator}$  is an element  $d_j$  of sequence  $D$ :
  - Let  $c$  be the cursor in the  $pUpToDateVector$  variable associated with the NC replica of the config NC such that  $c.uuidDsa = d_j!\text{invocationId}$ . If no such  $c$  exists (*No evidence of replication from current ITSG*):
    - Let  $i = j$ .
    - Let  $t = 0$ .
  - Else if the current time  $< c.\text{timeLastSyncSuccess} - f$  (*Evidence of time sync problem on current ITSG*):
    - Let  $i = 0$ .
    - Let  $t = 0$ .
  - Else (*Evidence of replication from current ITSG*):
    - Let  $i = j$ .
    - Let  $t = c.\text{timeLastSyncSuccess}$ .
- Otherwise (*Nominate local DC as ISTG*):
  - Let  $i$  be the integer such that  $d_i$  is the  $\text{nTDSDSA}$  object for the local DC.
  - Let  $t = \text{the current time}$ .
- (*Compute a function that maintains the current ISTG if it is alive, cycles through other candidates if not.*) Let  $k$  be the integer  $(i + ((\text{current time} - t) / o!\text{interSiteTopologyFailover})) \bmod |D|$ .

The local writable DC acts as an ISTG for its site if and only if  $d_k$  is the nTDSDSA object for the local DC. If the local DC does not act as an ISTG, the KCC skips the remainder of this task.

If the local DC does act as an ISTG and  $o$  exists but  $o!$ [interSiteTopologyGenerator](#) is not the dsname of the local DC's [nTDSDSA](#) object, the KCC performs an originating write to set  $o!$ [interSiteTopologyGenerator](#) to this value.

The KCC on an RODC always acts as an ISTG for itself.

#### 7.2.2.3.2 Merge of kCCFailedLinks and kCCFailedLinks from Bridgeheads

The KCC on a writable DC attempts to merge the link and connection failure information from bridgehead DCs in its own site to help it identify failed bridgehead DCs.

For each [nTDSDSA](#) object  $bh$  with [objectCategory](#) [nTDSDSA](#) other than the local DC but in the local DC's site, if  $bh$  has a child [nTDSConnection](#) object  $cn$  such that  $cn!$ [fromServer](#) is a reference to an [nTDSDSA](#) object in a site other than the local DC's site, the KCC adds the tuples from  $bh$ 's  $kCCFailedConnections$  and  $kCCFailedLinks$  to the tuples in those same variables on the local DC. It does so by calling in the sequence IDL\_DRSBind, IDL\_DRSGetReplInfo for DS\_REPL\_INFO\_KCC\_DSA\_CONNECT\_FAILURES, IDL\_DRSGetReplInfo for DS\_REPL\_INFO\_KCC\_DSA\_LINK\_FAILURES, and IDL\_DRSUnbind.

If any of these calls fails, the KCC adds a tuple for  $bh!$ [objectGUID](#) to  $kCCFailedConnections$ .

For each DS\_REPL\_INFO\_KCC\_DSA\_FAILUREW  $d$  it receives, the KCC updates its corresponding variable  $v$  ( $kCCFailedLinks$  for DS\_REPL\_INFO\_KCC\_DSA\_LINK\_FAILURES,  $kCCFailedConnections$  for DS\_REPL\_INFO\_KCC\_DSA\_CONNECT\_FAILURES) as follows:

- If a tuple  $f$  exists in  $v$  such that  $f.UUIDDsa = d.uuidDsaObjGuid$  and  $f.FailureCount \neq 0$ :
  - Set  $f.FailureCount$  to  $\text{MAX}(f.FailureCount, d.cNumFailures)$
  - Set  $f.TimeFirstFailure$  to  $\text{MIN}(f.TimeFirstFailure, d.ftimeFirstFailure)$
  - Set  $f.LastResult$  to  $d.dwLastResult$
- If a tuple  $f$  exists in  $v$  such that  $f.UUIDDsa = d.uuidDsaObjGuid$  and  $f.FailureCount = 0$ :
  - Set  $f.FailureCount$  to  $d.cNumFailures$
  - Set  $f.TimeFirstFailure$  to  $d.ftimeFirstFailure$
  - Set  $f.LastResult$  to  $d.dwLastResult$
- If no tuple  $f$  exists in  $v$  such that  $f.UUIDDsa = d.uuidDsaObjGuid$ , add tuple  $g$  to  $v$  such that
  - $g.UUIDDsa = d.uuidDsaObjGuid$
  - $g.FailureCount = d.cNumFailures$
  - $g.TimeFirstFailure = d.ftimeFirstFailure$
  - $g.LastResult = d.dwLastResult$

#### 7.2.2.3.3 Site Graph Concepts

For each NC with an NC replica that "should be present" on the local DC or "is present" on any DC in the same site as the local DC, the KCC constructs a site graph - a precursor to an NC replica graph.



The site connectivity for a site graph is defined by objects of class [interSiteTransport](#), [siteLink](#), and [siteLinkBridge](#) in the config NC. The semantics of these objects are described in section [7.1](#).

The pseudocode in the next section maps these objects and the various constraints on these objects as follows:

KCC concept	Site graph concept
<a href="#">site</a>	VERTEX
<a href="#">siteLink</a>	MULTIEDGE
<a href="#">siteLinkBridge</a>	MULTIEDGESET
<a href="#">interSiteTransport</a>	MULTIEDGE.Type
A <a href="#">siteLink</a> object may connect more than two sites.	All vertices in a MULTIEDGE are treated as a fully connected subgraph.
<a href="#">siteLink</a> object attributes: <a href="#">cost</a> , <a href="#">schedule</a> , <a href="#">options</a> , and <a href="#">replInterval</a> .	MULTIEDGE properties in its ReplInfo field: Cost, Schedule, Options, and Interval. As paths are formed, this information is aggregated.
<a href="#">siteLink</a> objects of different interSiteTransports objects co-exist in the same graph and compete based on cost.	MULTIEDGES with differing Types co-exist in the graph and in the spanning tree.
Only the <a href="#">siteLink</a> objects referenced by a <a href="#">siteLinkBridge</a> may be combined together to form aggregated paths, with the vertices in common acting as routers.	MULTIEDGES in a MULTIEDGESET are considered transitive.
NTDSTRANSPORT_OPT_BRIDGES_REQUIRED bit in the <a href="#">options</a> attribute of an <a href="#">interSiteTransport</a> object.	If clear, a MULTIEDGESET is inferred that includes all MULTIEDGES with the corresponding Type.
NTDSTRANSPORT_OPT_IGNORE_SCHEDULES bit in the <a href="#">options</a> attribute of an <a href="#">interSiteTransport</a> object	If set, all MULTIEDGES with the corresponding Type have a Schedule that is NULL.
For a given NC, a site may contain one or more writable replicas and zero or more partial read-only replicas, zero writable replicas but one or more partial read-only replicas, or zero writable replicas and zero partial read-only replicas.	VERTEX.Color VERTEX.Color is RED, BLACK, or WHITE, respectively.
A full replica cannot replicate from a partial replica.	No edge exists from a black vertex to a red vertex.
For each NC other than the config NC and the schema NC, the path from a writable replica to another full replica utilizes only the <b>RPC</b> transport.	VERTEX.AcceptRedRed VERTEX.AcceptBlack If both vertices for a given edge are red, the edge's type must be in the AcceptRedRed set of both vertices. If one or both vertices for a given edge are black, the edge's type must be in the AcceptBlack set of

KCC concept	Site graph concept
	both vertices.
A site without a bridgehead DC for a particular transport cannot replicate updates over that transport to or from DCs in other sites.	The vertex for such a site does not contain the corresponding type in its AcceptRedRed or AcceptBlack properties.

#### 7.2.2.3.4 Connection Creation

The methods described in this section calculate a spanning tree for each NC replica graph and create corresponding [nTDSConnection](#) objects that "imply" the corresponding spanning tree edges.

This pseudocode utilizes an type SEQUENCE<X>, which is a sequence of values of a given type X. Values of type X may be appended to and removed from the sequence. If s is a value of type SEQUENCE<X>, s[i] is the i'th value in s, such that  $0 \leq i < |s|$ .

It also references the types DWORD and GUID from [\[MS-DTYP\]](#) sections [2.2](#) and [2.3.2](#).

##### 7.2.2.3.4.1 Types

The following new types are used to represent and to evaluate site graphs:

```

/***** REPL_INFO *****/
/* Replication parameters of a graph edge. */
struct REPLINFO {
    DWORD Cost;                /* Cost of network traffic between
                               * vertices; lower is preferred. */
    DWORD Interval;            /* Interval between replication attempts.
                               */
    DWORD Options;             /* siteLink object options bits. */
    SCHEDULE Schedule;         /* Schedule during which communication is
                               * possible; NULL means "always". */
}
/***** COLOR *****/
/* Color of a vertex. */
enum COLOR {
    RED,    /* Site contains one or more full replicas. */
    BLACK,  /* Site contains no full replicas but one or more
             * partial replicas. */
    WHITE   /* Site contains no replicas. */
}
/***** VERTEX *****/
/* A vertex in the site graph. */
struct VERTEX {
    GUID ID;                   /* objectGUID of corresponding site
                               * object. */
    SEQUENCE<GUID> EdgeIDs;    /* Edges currently being evaluated
                               * for this vertex. */
    COLOR Color;               /* Color of the vertex. */
    SEQUENCE<GUID> AcceptRedRed; /* Edge types accepted when both
                               * vertices are RED. */
    SEQUENCE<GUID> AcceptBlack; /* Edge types accepted when one or
                               * both vertices are BLACK. */
    REPLINFO ReplInfo;         /* Replication parameters. */
    int DistToRed;              /* Distance in the spanning tree

```

```

        * from this vertex to the nearest
        * red vertex. */

/* Dijkstra data */
GUID RootID;

bool Demoted;

/* Kruskal data */
GUID ComponentID;

int ComponentIndex;

}
/***** MULTIEDGE *****/
/* Fully connected subgraph of vertices. */
struct MULTIEDGE {
    GUID ID;
    SEQUENCE<GUID> VertexIDs;
    GUID Type;
    REPLINFO ReplInfo;
    bool Directed;
}
/***** MULTIEDGESET *****/
/* Set of transitively connected MULTIEDGES. All edges within the set
 * have the same Type. */
struct MULTIEDGESET {
    GUID ID;
    SEQUENCE<GUID> EdgeIDs;
}
/***** GRAPH *****/
/* A site graph. */
struct GRAPH {
    SEQUENCE<VERTEX> Vertices;
    SEQUENCE<MULTIEDGE> Edges;
    SEQUENCE<MULTIEDGESET> EdgeSets;
}
/***** INTERNALEDGE *****/
/* Path found in the graph between two non-WHITE vertices. */
struct INTERNALEDGE {
    GUID V1ID, V2ID;
    bool RedRed;
    REPLINFO ReplInfo;
    GUID Type;
}

```

### 7.2.2.3.4.2 Main Entry Point

The CreateIntersiteConnections method is the beginning of the control flow. This method invokes the remainder of the methods, directly or indirectly.

```

/***** CreateIntersiteConnections *****/
/* Computes an NC replica graph for each NC replica that "should be
 * present" on the local DC or "is present" on any DC in the same site
 * as the local DC. For each edge directed to an NC replica on such a
 * DC from an NC replica on a DC in another site, the KCC creates an
 * nTDSConnection object to imply that edge if one does not already
 * exist.
 *
 * OUT: keepConnections - A sequence of objectGUID values of
 *       nTDSConnection objects for edges that are directed to the
 *       local DC's site in one or more NC replica graphs.
 * RETURNS: TRUE if spanning trees were created for all NC replica
 *          graphs, otherwise FALSE.
 */
CreateIntersiteConnections(OUT SEQUENCE<GUID> keepConnections) : bool
{
    LET allConnected be TRUE
    SET keepConnections to an empty sequence of GUID

    LET crossRefList be the set containing each object o of class
    crossRef such that o is a child of the CN=Partitions child of the
    config NC
    FOR each crossRef object cr in crossRefList
        IF cr!enabled has a value and is false, or if FLAG_CR_NTDS_NC
        is clear in cr!systemFlags, skip cr.
        LET g be the GRAPH return of SetupGraph()

        /* Create nTDSConnection objects, routing replication traffic
         * around "failed" DCs. */
        LET foundFailedDC be a Boolean variable
        LET c be the Boolean return of CreateConnections(g, cr, TRUE,
        keepConnections, foundFailedDC)

        IF !c
            SET allConnected to FALSE
            IF foundFailedDC
                /* One or more failed DCs preclude use of the ideal NC
                 * replica graph. Add connections for the ideal graph.
                 */
                CALL CreateConnections(graph, cr, FALSE,
                keepConnections, foundFailedDCs)
            ENDIF
        ENDIF
    ENDFOR

    RETURN allConnected
}

```

### 7.2.2.3.4.3 Site Graph Construction

The following methods construct the initial site graph, comprising the vertices, multi-edges, and multi-edge sets corresponding to the [site](#), [siteLink](#), and [siteLinkBridge](#) objects (resp.) in the config NC.

```

/***** SetupGraph *****/
/* Set up a GRAPH, populated with a VERTEX for each site object, a

```

```

* MULTIEDGE for each siteLink object, and a MUTLIEDGESET for each
* siteLinkBridge object (or implied siteLinkBridge).
*
* RETURNS: A new graph. */
SetupGraph() : GRAPH
{
    LET vertexIDs be the sequence containing the objectGUID of each
    site object child of the CN=Sites child of the config NC
    LET g be the GRAPH return of CreateGraph(vertexIDs)
    LET localSite be the site object for the site of the local DC

    FOR each interSiteTransport object t that is a child of the
    CN=Inter-Site Transports child of the CN=Sites child of the config
    NC
        LET L be the set containing each siteLink object that is a
        child of t
        FOR each l in L
            APPEND CreateEdge(t!objectGUID, l) to g.Edges
        ENDFOR
        IF NTDSTRANSOPT_BRIDGES_REQUIRED bit is clear in
        t!options and NTDSETTINGS_OPT_W2K3_BRIDGES_REQUIRED bit is
        clear in localSite!options
            APPEND CreateAutoEdgeSet(g, t!objectGUID, L) to g.EdgeSets
        ELSE
            FOR each siteLinkBridge object b that is a child of t
                APPEND CreateEdgeSet(g, t!objectGUID, b) to g.EdgeSets
            ENDFOR
        ENDIF
    ENDFOR
    RETURN g
}
/***** CreateGraph *****/
/* Create a GRAPH instance.
* IN: vertexIDs - Set containing the ID of each vertex to add to the
* graph.
* RETURNS: A new graph containing vertices with the specified IDs.
*/
CreateGraph(IN SEQUENCE<GUID> vertexIDs) : GRAPH
{
    LET g be a new GRAPH
    SORT vertexIDs in ascending order of objectGUID
    FOR each id in vertexIDs
        LET v be a new VERTEX
        SET v.ID to id
        APPEND v to g.Vertices
    ENDFOR
    RETURN g
}
/***** CreateEdge *****/
/* Create a MULTIEDGE instance.
* IN: type - Type of edge to add.
* IN: link - Corresponding siteLink object.
* RETURNS: A new MULTIEDGE instance.
*/
CreateEdge(IN GUID type, IN siteLink link) : MULTIEDGE
{
    LET e be a new MULTIEDGE
    SET e.ID to link!objectGUID

```

```

        SET e.VertexIDs to be the set containing the objectGUID value of
        each site referenced by link!siteList
        SET e.ReplInfo.Cost to link!cost;
        SET e.ReplInfo.Options to link!options
        SET e.ReplInfo.Interval to link!replInterval
        IF link!schedule has a value
            SET e.ReplInfo.Schedule to link!schedule
        ELSE
            SET e.ReplInfo.Schedule to NULL
        EndIF
        SET e.Type to type
        SET e.Directed to FALSE
    RETURN e
}
/***** CreateAutoEdgeSet *****/
/* Create a MULTIEDGESET instance containing edges for all siteLink
* objects.
* INOUT: g - Site graph.
* IN: type - Type of edges being connected.
* IN: L - All siteLink objects.
* RETURNS: A new MULTIEDGESET instance.
*/
CreateAutoEdgeSet(INOUT GRAPH g, IN GUID type,
    IN SET OF siteLink L) : MULTIEDGESET
{
    LET s be a new MULTIEDGESET
    SET s.ID to NULL GUID
    FOR each l in L
        LET e be the edge in g.Edges such that e.ID = l!objectGUID
        IF e.Type = type
            APPEND l!objectGUID to s.EdgeIDs
        ENDIF
    ENDFOR
    RETURN s
}
/***** CreateEdgeSet *****/
/* Create a MULTIEDGESET instance.
* INOUT: g - Site graph.
* IN: type - Type of edges being connected.
* IN: b - Corresponding siteLinkBridge object.
* RETURNS: A new MULTIEDGESET instance.
*/
CreateEdgeSet(INOUT GRAPH g, IN GUID type, IN siteLinkBridge b)
: MULTIEDGESET
{
    LET s be a new MULTIEDGESET
    SET e.ID to b!objectGUID
    FOR each DSNAME l in b!siteLinkList
        LET e be the edge in g.Edges such that e.ID = l!objectGUID
        IF e.Type = type
            APPEND l!objectGUID to s.EdgeIDs
        ENDIF
    ENDFOR
    RETURN s
}
/***** ColorVertices *****/
/* Color each vertex to indicate which kinds of NC replicas it
* contains.

```

```

* INOUT: g - Site graph.
* IN: cr - crossRef for NC.
* IN: detectFailedDCs - TRUE to detect failed DCs and route
      replication traffic around them, FALSE to assume no DC
      has failed.
* RETURNS: TRUE if one or more failed DCs were detected,
*           otherwise FALSE.
*/
ColorVertices(INOUT GRAPH g, IN crossRef cr,
              IN bool detectFailedDCs) : bool
{
    LET foundFailedDCs be FALSE

    FOR each v in g.Vertices
        LET s be the site object with objectGUID v.ID
        IF s contains one or more DCs with full replicas of the NC
            cr!nCName
            SET v.Color to COLOR.RED
        ELSEIF s contains one or more partial replicas of the NC
            SET v.Color to COLOR.BLACK
        ELSE
            SET v.Color to COLOR.WHITE
        ENDIF
    ENDFOR
    LET localSiteVertex be the vertex in graph.Vertices such that
    localSiteVertex.ID = objectGUID of the local DC's site object
    FOR each v in g.Vertices
        FOR each interSiteTransport object t that is a child of the
        CN=Inter-Site Transports child of the CN=Sites child of the
        config NC
            IF localSiteVertex.Color = COLOR.RED and t!name ≠ "IP"
            and FLAG_CR_NTDS_DOMAIN bit is set in cr!systemFlags
                Skip t
            ENDIF
            IF no edge e exists in g.Edges such that e.VertexIDs
            contains v.ID
                Skip t
            ENDIF
            LET partialReplicaOkay be TRUE if and only if
            localSiteVertex.Color = COLOR.BLACK

            LET bh be the result of GetBridgeheadDC(
            localSiteVertex.ID, cr, t, partialReplicaOkay,
            detectFailedDCs)
            IF bh = null
                /* No bridgehead DC is currently available. */
                SET foundFailedDCs to TRUE
                Skip t
            ENDIF
            APPEND t!objectGUID to v.AcceptRedRed
            APPEND t!objectGUID to v.AcceptBlack
        ENDFOR
    ENDFOR

    RETURN foundFailedDCs
}

```

#### 7.2.2.3.4.4 Spanning Tree Computation

The following methods process the site graph and compute the minimum-cost spanning tree.

```

/***** GetSpanningTreeEdges *****/
/* Calculate the spanning tree and return the edges that include the
 * vertex for the local site.
 * INOUT: g - Site graph.
 * OUT: componentCount - Set to the number of graph components
 *      calculated by Kruskal's algorithm. If 1, all sites are
 *      connected by a spanning tree. Otherwise, one or more sites
 *      could not be connected in a spanning tree.
 * RETURNS: Edges that include the vertex for the local site.
 */
GetSpanningTreeEdges(INOUT GRAPH g, OUT int componentCount)
: SET<MULTIEDGE>
{
    /* Phase I: Run Dijkstra's algorithm and build up a list of
     * internal edges, which are really just shortest-paths
     * connecting colored vertices.
     */
    LET internalEdges be an empty sequence of INTERNALEDGE

    FOR each s in g.EdgeSets
        LET edgeType be NULL GUID
        FOR each v in g.Vertices
            REMOVE all items from v.EdgeIDs
        ENDFOR

        FOR each edge e in g.Edges such that s.EdgeIDs contains e.ID
            SET edgeType to e.Type
            FOR each vertex v in g.Vertices such that e.VertexIDs
            contains v.ID
                APPEND e to v.Edges
            ENDFOR
        ENDFOR

        /* Run Dijkstra's algorithm with just the red vertices as
         * the roots */
        CALL Dijkstra(g, edgeType, FALSE)

        /* Process the minimum-spanning forest built by Dijkstra,
         * and add any inter-tree edges to our list of internal
         * edges */
        CALL ProcessEdgeSet(g, s, internalEdges)

        /* Run Dijkstra's algorithm with red and black vertices as
         * the root vertices */
        CALL Dijkstra(g, edgeType, TRUE)

        /* Process the minimum-spanning forest built by Dijkstra,
         * and add any inter-tree edges to our list of internal
         * edges */
        CALL ProcessEdgeSet(g, s, internalEdges)
    ENDFOR

    /* Process the implicit empty edge set */
    CALL SetupVertices(g)
}
```



```

CALL ProcessEdgeSet(g, NULL, internalEdges)

/* Phase II: Run Kruskal's Algorithm on the internal edges. */
LET outputEdges be the result of Kruskal(g, internalEdges)

/* Phase III: Post-process the output:
 * - Traverse tree structure to find one-way black-black edges
 * - Determine the component structure */
FOR each v in g.Vertices
  IF v.Color = COLOR.RED
    SET v.DistToRed to 0
  ELSEIF there exists a path from v to a COLOR.RED vertex
    SET v.DistToRed to the length of the shortest such path
  ELSE
    SET v.DistToRed to MAX DWORD
  ENDIF
ENDFOR
SET componentCount to CountComponents(g)
LET stEdgeList be CopyOutputEdges(g, outputEdges)

RETURN stEdgeList
}

/***** GetBridgeheadDC *****/
/* Get a bridgehead DC.
 * IN: siteObjectGUID - objectGUID of the site object representing
 *     the site for which a bridgehead DC is desired.
 * IN: cr - crossRef for NC to replicate.
 * IN: t - interSiteTransport object for replication traffic.
 * IN: partialReplicaOkay - TRUE if a DC containing a partial
 *     replica or a full replica will suffice, FALSE if only
 *     a full replica will suffice.
 * IN: detectFailedDCs - TRUE to detect failed DCs and route
 *     replication traffic around them, FALSE to assume no DC
 *     has failed.
 * RETURNS: nTDSDSA object for the selected bridgehead DC, or NULL if
 *     none is available.
 */
GetBridgeheadDC(IN GUID siteObjectGUID, IN crossRef cr,
  IN interSiteTransport t, IN bool partialReplicaOkay,
  IN bool detectFailedDCs) : nTDSDSA
{
  LET bhs be the result of GetAllBridgeheadDCs(siteObjectGUID, cr,
  t, partialReplicaOkay, detectFailedDCs)

  IF bhs is empty
    RETURN NULL
  ELSE
    RETURN bhs[0]
  ENDIF
}

/***** GetAllBridgeheadDCs *****/
/* Get all bridgehead DCs satisfying the given criteria.
 * IN: siteObjectGUID - objectGUID of the site object representing
 *     the site for which bridgehead DCs are desired.
 * IN: cr - crossRef for NC to replicate.
 * IN: t - interSiteTransport object for replication traffic.

```

```

* IN: partialReplicaOkay - TRUE if a DC containing a partial
*     replica or a full replica will suffice, FALSE if only
*     a full replica will suffice.
* IN: detectFailedDCs - TRUE to detect failed DCs and route
*     replication traffic around them, FALSE to assume no DC
*     has failed.
* RETURNS: nTDSDSA objects for available bridgehead DCs.
*/
GetAllBridgeheadDCs(IN GUID siteObjectGUID, IN crossRef cr,
    IN interSiteTransport t, IN bool partialReplicaOkay,
    IN bool detectFailedDCs) : SEQUENCE OF nTDSDSA
{
    LET bhs be an empty sequence of nTDSDSA objects
    LET s be the site object such that s!objectGUID = siteObjectGUID
    LET k be an object such that
        s!LDAPDisplayName = nTDSDSA and classSchema in s!objectClass
    LET allDCsInSite be the sequence of objects o that are
        descendants of s such that o!objectCategory = k

    FOR each dc in allDCsInSite
        IF t!bridgeheadServerListBL has one or more values and
            t!bridgeheadServerListBL does not contain a reference to the
            parent object of dc
            Skip dc
        ENDIF

        IF dc is in the same site as the local DC
            IF a replica of cr!nCName is not in the set of NC replicas
                that "should be present" on dc or a partial replica of the
                NC "should be present" but partialReplicasOkay = FALSE
                Skip dc
            ENDIF
        ELSE
            IF an NC replica of cr!nCName is not in the set of NC
                replicas that "are present" on dc or a partial replica of
                the NC "is present" but partialReplicasOkay = FALSE
                Skip dc
            ENDIF
        ENDIF

        IF AmIRODC() and cr!nCName corresponds to default NC then
            Let dsaobj be the nTDSDSA object of the dc
            IF dsaobj.msDS-Behavior-Version < DS_BEHAVIOR_WIN2008
                Skip dc
            ENDIF
        ENDIF

        IF t!name ≠ "IP" and the parent object of dc has no value for
            the attribute specified by t!transportAddressAttribute
            Skip dc
        ENDIF

        IF BridgeheadDCFailed(dc!objectGUID, detectFailedDCs) = TRUE
            Skip dc
        ENDIF

        APPEND dc to bhs
    ENDFOR
}

```

```

    IF bit NTDSSETTINGS_OPT_IS_RAND_BH_SELECTION_DISABLED is set in
    s!options
        SORT bhs such that all GC servers precede DCs that are not GC
        servers, and otherwise by ascending objectGUID
    ELSE
        SORT bhs in a random order
    ENDIF

    RETURN bhs
}

/***** BridgeheadDCFailed *****/
/* Determine whether a given DC is known to be in a failed state.
 * IN: objectGUID - objectGUID of the DC's nTDSDSA object.
 * IN: detectFailedDCs - TRUE if and only if failed DC detection is
 *     enabled.
 * RETURNS: TRUE if and only if the DC should be considered to be in a
 *     failed state.
 */
BridgeheadDCFailed(IN GUID objectGUID, IN bool detectFailedDCs) : bool
{
    IF bit NTDSSETTINGS_OPT_IS_TOPL_DETECT_STALE_DISABLED is set in
    the options attribute of the site settings object for the local
    DC's site
        RETURN FALSE
    ELSEIF a tuple z exists in the kCCFailedLinks or
    kCCFailedConnections variables such that z.UUIDSsa =
    objectGUID, z.FailureCount > 1, and the current time -
    z.TimeFirstFailure > 2 hours
        RETURN TRUE
    ELSE
        RETURN detectFailedDCs
    ENDIF
}

/***** SetupVertices *****/
/* Setup the fields of the vertices that are relevant to Phase I
 * (Dijkstra's Algorithm). For each vertex we set up its cost,
 * root vertex, and component. This defines the shortest-path
 * forest structures.
 * INOUT: graph - Site graph.
 */
SetupVertices(INOUT GRAPH g)
{
    FOR each v in g.Vertices
        IF v.Color = COLOR.WHITE
            SET v.ReplInfo.Cost to MAX DWORD
            SET v.RootID to NULL GUID
            SET v.ComponentID to NULL GUID
        ELSE
            SET v.ReplInfo.Cost to 0
            SET v.RootID to v.ID
            SET v.ComponentID to v.ID
        ENDIF

        SET v.ReplInfo.Interval to 0
        SET v.ReplInfo.Options to 0xFFFFFFFF

```

```

        SET v.ReplInfo.Schedule to NULL
        SET v.HeapLocation to STHEAP_NOT_IN_HEAP
        SET v.Demoted to FALSE
    ENDFOR
}

/***** Dijkstra *****/
/* Run Dijkstra's algorithm with the red (and possibly black) vertices
 * as the root vertices, and build up a shortest-path forest.
 * INOUT: g - Site graph.
 * IN: edgeType - Type of the edges in the current edge set.
 * IN: fIncludeBlack - If this is true, black vertices are also used
 *       as roots.
 */
Dijkstra(INOUT GRAPH g, IN GUID edgeType, IN bool fIncludeBlack)
{
    LET vs be the result of SetupDijkstra(g, edgeType, fIncludeBlack)
    WHILE vs is not empty
        LET c be the least ReplInfo.Cost of any vertex in vs
        LET u be the vertex in vs with the least ID of all
        vertices with ReplInfo.Cost = c
        REMOVE u from vs

        FOR each e in g.Edges such that u.EdgeIDs contains e.ID
            FOR each vertexId in e.VertexIDs
                LET v be the vertex in g.Vertices such that v.ID =
                vertexId
                CALL TryNewPath(g, vs, u, e, v)
            ENDFOR
        ENDFOR
    ENDWHILE
}

/***** SetupDijkstra *****/
/* Build the initial sequence for use with Dijkstra's algorithm. It
 * will contain the red and black vertices as root vertices, unless
 * these vertices accept no edges of the current edgeType, or unless
 * we're not including black vertices.
 * INOUT: g - Site graph.
 * IN: edgeType - Type of the edges in the current edge set.
 * IN: fIncludeBlack - If this is true, black vertices are also used
 *       as roots.
 * RETURNS: Sequence of vertices.
 */
SetupDijkstra(INOUT GRAPH g, IN GUID edgeType, IN bool fIncludeBlack)
: SEQUENCE<VERTEX>
{
    CALL SetupVertices(g)
    LET vs be an empty sequence of VERTEX
    FOR each v in g.Vertices
        IF v.Color = COLOR.WHITE
            Skip v
        ENDIF

        IF (v.Color = COLOR.BLACK and fIncludeBlack = FALSE) or
        v.AcceptBlack does not contain edgeType or v.AcceptRedRed
        does not contain edgeType
            /* If we're not allowing black vertices, or if this vertex

```

```

        * accepts neither red-red nor black edges, then we
        * 'demote' it to a WHITE vertex for the purposes of Phase
        * I. Note that the 'Color' member of the vertex structure
        * is not changed. */
        SET v.ReplInfo.Cost to MAX DWORD
        SET v.RootID to NULL GUID
        SET v.Demoted to TRUE
    ELSE
        APPEND v to vs
    ENDIF
ENDFOR

RETURN vs
}

/***** TryNewPath *****/
/* Helper function for Dijkstra's algorithm. We have found a new path
 * from a root vertex to vertex v. This path is (u->root, ..., u, v).
 * Edge e is the edge connecting u and v. If this new path is better
 * (in our case cheaper, or has a longer schedule), we update v to use
 * the new path.
 * INOUT: g - Site graph.
 * INOUT: vs - Vertices being evaluated.
 * IN: u - Vertex connected by e to v.
 * IN: e - Edge between u and v.
 * INOUT: v - Vertex connected by e to u.
 */
TryNewPath(INOUT GRAPH g, INOUT SEQUENCE<VERTEX> vs, IN VERTEX u,
    IN MULTIEDGE e, INOUT VERTEX v)
{
    LET newRI be an empty REPLINFO
    LET fIntersect be the result of CombineReplInfo(g, u.ReplInfo,
        Edge.ReplInfo, OUT newRI)

    IF newRI.Cost > v->ReplInfo.Cost
        RETURN
    ENDIF

    IF newRI.Cost < v.ReplInfo.Cost and fIntersect = FALSE
        RETURN
    ENDIF

    LET newDuration be the total duration newRI.Schedule shows as
    available
    LET oldDuration be the total duration v.ReplInfo.Schedule shows as
    available

    IF newRI.cost < v.ReplInfo.Cost or newDuration > oldDuration
        /* The new path to v is either cheaper or has a longer
         * schedule. We update v with its new root vertex, cost, and
         * replication info. */
        SET v.RootID to u.RootID
        SET v.ComponentID to u.ComponentID
        SET v.ReplInfo to newRI
        APPEND v to vs
    ENDIF
}

```

```

/***** CombineReplInfo *****/
/* Merge schedules, replication intervals, options and costs.
 * INOUT: g - Site graph.
 * IN: a - Replication info to combine with b.
 * IN: b - Replication info to combine with a.
 * OUT: c - Combination of a and b.
 * RETURNS: TRUE if schedules intersect, FALSE if they don't.
 */
CombineReplInfo(INOUT GRAPH g, IN REPLINFO a, IN REPLINFO b,
  OUT REPLINFO c) : bool
{
  LET s be the schedule that is the intersection of a.Schedule and
  b.Schedule, such that a given time is available in c if and only
  if that time is available in both a.Schedule and b.Schedule

  IF s has no available time
    RETURN FALSE
  ENDIF

  IF a.Cost + b.Cost overflows
    SET c.Cost to MAX DWORD
  ELSE
    SET c.Cost to a.Cost + b.Cost
  ENDIF

  SET C.Interval to maximum of a.Interval and b.Interval
  SET C.Options to a.Options BITWISE-AND b.Options
  SET C.Schedule = s

  RETURN TRUE
}

/***** ProcessEdgeSet *****/
/* After running Dijkstra's algorithm to determine the shortest-path
 * forest, examine all edges in this edge set. We find all inter-tree
 * edges, from which we build the list of 'internal edges', which we
 * will later pass on to Kruskal's algorithm.
 * INOUT: g - Site graph.
 * IN: s - Edge set, or NULL for the implicit edge set with no edges.
 * INOUT: internalEdges - Sequence to which to add new internal edges.
 */
ProcessEdgeSet(INOUT GRAPH g, IN MULTIEDGESET s,
  INOUT SEQUENCE<INTERNALEDGE> internalEdges)
{
  IF s = NULL
    FOR each e in g.Edges
      FOR each v in g.Vertices such that e.VertexIDs contains
        v.ID
        CALL CheckDemoteOneVertex(v, e.Type)
      ENDFOR
      CALL ProcessEdge(g, e, internalEdges)
      FOR each v in g.Vertices such that e.VertexIDs contains
        v.ID
        CALL UndemoteOneVertex(v)
      ENDFOR
    ENDFOR
  ELSE
    FOR each e in g.Edges such s.EdgeIDs contains e.ID

```

```

        CALL ProcessEdge(g, e, internalEdges)
    ENDFOR
ENDIF
}

/***** CheckDemoteOneVertex *****/
/* Demote one vertex if necessary
 * INOUT: v - Vertex to check and possibly demote.
 * IN: edgeType - Type of edge being processed.
 */
CheckDemoteOneVertex(INOUT VERTEX v, IN GUID edgeType)
{
    IF v.Color = COLOR.WHITE
        RETURN
    ENDIF

    IF v.AcceptBlack does not contain edgeType and v.AcceptRedRed does
    not contain edgeType
        /* If this vertex accepts neither red-red nor black edges,
         * then we 'demote' it to a WHITE vertex for the purposes of
         * Phase I. Note that the 'Color' member of the vertex
         * structure is not changed. */
        SET v.ReplInfo.Cost to MAX DWORD
        SET v.RootID to NULL GUID
        SET v.Demoted to TRUE
    ENDIF
}

/**** UndemoteOneVertex *****/
/* Clear the demoted state of a vertex
 * INOUT: v - Vertex to 'undemote'.
 */
UndemoteOneVertex(INOUT VERTEX v)
{
    IF v.Color = COLOR.WHITE
        RETURN
    ENDIF

    SET v.ReplInfo.Cost to 0
    SET v.RootID to v.ID
    SET v.Demoted to FALSE
}

/***** ProcessEdge *****/
/* After running Dijkstra's algorithm, this function examines a
 * multi-edge and adds internal edges between every tree connected by
 * this edge.
 * INOUT: g - Site graph.
 * IN: e - Multi-edge to examine.
 * INOUT: internalEdges - Sequence to which to add any new internal
 * edges.
 */
ProcessEdge(INOUT GRAPH g, IN MULTIEDGE e,
    INOUT SEQUENCE<INTERNALEDGE> internalEdges)
{
    /* Find the best vertex to be the 'root' of this multi-edge. */
    LET vs be a sequence containing each vertex v such that
    e.VertexIDs contains v.ID

```

```

SORT vs such that RED vertices precede BLACK vertices, a vertex
with lower ReplInfo.Cost precedes a vertex with higher
ReplInfo.Cost if both vertices have the same Color, and a vertex
with a lower ID precedes a vertex with higher ID if both vertices
have the same Color and ReplInfo.Cost

LET bestV be vs[0]

/* Add to internalEdges an edge from every colored vertex to
bestV.*/
FOR each vertex v in g.Vertices such that e.VertexIDs contains v
  IF v.ComponentID ≠ NULL GUID and v.RootID ≠ NULL GUID
    Skip v
  ENDIF

  /* Only add this edge if it is a valid inter-tree edge.
  * (The two vertices must be reachable from the root vertices,
  * and in different components.) */
  IF bestV.ComponentID ≠ NULL GUID and bestV.RootID ≠ NULL GUID
  and v.ComponentID ≠ NULL GUID and bestV.RootID ≠ NULL GUID
  and bestV.ComponentID ≠ v.ComponentID
    CALL AddIntEdge(g, internalEdges, e, bestV, v)
  ENDIF
ENDFOR
}

/***** AddIntEdge *****/
/* Add an edge to the list of edges we will process with Kruskal's.
* The endpoints are in fact the roots of the vertices we pass in, so
* the endpoints are always colored vertices.
* INOUT: g - Site graph.
* INOUT: internalEdges - Sequence to which to add the new internal
* edge.
* IN: e - Existing edge being examined.
* IN: v1 - Vertex to connect with new internal edge.
* IN: v2 - Vertex to connect with new internal edge.
*/
AddIntEdge(INOUT GRAPH g, INOUT SEQUENCE<INTERNALEDGE> internalEdges,
  IN MULTIEDGE e, IN VERTEX v1, IN VERTEX v2)
{
  /* The edge we pass on to Kruskal's algorithm actually goes
  * between the roots of the two shortest-path trees. */
  LET root1 be the vertex in g.Vertices such that root1.ID =
  v1.RootID
  LET root2 be the vertex in g.Vertices such that root2.ID =
  v2.RootID

  /* Check if both endpoints will allow this type of edge */
  IF root1.Color = COLOR.RED and root2.Color = COLOR.RED
    LET redRed be TRUE
  ELSE
    LET redRed be FALSE
  ENDIF

  IF redRed = TRUE
    IF root1.AcceptRedRed does not contain e.Type or
    root2.AcceptRedRed does not contain e.Type

```



```

        RETURN
    ENDIF
ELSE
    IF root1.AcceptBlack does not contain e.Type or
    root2.AcceptBlack does not contain e.Type
        RETURN
    ENDIF
ENDIF

/* Combine the schedules of the path from root1 to v1, root2 to
 * v2, and edge e */
LET ri be an empty REPLINFO
LET ri2 be an empty REPLINFO
IF CombineReplInfo(g, v1.ReplInfo, v2.ReplInfo, OUT ri) = FALSE
or CombineReplInfo(g, ri, e.ReplInfo, OUT ri2) = FALSE
    RETURN
ENDIF

/* Set up the internal simple edge from root1 to root2 */
LET newIntEdge be an empty INTERNALEDGE
SET newIntEdge.V1ID to root1.ID
SET newIntEdge.V2ID to root2.ID
SET newIntEdge.RedRed to redRed
SET newIntEdge.ReplInfo to ri2
SET newIntEdge.Type to e.Type

/* Sort newIntEdge's vertices by ID */
IF newIntEdge.V1ID > newIntEdge.V2ID
    Swap newIntEdge.V1ID and newIntEdge.V2ID
ENDIF

IF internalEdges does not contain an INTERNALEDGE that is
identical to newIntEdge
    APPEND newIntEdge to internalEdges
ENDIF
}

/***** Kruskal *****/
/* Run Kruskal's minimum-cost spanning tree algorithm on the internal
 * edges (that represent shortest paths in the original graph between
 * colored vertices).
 * INOUT: g - Site graph.
 * INOUT: internalEdges - Edges between trees.
 * RETURNS: Spanning tree edges for the vertex representing the local
 *          DC's site.
 */
Kruskal(INOUT GRAPH g, INOUT SEQUENCE<INTERNALEDGE> internalEdges)
: SEQUENCE<MULTIEDGE>
{
    FOR each v in g.Vertices
        REMOVE all items from v.EdgeIDs
    ENDFOR

    SORT internalEdges by (descending RedRed, ascending ReplInfo.Cost,
    descending available time in ReplInfo.Schedule, ascending V1ID,
    ascending V2ID, ascending Type)

    LET numExpectedTreeEdges be the count of vertices v in g.Vertices

```

```

such that v.Color = COLOR.RED or v.Color = COLOR.WHITE
LET cSTEdges be 0
LET outputEdges be an empty sequence of MULTIEDGE

WHILE internalEdges is not empty and cSTEdges <
numExpectedTreeEdges
    LET e be internalEdges[0]

    /* We must prevent cycles in the spanning tree. If we are to
    * add edge e, we must ensure its endpoints are in different
    * components. */
    LET comp1 be the return of GetComponentID(g, e.V1ID)
    LET comp2 be the return of GetComponentID(g, e.V2ID)
    IF comp1 ≠ comp2
        /* Add spanning tree edge. */
        INCREMENT cSTEdges by 1
        CALL AddOutEdge(g, outputEdges, e)

        /* Combine the two connected components. */
        LET v be the vertex in g.Vertices such that v.ID = comp1
        SET v.ComponentID to comp2
    ENDIF

    REMOVE e from internalEdges
ENDWHILE

RETURN outputEdges
}

/***** GetComponentID *****/
/* Returns the id of the component containing vertex v by traversing
* the up-tree implied by the component pointers.
* INOUT: g - Site graph.
* INOUT: v - Vertex for which the component ID is desired.
* RETURNS: The component ID of v.
*/
GetComponentID(INOUT GRAPH g, INOUT VERTEX v) : GUID
{
    /* Find root of the up-tree created by component pointers */
    LET u be v
    WHILE u.ComponentID ≠ u.ID
        LET id be u.ComponentID
        SET u to the vertex in g.Vertices such that u.ID = id
    ENDWHILE
    LET root be u.ID

    /* Compress the path to the root */
    SET u to v
    WHILE u.ComponentID ≠ u.ID
        LET id be u.ComponentID
        LET w be the vertex in g.Vertices such that w.ID = id
        SET u.ComponentID to root
        SET u to w
    ENDWHILE

    RETURN root
}

```

```

/***** AddOutEdge *****/
/* We have found a new edge, e, for our spanning tree edge. Add this
 * edge to our list of output edges.
 * INOUT: g - Site graph.
 * INOUT: outputEdges - Sequence to which to add the output edge.
 * IN: e - Edge to add.
 */
AddOutEdge(INOUT GRAPH g, INOUT SEQUENCE<MULTIEDGE> outputEdges,
           IN INTERNALEDGE e)
{
    LET v1 be the vertex in g.Vertices such that v1.ID = e.V1ID
    LET v2 be the vertex in g.Vertices such that v2.ID = e.V2ID

    /* Create an output multi edge */
    LET ee be an empty MULTIEDGE
    SET ee.Directed to FALSE
    APPEND v1.ID to ee.VertexIDs
    APPEND v2.ID to ee.VertexIDs
    SET ee.Type to e.Type
    SET ee.ReplInfo to e.ReplInfo
    APPEND ee to outputEdges

    /* We also add this new spanning-tree edge to the edge lists of
     * its endpoints. */
    APPEND ee to v1.EdgeIDs
    APPEND ee to v2.EdgeIDs
}

/***** CountComponents *****/
/* Count the number of components. A component is considered to be a
 * bunch of colored vertices that are connected by the spanning tree.
 * Vertices whose component id is the same as their vertex id are the
 * root of a connected component.
 *
 * When we find a root of a component, we record its 'component
 * index'. The component indices are a contiguous sequence of numbers
 * that uniquely identify a component.
 *
 * INOUT: g - Site graph.
 * RETURNS: Number of components.
 */
CountComponents(INOUT GRAPH g) : int
{
    LET numComponents be 0
    FOR each v in g.Vertices
        IF v.Color = COLOR.WHITE
            Skip v
        ENDIF

        LET compId be the result of GetComponentID(g, v)
        IF compId = v.ID
            /* It's a component root */
            SET v.ComponentIndex to numComponents
            Increment numComponents by 1
        ENDIF
    ENDFOR

    RETURN numComponents
}

```

```

}

/***** CopyOutputEdges *****/
/* Copy all spanning tree edges from outputEdges that contain the
 * vertex for DCs in the local DC's site.
 * INOUT: g - Site graph.
 * IN: outputEdges - All spanning tree edges.
 * RETURNS: Spanning tree edges for DCs in the local DC's site.
 */
CopyOutputEdges(INOUT GRAPH g, IN SEQUENCE<MULTIEDGE> outputEdges)
: SEQUENCE<MULTIEDGE>
{
    LET s be an empty sequence of MULTIEDGE
    LET vid be the objectGUID of site object for the local DC's site
    FOR each e in outputEdges
        LET v be the vertex in g.Vertices such that v.ID =
            e.VertexIDs[0]
        LET w be the vertex in g.Vertices such that w.ID =
            e.VertexIDs[1]

        IF v.ID = vid or w.ID = vid
            /* Check if this edge meets the criteria of a 'directed
             * edge'. */
            IF (v.Color = COLOR.BLACK or w.Color = COLOR.BLACK) and
                v.DistanceToRed ≠ MAX DWORD
                SET e.Directed to TRUE

                /* Swap the vertices so that e->vertexNames[0] is
                 * closer to a red vertex than e->vertexNames[1]. */
                IF w.DistanceToRed < v.DistanceToRed
                    Swap e.VertexIDs[0] and e.VertexIDs[1]
                ENDIF
            ENDIF

            APPEND e to s
        ENDIF
    ENDFOR

    RETURN s
}

```

#### 7.2.2.3.4.5 nTDSConnection Creation

The following methods create [nTDSConnection](#) objects to "imply" the minimum-cost spanning tree edges for which no [nTDSConnection](#) objects yet exist.

```

/***** CreateConnections *****/
/* Construct an NC replica graph for the NC identified by the given
 * crossRef, then create any additional nTDSConnection objects
 * required.
 *
 * INOUT: g - Site graph.
 * IN: cr - crossRef object for NC.
 * IN: detectFailedDCs - TRUE to detect failed DCs and route
 *      replication traffic around them, FALSE to assume no DC
 *      has failed.
 */

```

```

* INOUT: keepConnections - Sequence to which to add any connections
*   deemed to be "in use".
* OUT: foundFailedDCs - Set to TRUE if one or more failed DCs
*   were detected, otherwise set to FALSE.
* RETURNS: TRUE if the resulting NC replica graph connects
*   all sites that need to be connected.
*/
CreateConnections(INOUT GRAPH g, IN crossRef cr,
  IN bool detectFailedDCs, INOUT SEQUENCE<GUID> keepConnectoins,
  OUT bool foundFailedDCs) : bool
{
  LET connected be a bool, initialized to true
  SET foundFailedDCs to the return of ColorVertices(g, cr,
    detectFailedDCs)

  LET localSiteVertex be the vertex in g.Vertices such that
  localSiteVertex.ID is the objectGUID of the local DC's site object
  IF localSiteVertex.Color = COLOR.WHITE
    /* No NC replicas for this NC in the site of the local DC,
     * so no nTDSConnection objects need be created. */
    return TRUE
  ENDIF

  LET componentCount be an integer
  LET edges be the sequence of MULTIEDGE returned by
  LET stEdgeList be the result of GetSpanningTreeEdges(graph,
    OUT componentCount)

  IF componentCount > 1
    /* Not all sites could be connected by the spanning tree. */
    SET connected to false
  ENDIF

  LET partialReplicaOkay be TRUE if and only if
  localSiteVertex.Color = COLOR.BLACK

  FOR each edge e in stEdgeList
    /* Ignore directed edges not directed to our site. */
    IF e.Directed and e.VertexIDs[1] ≠ localSiteVertex.ID
      Skip e
    ENDIF

    IF e.VertexIDs[0] = localSiteVertex.ID
      LET otherSiteVertex be the vertex in g.Vertices such that
      otherSiteVertex.ID = e.VertexIDs[1]
    ELSE
      LET otherSiteVertex be the vertex in g.Vertices such that
      otherSiteVertex.ID = e.VertexIDs[0]
    ENDIF

    LET t be the interSiteTransport object with objectGUID e.Type
    LET rbh be the result of GetBridgeheadDC(otherSiteVertex.ID,
      cr, t, partialReplicaOkay, detectFailedDCs)
    /* RODC acts as an BH for itself */
    IF AmIRODC() then
      LET lbh be the nTDSDSA object of the local DC
    ELSE
      LET lbh be the result of GetBridgeheadDC(localSiteVertex.ID,

```

```

        cr, t, partialReplicaOkay, detectFailedDCs)
    ENDIF

    LET sched be a new SCHEDULE such that the first available time
    is that of e.ReplInfo.Schedule and each subsequent available
    time is e.ReplInfo.Interval minutes after the previous
    available time

    CALL CreateConnection(cr, rbh, t, lbh, e.ReplInfo, sched,
    partialReplicaOkay)
ENDFOR

RETURN connected
}

/***** CreateConnection *****/
/* Create an nTDSConnection object with the given parameters if one
* does not already exist.
* IN: cr - crossRef object for the NC to replicate.
* IN: rbh - nTDSDSA object for DC to act as the IDL_DRSGetNCChanges
* server (which is in a site other than the local DC's site).
* IN: t - interSiteTransport object for the transport to use for
* replication traffic.
* IN: lbh - nTDSDSA object for DC to act as the IDL_DRSGetNCChanges
* client (which is in the local DC's site).
* IN: ri - Replication parameters (aggregated siteLink options, etc.)
* IN: sch - Schedule specifying the times at which to begin
* replicating.
* IN: detectFailedDCs - TRUE to detect failed DCs and route
* replication traffic around them, FALSE to assume no DC
* has failed.
* IN: partialReplicaOkay - TRUE if bridgehead DCs containing partial
* replicas of the NC are acceptable.
* INOUT: keepConnections - Sequence to which to add any connections
* deemed to be "in use".
*/
CreateConnection(IN crossRef cr, IN nTDSDSA rbh,
    IN interSiteTransport t, IN nTDSDSA lbh, IN REPLINFO ri,
    IN SCHEDULE sch, INOUT SEQUENCE<GUID> keepConnections)
{
    LET rsiteGuid be the objectGUID of the site object ancestor of rbh
    LET lsiteGuid be the objectGUID of the site object ancestor of lbh

    LET rbhsAll be the result of GetAllBridgeheadDCs(rsiteGuid, cr,
    t, partialReplicaOkay, FALSE)
    LET rbhsAvail be the result of GetAllBridgeheadDCs(rsiteGuid, cr,
    t, partialReplicaOkay, detectFailedDCs)
    LET lbhsAll be the result of GetAllBridgeheadDCs(lsiteGuid, cr,
    t, partialReplicaOkay, FALSE)
    LET lbhsAvail be the result of GetAllBridgeheadDCs(lsiteGuid, cr,
    t, partialReplicaOkay, detectFailedDCs)

    FOR each nTDSConnection object cn such that the parent of cn is
    a DC in lbhsAll and cn!fromServer references a DC in rbhsAll
        IF bit NTDSConn_OPT_IS_GENERATED is set in cn!options and
        cn!transportType references t
            IF bit NTDSConn_OPT_USER OWNED SCHEDULE is clear in
            cn!options and cn!schedule ≠ sch

```



```

        IF BridgeheadDCFailed(rguid, detectFailedDCs) = FALSE and
        BridgeheadDCFailed(lguid, detectFailedDCs) = FALSE
            Increment cValidConnections by 1
        ENDIF

        IF keepConnections does not contain cn!objectGUID
            APPEND cn!objectGUID to keepConnections
        ENDIF
    ENDIF
ENDFOR

IF cValidConnections = 0
    LET opt be NTDSCONN_OPT_IS_GENERATED
    IF bit NTDSSITELINK_OPT_USE_NOTIFY is set in ri.Options
        SET bits NTDSCONN_OPT_OVERRIDE_NOTIFY_DEFAULT and
        NTDSCONN_OPT_USE_NOTIFY in opt
    ENDIF
    IF bit NTDSSITELINK_OPT_TWOWAY_SYNC is set in ri.Options
        SET bit NTDSCONN_OPT_TWOWAY_SYNC opt
    ENDIF
    IF bit NTDSSITELINK_OPT_DISABLE_COMPRESSION is set in
    ri.Options
        SET bit NTDSCONN_OPT_DISABLE_INTERSITE_COMPRESSION in opt
    ENDIF

    Perform an originating update to create a new nTDSCConnection
    object cn that is a child of lbh, cn!enabledConnection = TRUE,
    cn!options = opt, cn!transportType is a reference to t,
    cn!fromServer is a reference to rbh, and cn!schedule = sch

    APPEND cn!objectGUID to keepConnections
ENDIF
}

```

#### 7.2.2.4 Removing Unnecessary Connections

This task deletes [nTDSCConnection](#) objects not needed to imply edges in any NC replica graph.

Given an [nTDSCConnection](#) object *cn*, if the DC with the [nTDSDSA](#) object *dc* that is the parent object of *cn* and the DC with the nTDSDA object referenced by *cn!fromServer* are in the same site, the KCC on *dc* deletes *cn* if all of the following are true:

- Bit NTDSCONN\_OPT\_IS\_GENERATED is clear in *cn!options*.
- No site settings object *s* exists for the local DC's site, or bit NTDSSITELINK\_OPT\_IS\_TOPL\_CLEANUP\_DISABLED is clear in *s!options*.
- Another [nTDSCConnection](#) object *cn2* exists such that *cn* and *cn2* have the same parent object, *cn!fromServer* = *cn2!fromServer*, and either
  - *cn!whenCreated* < *cn2!whenCreated*
  - *cn!whenCreated* = *cn2!whenCreated* and *cn!objectGUID* < *cn2!objectGUID*



Given an [nTDSConnection](#) object *cn*, if the DC with the [nTDSDSA](#) object *dc* that is the parent object of *cn* and the DC with the [nTDSDSA](#) object referenced by *cn!fromServer* are in different sites, a KCC acting as an ISTG in *dc*'s site deletes *cn* if all of the following are true:

- Bit NTDSConn\_OPT\_IS\_GENERATED is clear in *cn!options*.
- *cn!fromServer* references an [nTDSDSA](#) object for a DC in a site other than the local DC's site.
- The *keepConnections* sequence returned by *CreateIntersiteConnections()* does not contain *cn!objectGUID*, or *cn* is "superseded by" (see below) another [nTDSConnection](#) *cn2* and *keepConnections* contains *cn2!objectGUID*.
- The return value of *CreateIntersiteConnections()* was true.

An [nTDSConnection](#) *cn* is said to be "superseded by" another [nTDSConnection](#) *cn2* if both of the following are true:

- If *cn* implies a tuple in *r!repsFrom*, *cn2* also implies a tuple in *r!repsFrom*.
- If *s* is (*cn!fromServer*)!*objectGUID* and *t* is (*cn!parent*)!*objectGUID*, *BridgeheadDCFailed(s, true)* = false and *BridgeheadDCFailed(t, true)* = false.

### 7.2.2.5 Connection Translation

This task adjusts values of [repsFrom](#) abstract attributes of NC replicas on the local DC to match those "implied" by [nTDSConnection](#) objects.

If the NTDSDSA\_OPT\_DISABLE\_NTDSConn\_XLATE bit is set in the value of the [options](#) attribute of the local DC's [nTDSDSA](#) object, the KCC skips this task.

First, the KCC inspects *n!repsFrom* for each NC replica *n* that "is present" or "should be present" on the local DC. If *n* is not an NC replica that "should be present" on the local DC, the KCC calls *IDL\_DRSReplicaDel* to remove all tuples from *n!repsFrom* and to remove *n*.

Otherwise, for each tuple *t* in *n!repsFrom*, let *s* be the [nTDSDSA](#) object such that *s!objectGUID* = *t.uuidDsa*. Let *cn* be the [nTDSConnection](#) object such that *cn* is a child of the local DC's [nTDSDSA](#) object and *cn!fromServer* = *s*, or NULL if no such *cn* exists. The KCC calls *IDL\_DRSReplicaDel* to remove *t* from *n!repsFrom* if any of the following is true:

- *cn* = NULL.
- No NC replica of the NC "is present" on *s*.
- A writable replica of the NC "should be present" on the local DC, but a partial replica "is present" on *s*.

If the KCC did not remove *t* from *n!repsFrom*, it updates *t* if necessary to satisfy the following invariants. Such updates are typically required when the *IDL\_DRSGetNCChanges* server has moved from one site to another - for example, to enable compression when the server is moved from the client's site to another site.

- *t.schedule* = *cn!schedule*
- Bit DRS\_PER\_SYNC is set in *t.replicaFlags* if and only if *cn!schedule* has a value *v* that specifies scheduled replication is to be performed at least once per week.
- Bit DRS\_INIT\_SYNC is set in *t.replicaFlags* if and only if *s* and the local DC's [nTDSDSA](#) object are in the same site or *s* is the FSMO role owner of one or more FSMO roles in the NC replica.

- If bit NTDSConn\_OPT\_OVERRIDE\_NOTIFY\_DEFAULT is set in *cn!options*, bit DRS\_NEVER\_NOTIFY is set in *t.replicaFlags* if and only if bit NTDSConn\_OPT\_USE\_NOTIFY is clear in *cn!options*. Otherwise, bit DRS\_NEVER\_NOTIFY is set in *t.replicaFlags* if and only if *s* and the local DC's [nTDSDSA](#) object are in different sites.
- Bit DRS\_USE\_COMPRESSION is set in *t.replicaFlags* if and only if *s* and the local DC's [nTDSDSA](#) object are not in the same site and the NTDSConn\_OPT\_DISABLE\_INTERSITE\_COMPRESSION bit is clear in *cn!options*.
- Bit DRS\_TWOWAY\_SYNC is set in *t.replicaFlags* if and only if bit NTDSConn\_OPT\_TWOWAY\_SYNC is set in *cn!options*.
- Bits DRS\_DISABLE\_AUTO\_SYNC and DRS\_DISABLE\_PERIODIC\_SYNC are set in *t.replicaFlags* if and only if *cn!enabledConnection* = false.
- If *s* and the local DC's [nTDSDSA](#) object are in the same site, *cn!transportType* has no value, or the RDN of *cn!transportType* is CN=IP:
  - Bit DRS\_MAIL\_REP in *t.replicaFlags* is clear.
  - *t.uuidTransport* = NULL **GUID**.
  - *t.uuidDsa* = The **GUID-based DNS name** of *s*.
- Otherwise:
  - Bit DRS\_MAIL\_REP in *t.replicaFlags* is set.
  - If *x* is the object with dsname *cn!transportType*, *t.uuidTransport* = *x!objectGUID*.
  - Let *a* be the attribute identified by *x!transportAddressAttribute*. If *a* is the [dNSHostName](#) attribute, *t.uuidDsa* = the **GUID-based DNS name** of *s*. Otherwise, *t.uuidDsa* = (*s!parent*)!*a*.

Finally, the KCC calls IDL\_DRSReplicaAdd to add a tuple *u* to *n!repsFrom* for each IDL\_DRSGetNCChanges server "implied" by the [nTDSConnection](#) object children of the local DC's [nTDSDSA](#) object if such a *u* does not already exist. For each such [nTDSConnection](#) *cn*, a tuple *u* is implied if all of the following are true:

- *cn!enabledConnection* = true.
- *cn!fromServer* references an [nTDSDSA](#) object.
- An NC replica of the NC "is present" on the DC to which the [nTDSDSA](#) object referenced by *cn!fromServer* corresponds.
- An NC replica of the NC "should be present" on the local DC.
- The NC replica on the DC referenced by *cn!fromServer* is a writable replica or the NC replica that "should be present" on the local DC is a partial replica.
- The NC is not a **domain** NC, the NC replica that "should be present" on the local DC is a partial replica, *cn!transportType* has no value, or *cn!transportType* has an RDN of CN=IP.

If tuple *u* is implied, its fields satisfy each of the criteria defined above for tuple *t* when *t* is updated using IDL\_DRSReplicaModify, plus the following additional criteria:

- *u.uuidDsa* = the [objectGUID](#) of the [nTDSDSA](#) object referenced by *cn!fromServer*.

- *u.uuidInvocId*, *u.usnVec*, *u.consecutiveFailure*, *u.timeLastSuccess*, *u.timeLastAttempt*, and *u.resultLastAttempt* are 0.

If an attempt to contact another DC is made and it fails, the KCC adds a tuple for that DC to the local DC's *kCCFailedConnections* variable.

#### 7.2.2.6 Remove Unneeded kCCFailedLinks and kCCFailedConnections Tuples

This task removes tuples from *kCCFailedLinks* and *kCCFailedConnections* that are not as inputs to future runs.

For each tuple *f* in *kCCFailedLinks*, if *f.FailureCount* = 0 the KCC removes *f*.

For each tuple *k* in *kCCFailedConnections*, if no attempt was made in this run to contact the corresponding DC (the DC with [nTDSDSA](#) object *o* such that *o!objectGUID* = *k.UUIDDsa*) or an attempt was made and it was successful, the KCC removes *k*.

### 7.3 Publishing and Locating a Domain Controller

Active Directory is a distributed service, which means that when a client needs AD services it may be able to receive those services from any of a number of equivalent DCs. Clients cannot be expected to know *a priori* the names of all possible suitable DCs. This implies a need for a protocol by which clients can dynamically discover which DCs are configured, operational, and reachable such that they could supply the needed services, and to choose among those DCs.

Locating a DC works differently for AD/DS than for AD/LDS.

#### ▪ AD/DS

Locating AD/DS DCs is performed in two separate ways, one based on NetBIOS and mailslots, the other based on DNS and **LDAP**. While the network representations of the two ways are radically different, they are functionally very similar. It is worthwhile to explain the conceptual similarities and motivations before starting the detailed discussion of the differing implementation details.

The NetBIOS version is required for compatibility with older clients (such as Windows NT 4.0) that are not aware of Active Directory. Being based on NetBIOS, however, it is dependent either on network **broadcasts** or on the deployment of an **NBNS** infrastructure; broadcasts cannot be used in a wide-area network where they are typically blocked. The DNS based version makes no use of broadcasts and includes extra support for determining network locality.

Both versions of the protocol work in two phases. In the first phase DCs publish data about themselves (in DNS, or in NBNS, or by local configuration of the responder to NetBIOS broadcasts, depending on which version of publication is being used). In the second phase clients look up this static data to determine a set of possible DCs, and then send small messages to some or all of the set, examining the responses in order to determine liveness, reachability, and suitability. Given their conceptual similarity to an ICMP "ping" message, these small messages are referred to as "LDAP ping" and "mailslot ping".

Sections [7.3.1](#) through [7.3.7](#) specify the precise details of both the data servers publish about themselves and the two "ping" protocols.

#### ▪ AD/LDS

An AD/LDS DC that is joined to an AD/DS **domain** publishes itself by creating an object in AD/DS; a client queries AD/DS and selects an AD/LDS DC based on the query results. The information that an AD/LDS DC publishes about itself is described in section [7.3.8](#).

## 7.3.1 Structures and Constants

### 7.3.1.1 NETLOGON\_NT\_VERSION Options Bits

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	V	V	V	V	V	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	V	V	V	V	X	X	X	V
			C	5	5	5	1																G	L	I	P				N	
			S	E	E																		C		P	D				T	
			P	X																						C				4	

**Note** The bits are presented in little-endian byte order.

**V1 (NETLOGON\_NT\_VERSION\_1, 0x00000001):** Unless overridden by **V5**, **V5EX**, or **V5EP**, it instructs the server to respond to **LDAP** "Ping" (section 7.3.3) and Mailslot "Ping" (section 7.3.5) using the NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 structure.

**V5 (NETLOGON\_NT\_VERSION\_5, 0x00000002):** Unless overridden by **V5EX** or **V5EP**, it instructs the server to respond to **LDAP** "Ping" (section 7.3.3) and Mailslot "Ping" (section 7.3.5) using the NETLOGON\_SAM\_LOGON\_RESPONSE structure.

**V5EX (NETLOGON\_NT\_VERSION\_5EX, 0x00000004):** Unless overridden by **V5EP**, it instructs the server to respond to **LDAP** "Ping" (section 7.3.3) and Mailslot "Ping" (section 7.3.5) using the NETLOGON\_SAM\_LOGON\_RESPONSE\_EX structure.

**V5EP (NETLOGON\_NT\_VERSION\_5EX\_WITH\_IP, 0x00000008):** Instructs the server to respond to **LDAP** "Ping" (section 7.3.5) using the NETLOGON\_SAM\_LOGON\_RESPONSE\_EX structure and also return the IP address of the server in the response.

**VCS (NETLOGON\_NT\_VERSION\_WITH\_CLOSEST\_SITE, 0x00000010):** Indicates that the client is querying for the closest site information. This flag is interpreted by DCs with DC functional level greater than or equal to DS\_BEHAVIOR\_WIN2008.

**VNT4 (NETLOGON\_NT\_VERSION\_AVOID\_NT4EMUL, 0x01000000):** Forces the server to respond to **LDAP** "Ping" (section 7.3.3) and to honor all the NetLOGON\_NT\_VERSION options client specifies in the **LDAP** "Ping" (section 7.3.3) or mailslot "Ping" (section 7.3.5). The client specifies NETLOGON\_NT\_VERSION\_AVOID\_NT4EMUL to force the server to respond to **LDAP** "Ping" even if the server is configured to ignore **LDAP** "Ping" requests, and to honor all the NETLOGON\_NT\_VERSION options specified by the client in a mailslot "Ping", even if the server is configured to assume NETLOGON\_NT\_VERSION\_1 in mailslot "Ping" requests.

**VPDC (NETLOGON\_NT\_VERSION\_PDC, 0x10000000):** Indicates that the client is querying for a PDC.

**VIP (NETLOGON\_NT\_VERSION\_IP, 0x20000000):** Obsolete, ignored.

**VL (NETLOGON\_NT\_VERSION\_LOCAL, 0x40000000):** Indicates that the client is the local machine.

**VGC (NETLOGON\_NT\_VERSION\_GC, 0x80000000):** Indicates that the client is querying for a GC.

**X:** Reserved for future expansion. Client must set it to 0, server must ignore it.

### 7.3.1.2 DS\_FLAG Options Bits

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
F	F	F	F	F	F	X	F	X	X	X	F	F	F	F	F	X	X	X	X	X	X	X	X	F	F	F	X	X	X	X	X
C	T	K	D	L	G		P				F	S	N	G	W								F	D	D						
											S	S		T										M	N	S					

**Note** The bits are presented in little-endian byte order.

**FP (DS\_PDC\_FLAG, 0x00000001):** The server holds the **PDC FSMO role** (*PdcEmulationMasterRole*). FSMO roles are introduced in section [3.1.1.1.11](#). Certain updates can be performed only on the holder of the PDC FSMO role. These updates are specified in section [3.1.1.5.1.7](#).

**FG (DS\_GC\_FLAG, 0x00000004):** The server is a **global catalog server**, and will accept and process messages directed to it on the **global catalog** ports (see section [3.1.1.3.1.10](#)).

**FL (DS\_LDAP\_FLAG, 0x00000008):** The server is an **LDAP** server.

**FD (DS\_DS\_FLAG, 0x00000010):** The server is a DC.

**FK (DS\_KDC\_FLAG, 0x00000020):** The server is running the Kerberos Key Distribution Center service.

**FT (DS\_TIMESERV\_FLAG, 0x00000040):** The Win32 Time Service, as specified in [MS-W32T], is present on the server.

**FC (DS\_CLOSEST\_FLAG, 0x00000080):** The server is in the same site as the client. This is a hint to the client that it is well connected to the server in terms of speed.

**FW (DS\_WRITABLE\_FLAG, 0x00000100):** Indicates that the server is not an **RODC**. As explained in section [3.1.1.1.9](#), all **NC replicas** hosted on an RODC do not accept originating writes.

**FGT (DS\_GOOD\_TIMESERV\_FLAG, 0x00000200):** The server is a reliable time server.

**FN (DS\_NDNC\_FLAG, 0x00000400):** The NC is an application NC.

**FSS (DS\_SELECT\_SECRET\_DOMAIN\_6\_FLAG, 0x00000800):** The server is an RODC.

**FFS (DS\_FULL\_SECRET\_DOMAIN\_6\_FLAG, 0x00001000):** The server is a writable DC, not Windows Server 2003 or Windows 2000 Server.

**FDNS (DS\_DNS\_CONTROLLER\_FLAG, 0x20000000):** The server has a DNS name.

**FDM (DS\_DNS\_DOMAIN\_FLAG, 0x40000000):** The NC is a default NC.

**FF (DS\_DNS\_FOREST\_FLAG, 0x80000000):** The NC is the forest root.

**X:** Reserved for future expansion. Server must return 0, client must ignore.

### 7.3.1.3 Operation Code

Operation code set in the request and response of **LDAP** "Ping" (see [7.3.3](#)) or Mailslot "Ping" (see [7.3.5](#)).

Symbolic Name	Value
LOGON_PRIMARY_QUERY	7
LOGON_SAM_LOGON_REQUEST	18
LOGON_SAM_LOGON_RESPONSE	19
LOGON_SAM_PAUSE_RESPONSE	20
LOGON_SAM_USER_UNKNOWN	21
LOGON_SAM_LOGON_RESPONSE_EX	23
LOGON_SAM_PAUSE_RESPONSE_EX	24
LOGON_SAM_USER_UNKNOWN_EX	25

### 7.3.1.4 NETLOGON\_SAM\_LOGON\_REQUEST

The format of a mailslot "Ping" as documented in section Mailslot "Ping" (see [7.3.5](#)).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Opcode																RequestCount															
UnicodeComputerName (variable)																															
...																															
UnicodeUserName (variable)																															
...																															
MailslotName (variable)																															
...																															
AllowableAccountControlBits																															
DomainSidSize																															
DomainSid (variable)																															
...																															
NtVersion																															
LmNtToken																Lm20Token															

**Opcode:** Operation Code (see [7.3.1.3](#)). Set to LOGON\_PRIMARY\_QUERY if PDC is required, set to LOGON\_SAM\_LOGON\_REQUEST for all other cases.

**RequestCount:** A USHORT that contains the number of times the user has repeated this request.

**UnicodeComputerName:** Null-terminated **Unicode** value of the NETBIOS name of the client. This field SHOULD contain at least one character: the null terminator. Each **Unicode** value is encoded as 2 bytes.

**UnicodeUserName:** Null-terminated **Unicode** value of the account name of the user being queried. This field SHOULD contain at least one character: the null terminator. Each **Unicode** value is encoded as 2 bytes.

**MailslotName:** Null-terminated ASCII value of the name of the mailslot the client listens on.

**AllowableAccountControlBits:** Represents the [userAccountControl](#) attribute of an account.

**DomainSidSize:** A DWORD that contains the size of the DomainSid field.

**DomainSid:** The SID of the **domain**, specified as a SID structure, which is defined in [\[MS-DTYP\]](#) section **2.4.2**. Its length is defined in the **DomainSidSize** field.

**NtVersion:** NETLOGON\_NT\_VERSION Options (see [7.3.1.1](#)).

**LmNtToken:** This MUST be set to 0xFF.

**Lm20Token:** This MUST be set to 0xFF.

**Note** There is no padding for alignment. Therefore all fields after MailslotName can occur on odd byte boundaries.

All multi-byte quantities are represented in little-endian.

**7.3.1.5 NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40**

The NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 structure is the server's response to **LDAP** "Ping" (see [7.3.3](#)) or mailslot "Ping" (see [7.3.5](#)).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Opcode																UnicodeLogonServer (variable)															
...																															
UnicodeUserName (variable)																															
...																															
UnicodeDomainName (variable)																															
...																															
NtVersion																															
LmNtToken																Lm20Token															

**Opcode:** Operation Code (see [7.3.1.3](#)).

**UnicodeLogonServer:** Null-terminated **Unicode** value of the NetBIOS name of the server. This field MUST contain at least one character: the null terminator. Each **Unicode** value is encoded as 2 bytes.

**UnicodeUserName:** Null-terminated **Unicode** value of the name of the user copied directly from client's request. This field MUST contain at least one character: the null terminator. Each **Unicode** value is encoded as 2 bytes.



**UnicodeDomainName:** Null-terminated **Unicode** value of the NetBIOS name of the NC. This field MUST contain at least one character: the null terminator. Each **Unicode** value is encoded as 2 bytes.

**NtVersion:** Set to NETLOGON\_NT\_VERSION\_1.

**LmNtToken:** This MUST be set to 0xFF.

**Lm20Token:** This MUST be set to 0xFF.

**Note** All multi-byte quantities are represented in little-endian.

**7.3.1.6 NETLOGON\_SAM\_LOGON\_RESPONSE**

The NETLOGON\_SAM\_LOGON\_RESPONSE structure is the first extended version of the server's response to **LDAP** "Ping" (see [7.3.3](#)) or mailslot "Ping" (see [7.3.5](#)).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Opcode																UnicodeLogonServer (variable)															
...																															
UnicodeUserName (variable)																															
...																															
UnicodeDomainName (variable)																															
...																															
DomainGuid																															
...																															
...																															
...																															
NullGuid																															
...																															
...																															

...	
DnsForestName (variable)	
...	
DnsDomainName (variable)	
...	
DnsHostName (variable)	
...	
DcIpAddress	
Flags	
NtVersion	
LmNtToken	Lm20Token

**Opcode:** Operation Code (see [7.3.1.3](#)).

**UnicodeLogonServer:** Null-terminated **Unicode** value of the NetBIOS name of the server. This field always contains at least one character: the null terminator. Each **Unicode** value is encoded as 2 bytes.

**UnicodeUserName:** Null-terminated **Unicode** value of the name of the user copied directly from client's request. This field always contains at least one character: the null terminator. Each **Unicode** value is encoded as 2 bytes.

**UnicodeDomainName:** Null-terminated **Unicode** value of the NetBIOS name of the NC. This field always contains at least one character: the null terminator. Each **Unicode** value is encoded as 2 bytes.

**DomainGuid:** The value of the NC's GUID attribute specified as a [GUID](#) structure, which is defined in [\[MS-DTYP\]](#) section **2.3.2**.

**NullGuid:** A NULL **GUID**. The **GUID** structure is defined in [\[MS-DTYP\]](#) section **2.3.2**. Always set zero values for all fields in the GUID structure.

**DnsForestName:** Compressed UTF-8 encoded value of the DNS forest name. To get the decompressed string, see [7.3.7](#).

**DnsDomainName:** Compressed UTF-8 encoded value of the DNS NC name. To get the decompressed string, see [7.3.7](#).

**DnsHostName:** Compressed UTF-8 encoded value of the DNS server name. To get the decompressed string, see [7.3.7](#).

**DcIpAddress:** The domain controller IP address, as specified in [\[RFC791\]](#).

**Flags:** DS\_FLAG Options (see [7.3.1.2](#)).

**NtVersion:** Set to NETLOGON\_NT\_VERSION\_5.

**LmNtToken:** This MUST be set to 0xFF.

**Lm20Token:** This MUST be set to 0xFF.

**Note** All multi-byte quantities are represented in little-endian.

**7.3.1.7 NETLOGON\_SAM\_LOGON\_RESPONSE\_EX**

The NETLOGON\_SAM\_LOGON\_RESPONSE\_EX structure is the second extended version of the server's response to **LDAP** "Ping" (see [7.3.3](#)) or mailslot "Ping" (see [7.3.5](#)).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Opcode																Sbz															
Flags																															
DomainGuid																															
...																															
...																															
...																															
DnsForestName (variable)																															
...																															
DnsDomainName (variable)																															
...																															
DnsHostName (variable)																															
...																															

NetbiosDomainName (variable)	
...	
NetbiosComputerName (variable)	
...	
UserName (variable)	
...	
DcSiteName (variable)	
...	
ClientSiteName (variable)	
...	
DcSockAddrSize	DcSockAddr (variable)
...	
NextClosestSiteName (variable)	
...	
NtVersion	
LmNtToken	Lm20Token

**Opcode:** Operation Code (see [7.3.1.3](#)).

**Sbz:** This MUST be set to 0.

**Flags:** DS\_FLAG Options (see [7.3.1.2](#)).

**DomainGuid:** The value of the NC's GUID attribute specified as a [GUID](#) structure, which is defined in [\[MS-DTYP\]](#) section **2.3.2**.

**DnsForestName:** Compressed UTF-8 encoded value of the DNS name of the forest. To get the decompressed string, see [7.3.7](#).

**DnsDomainName:** Compressed UTF-8 encoded value of the DNS name of the NC. To get the decompressed string, see [7.3.7](#).

**DnsHostName:** Compressed **UTF-8** encoded value of the DNS name of the server. To get the decompressed string, see [7.3.7](#).

**NetbiosDomainName:** Compressed UTF-8 encoded value of the NetBIOS name of the NC. To get the decompressed string, see [7.3.7](#).

**NetbiosComputerName:** Compressed UTF-8 encoded value of the NetBIOS name of the server. To get the decompressed string, see [7.3.7](#).

**UserName:** Compressed UTF-8 encoded value of the user specified in the client's request. To get the decompressed string, see [7.3.7](#).

**DcSiteName:** Compressed UTF-8 encoded value of the site name of the server. To get the decompressed string, see [7.3.7](#).

**ClientSiteName:** Compressed UTF-8 encoded value of the site name of the client. To get the decompressed string, see [7.3.7](#).

**DcSockAddrSize:** A CHAR that contains the size of the server's IP address. This field is included only if the client specifies NETLOGON\_NT\_VERSION\_5EX\_WITH\_IP in the request.

**DcSockAddr:** The domain controller IP address, as specified in [\[RFC791\]](#). This field is included only if the client specifies NETLOGON\_NT\_VERSION\_5EX\_WITH\_IP in the request.

**NextClosestSiteName:** This field is included only if the client specifies NETLOGON\_NT\_VERSION\_WITH\_CLOSEST\_SITE in the request, and if the responding DC has DC functional level DS\_BEHAVIOR\_WIN2008 or greater. When included, NextClosestSiteName contains the name of the site that is closest by cost to ClientSiteName, without being equal to it. The site name is compressed UTF-8 encoded. To get the decompressed string, see section [7.3.7](#).

**NtVersion:** NETLOGON\_NT\_VERSION\_5EX.

**LmNtToken:** This MUST be set to 0xFF.

**Lm20Token:** This MUST be set to 0xFF.

**Note** All multi-byte quantities are represented in little-endian.

## 7.3.2 DNS Record Registrations

### 7.3.2.1 SRV Records Registered by DC

The **SRV DNS Resource Record** for specifying the location of services is specified in [\[RFC2782\]](#). A **SRV record** maps the name of a service to the DNS name of a server that offers that service.

The name of a SRV Resource Record is in the following form:

- Service.Proto.Name TTL Class SRV Priority Weight Port Target

A client queries for these records by sending a DNS SRV query [\[RFC2782\]](#) to a DNS server.

If a server is a DC with default NC X (and NC X's **GUID** is G) in site Y and in forest Z then it registers SRV records with Service.Proto.Name equal to:

```
_ldap_tcp.X  
_ldap_tcp.Y._sites.X  
_ldap_tcp.dc._msdcs.X  
_ldap_tcp.Y._sites.dc._msdcs.X  
_ldap_tcp.G.domains._msdcs.Z  
_kerberos_tcp.X  
_kerberos_udp.X  
_kerberos_tcp.Y._sites.X  
_kerberos_tcp.dc._msdcs.X  
_kerberos_tcp.Y._sites.dc._msdcs.X  
_kpasswd_tcp.X  
_kpasswd_udp.X
```

If the server is also a non-RODC GC server then it registers SRV records with Service.Proto.Name equal to:

```
_ldap_tcp.gc._msdcs.Z  
_ldap_tcp.Y._sites.gc._msdcs.Z  
_gc_tcp.Z  
_gc_tcp.Y._sites.Z
```

If the server is a RODC GC server then it registers SRV records with Service.Proto.Name equal to

```
_ldap_tcp.Y._sites.gc._msdcs.Z  
_gc_tcp.Y._sites.Z
```

If the server is also the PDC of its default NC then it registers SRV records with Service.Proto.Name equal to:

```
_ldap_tcp.pdc._msdcs.X
```

If the server also hosts application NCs, then for each application NC  $A_i$ , it registers SRV records with Service.Proto.Name equal to:

```
_ldap_tcp.Ai  
_ldap_tcp.Y._sites.Ai
```

Example: If a DC with default NC:

```
X = na.fabricom.com
```

is in site:

```
Y = site1
```

and forest:

```
Z = fabricom.com
```

and NC X's GUID is:

```
G = 52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8
```

then its record of type `_ldap._tcp.Y._sites.dc._msdcs.X` has:

```
Service.Proto.Name =  
_ldap._tcp.site1._sites.dc._msdcs.na.fabricom.com
```

and its record of type `_ldap._tcp.G.domains._msdcs.Z` has:

```
Service.Proto.Name =  
_ldap._tcp.52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8.domains._msdcs.fabricom.com
```

The following table describes the other fields of each SRV record registered by a server.

Field	Value
TTL	Set to 600 seconds and is configurable. (The way in which the server is configured is outside the state model and is implementation-dependent.)
Class	Set to IN.
SRV	Set to SRV.
Priority	Set to 0 and is configurable. (The way in which the server is configured is outside the state model and is implementation-dependent.)
Weight	Set to 100 and is configurable. (The way in which the server is configured is outside the state model and is implementation-dependent.)
Port	Set to 389 for <b>LDAP</b> service. Set to 3268 for GC service. Set to 88 for Kerberos KDC service. Set to 464 for Kerberos Password Change service.
Target	Set to the fully qualified DNS name of the server.

### 7.3.2.2 Non-SRV Records Registered by a DC

In addition to SRV records, a DC also registers CNAME [RFC1034] and type A [RFC1034] DNS records.

A CNAME record acts as an alias for a DNS hostname and has the following form

- Name TTL class type RDATA

A client queries for these records by sending a DNS A, CNAME, or \* query [RFC1034] to a DNS server.

If a server is a DC in forest Z, and its **DSA GUID** is G, then the server registers a CNAME record with Name field set to `G._msdcs.Z`. This name is called the DC's **GUID**-based DNS name.

Example: If a DC is in forest:

```
Z = fabricom.com
```

and its DSA **GUID** is:

```
G = 52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8
```

then it registers a CNAME record with:

```
Name =  
52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8._msdcs.fabricom.com
```

The following table describes the other fields of each CNAME record registered by a server.

Field	Value
TTL	Set to 600 seconds and is configurable. (The way in which the server is configured is outside the state model and is implementation-dependent.)
Class	Set to IN.
Type	Set to CNAME.
RDATA	Set to the fully qualified DNS name of the server.

A Type A record associates an IP address with a name, and takes the form

- Name TTL class type RDATA

A client queries for these records by sending a DNS A or \* query [RFC1034] to a DNS server.

If a server is a DC with default NC X in forest Z, then it publishes a type A record with Name field X. If the DC is a GC server, it also publishes a type A record with Name field gc.\_msdcs.Z.

Example: If a DC has default NC:

```
X = na.fabricam.com
```

and is in forest:

```
Z = fabricom.com
```

then it registers a type A record with:

```
Name = na.fabricam.com
```

If the DC is a GC server, it registers a type A record with:



Name = gc.\_msdcs.fabricom.com

The following table describes the other fields of each type A record registered by a server.

Field	Value
TTL	Set to 600 seconds and is configurable. (The way in which the server is configured is outside the state model and is implementation-dependent.)
Class	Set to IN.
Type	Set to A.
RDATA	Set to the IP address of the server used for DC functions.

### 7.3.3 LDAP "Ping"

This topic describes the usage of the Lightweight Directory Access Protocol to verify the aliveness of the domain controller and also check whether the domain controller matches a specific set of requirements. This operation is commonly referred to as "LDAP Ping".

An **LDAP** rootDSE search (section [3.1.1.3.2](#)) that retrieves the attribute "NetLogon" (NetLogon is not the name of an actual attribute in the schema) triggers the following processing on the server.

The **LDAP** search filter [RFC2254] included in the SearchRequest is a one level AND of equality test of below elements.

Each element is described below:

**DnsDomain:** The DNS name of an NC (default NC or application NC).

**Host:** The NetBIOS name of the client.

**User:** An AccountName of some account in the **domain** specified by DnsDomain, or DomainSid or DomainGuid.

**AAC:** Represents the [userAccountControl](#) attribute of an account.

**DomainSid:** The SID of a **domain**.

**DomainGuid:** The **GUID** of a **domain**.

**NtVer:** NETLOGON\_NT\_VERSION Options (see section [7.3.1.1](#)).

Example:

```
(&(DnsDomain=abcde.corp.microsoft.com)(Host=abcdefgh-dev)(User=abcdefgh-dev$)(AAC=\80\00\00\00)(DomainGuid=\3b\b0\21\ca\d3\6d\d1\11\8a\7d\b8\df\b1\56\87\1f)(NtVer=\06\00\00\00))
```

Network payload:

```
A0 84 00 00 00 A8 A3 84 00 00 00 25 04 09 44      ?...`£?...%.D
6E 73 44 6F 6D 61 69 66 04 18 61 62 63 64 65      nsDomain..abcde
2E 63 6F 72 70 2E 6D 69 63 72 6F 73 6F 66 74      .corp.microsoft
2E 63 6F 6D A3 84 00 00 00 14 04 04 48 6F 73      .com£?.....Hos
```

74 04 0C 61 62 63 64 65 66 67 68 2D 64 65 76	t..abcdefgh-dev
A3 84 00 00 00 15 04 04 55 73 65 72 04 0D 61	£?.....User..a
62 63 64 65 66 67 68 2D 64 65 76 24 A3 84 00	bcdefgh-dev\$£?.
00 00 0B 04 03 41 41 43 04 04 80 00 00 00 A3	.....AAC...?...£?
84 00 00 00 1E 04 0A 44 6F 6D 61 69 6E 47 75	.....DomainGu
69 64 04 10 3B B0 21 CA D3 6D D1 11 8A 7D B8	id...;°!ÊÔmN.?.},
DF B1 56 87 1F A3 84 00 00 00 0D 04 05 4E 74	ß±V?..£?.....Nt
56 65 72 04 04 06 00 00 00 30 84 00 00 00 0A	Ver.....0?....
04 08 6E 65 74 6C 6F 67 6F 6E	..netlogon

### 7.3.3.1 Syntactic Validation of the Filter

If any of the elements is specified more than once, then the filter is invalid.

If the value of the string passed with DomainGuid has a different size than the size of **GUID** ([\[MS-DTYP\]](#) section **2.3.2**), then the filter is invalid.

If the numeric value of the string passed with AAC is longer than the largest unsigned integer that can be represented in a DWORD, then the filter is invalid.

If the numeric value of the string passed with NtVer is longer than the largest unsigned integer that can be represented in a DWORD, then the filter is invalid.

The response of the dc for invalid filter case is documented in section [7.3.3.3](#).

### 7.3.3.2 Domain Controller Response to an LDAP "Ping"

Let reqDnsNC be the NC replica (full or partial) hosted by the server:

- If the filter does not include the (DnsDomain=dnsDomain) clause, reqDnsNC is set to NULL.
- If the filter includes the (DnsDomain=dnsDomain) clause, if dnsDomain is empty, the response of the dc is documented in section [7.3.3.3](#). If there is no NC hosted by the server whose DNS name is dnsDomain, the response of the dc is documented in section [7.3.3.3](#). Otherwise, reqDnsNC is set to dnsDomain.

Let reqGuidNC be the **GUID** of the NC replica (full or partial) hosted by the server:

- If the filter does not include the (DomainGuid=domainGuid) clause, reqGuidNC is set to NULL.
- If the filter includes the (DomainGuid=domainGuid) clause, if domainGuid is not a valid **GUID**, the response of the dc is documented in section [7.3.3.3](#). If there is no NC hosted by the server whose Guid is domainGuid, the response of the dc is documented in section [7.3.3.3](#). Otherwise, reqGuidNC is set to domainGuid.

Let reqSidNC be the sid of the NC replica (full or partial) hosted by the server:

- If the filter does not include the (DomainSid=domainSid) clause, reqSidNC is set to NULL.
- If the filter includes the (DomainSid=domainSid) clause, if domainSid is not a valid sid, the response of the dc is documented in section [7.3.3.3](#). If there is no NC hosted by the server whose Sid is domainSid, the response of the dc is documented in section [7.3.3.3](#). Otherwise, reqSidNC is set to domainSid.

If both reqDnsNC and reqGuidNC are NULL, the response of the dc is documented in section [7.3.3.3](#).

When reqDnsNC and reqGuidNC are not NULL, if reqGuidNC is not equal to the Guid of NC reqDnsNC, the response of the dc is documented in section [7.3.3.3](#).

When reqDnsNC and reqSidNC are not NULL, if reqSidNC is not equal to the Sid of NC reqDnsNC, the response of the dc is documented in section [7.3.3.3](#).

When reqGuidNC and reqSidNC are not NULL, if reqSidNC is not equal to the Sid of NC reqGuidNC, the response of the dc is documented in section [7.3.3.3](#).

If all three of reqDnsNC, reqSidNC, reqGuidNC are NULL, let reqDnsNC equal the default NC of the server.

Let nc be the NC designated by whichever of reqDnsNC and reqGuidNC that is not NULL.

If client specifies a User value, let u equal to the supplied value. If a User value is not specified, let u equal to NULL.

- Let x be the object in nc such that x![sAMAccountName](#) = u.
- If no such object exists in nc, then let x equal to NULL.
- If x <> NULL, do the following four checks:
  - If AAC is supplied in the search filter, let aac equal the supplied value, otherwise, let aac equal to 0.
  - Let **uac** equal x![userAccountControl](#).
  - If uac has USER\_ACCOUNT\_DISABLED ([MS-SAMR] section [2.2.1.12](#)) bit set, then let x equal to NULL.
  - If (aac & uac & (USER\_TEMP\_DUPLICATE\_ACCOUNT | USER\_NORMAL\_ACCOUNT | USER\_INTERDOMAIN\_TRUST\_ACCOUNT | USER\_WORKSTATION\_TRUST\_ACCOUNT | USER\_SERVER\_TRUST\_ACCOUNT [MS-SAMR] section 2.2.1.12) is zero, then let x equal to NULL. The effect of doing this is so that server only checks USER\_TEMP\_DUPLICATE\_ACCOUNT | USER\_NORMAL\_ACCOUNT | USER\_INTERDOMAIN\_TRUST\_ACCOUNT | USER\_WORKSTATION\_TRUST\_ACCOUNT | USER\_SERVER\_TRUST\_ACCOUNT bits.

Let s be the site containing the client's IP address (see [7.1](#)). If there is no such site, let s equal to NULL.

Let v be the NtVer requested by the client in the search filter.

- If the server is configured to respond to "ping" requests in the form of a NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 structure (the way in which the server is configured is outside the state model and is implementation-dependent), and v does not have NETLOGON\_NT\_VERSION\_AVOID\_NT4EMUL bit set, the response of the dc is documented in "Response to Invalid Filter" (section [7.3.3.3](#)).
- Else if v has NETLOGON\_NT\_VERSION\_5EX or NETLOGON\_NT\_VERSION\_5EX\_WITH\_IP bit set, server uses structure NETLOGON\_SAM\_LOGON\_RESPONSE\_EX to send response back.
- Else if v has NETLOGON\_NT\_VERSION\_5 bit set, server uses structure NETLOGON\_SAM\_LOGON\_RESPONSE to send response back.
- For all other cases, server uses structure NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 to send response back.

Let t be 0.

- When NetLogon service is in paused state, if v does not have NETLOGON\_NT\_VERSION\_PDC bit set or server is not a PDC, let t be 1.
- If the value of rootDSE attribute isSynchronized (see [3.1.1.3](#)) is false, let t be 1.
- When NetLogon **RPC** server is not initialized, if v does not have NETLOGON\_NT\_VERSION\_LOCAL bit set, let t be 1.
- If **FRS** service is in paused state, let t be 1.

After normal processing, the server returns an **LDAP** SearchResultEntry to the client with the following form:

- The ObjectName of the SearchResultEntry is NULL and the attribute list contains one attribute. This attribute is named "Netlogon" and its value is little-endian octet string packed in NETLOGON\_SAM\_LOGON\_RESPONSE\_EX, or NETLOGON\_SAM\_LOGON\_RESPONSE, or NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 depending on value v.
- If the server uses NETLOGON\_SAM\_LOGON\_RESPONSE\_EX to pack the value, it does the following:

**OperationCode:** Set to LOGON\_SAM\_PAUSE\_RESPONSE\_EX if t is equal to 1. Set to LOGON\_SAM\_USER\_UNKNOWN\_EX if u is not NULL, but x is NULL. Set to LOGON\_SAM\_LOGON\_RESPONSE\_EX in other cases.

**Flags:**

Bit values are taken from DS\_FLAGS in section [7.3.1.2](#).

- If the server holds the PDC FSMO role (see section [3.1.1.1.11](#)), the DS\_PDC\_FLAG bit is set.
- If the server is a global catalog server, the DS\_GC\_FLAG bit is set. This bit is set if and only if the isGlobalCatalogReady attribute on the rootDSE is true (see section [3.1.1.3.2.10](#)).
- If the server is a KDC, the DS\_KDC\_FLAG bit is set.
- If the server hosts the Win32 Time Service, as specified in [MS-W32T], the DS\_TIMESERV\_FLAG bit is set.
- If the server is in the same site as the client, the DS\_CLOSEST\_FLAG bit is set.
- If the server is not an RODC, the DS\_WRITABLE\_FLAG bit is set. [MS-DRSR] section [5:AmIRODC](#) explains how to determine if a **DC** is an RODC.
- If the server is configured to be a reliable time source (the way in which the configuration can be done is outside the scope of the state model and is implementation-dependent), the DS\_GOOD\_TIMESERV\_FLAG bit is set.
- If the DnsDomain value specified in the search filter is an application NC, the DS\_NDNC\_FLAG bit is set.
- If the server is an RODC, the DS\_SELECT\_SECRET\_DOMAIN\_6\_FLAG bit is set.
- If the server is a writable DC and not running Windows Server 2003 or Windows 2000 Server, the DS\_FULL\_SECRET\_DOMAIN\_6\_FLAG bit is set.

- If the server has a DNS name, the DS\_DNS\_CONTROLLER\_FLAG bit is set.
- If the DnsDomain value specified in the search filter is the DNS name of the default NC, the DS\_DNS\_DOMAIN\_FLAG bit is set.
- If the DnsDomain value specified in the search filter is the forest name, the DS\_DNS\_FOREST\_FLAG bit is set.
- Always set the DS\_LDAP\_FLAG and DS\_DS\_FLAG bits.
- All the other bits of DS\_FLAG are set to 0.

**DomainGuid:** Set to Guid of NC nc.

**DnsForestName:** Set to the DNS name of the forest.

**DnsDomainName:** Set to the DNS name of the NC nc.

**DnsHostName:** Set to the DNS name of the server.

**NetbiosDomainName:** Set to the NetBIOS name of the NC nc.

**NetbiosComputerName:** Set to the NetBIOS name of the server.

**UserName:** Set to u.

**DcSiteName:** Set to the site name of the server.

**ClientSiteName:** Set to the site s.

**DcSockAddrSize:** Set to the size of server's IP address.

**SockAddr:** Set to server's IP address of the server.

**NextClosestSiteName:** If v has NETLOGON\_NT\_VERSION\_WITH\_CLOSEST\_SITE and the DC has DC functional level DS\_BEHAVIOR\_WIN2008 or greater, use IDL\_DRSQuerySitesByCost ([MS-DRSR] section [4.1.16](#)) to find the site C that is closest to ClientSiteName but not equal to ClientSiteName, and set this field to C. Otherwise omit this field.

**NtVersion:** If the NextClosestSiteName field is set, set this field to {NETLOGON\_NT\_VERSION\_WITH\_CLOSEST\_SITE, NETLOGON\_NT\_VERSION\_5EX}, otherwise set this field to {NETLOGON\_NT\_VERSION\_5EX}.

**LmNtToken:** Always set to 0xFF.

**Lm20Token:** Always set to 0xFF.

- If the server uses NETLOGON\_SAM\_LOGON\_RESPONSE to pack the value, it does the following:

**OperationCode:** Set to LOGON\_SAM\_PAUSE\_RESPONSE if t is equal to 1. Set to LOGON\_SAM\_USER\_UNKNOWN if u is not NULL, but x is NULL. Set to LOGON\_SAM\_LOGON\_RESPONSE in other cases.

**UnicodeLogonServer:** Set to the NetBIOS name of the server.

**UnicodeUserName:** Set to u.

**UnicodeDomainName:** Set to the NetBIOS name of the domain.

**DomainGuid:** Set to the **GUID** of the domain.

**SiteGuid:** Always set to NULL **GUID**.

**DnsForestName:** Set to the DNS name of the forest.

**DnsDomainName:** Set to the DNS name of the domain.

**DnsHostName:** Set to the DNS name of the server.

**DcIpAddress:** Set to the IP address of the server.

**Flags:** If server is a PDC, bit DS\_PDC\_FLAG is set; bit DS\_DS\_FLAG is always set; all the other bits of DS\_FLAG are set to 0.

**NtVersion:** Set to NETLOGON\_NT\_VERSION\_5.

**LmNtToken:** Always set to 0xFF.

**Lm20Token:** Always set to 0xFF.

- If server uses NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 to pack the value, it does the following:

**OperationCode:** Set to LOGON\_SAM\_PAUSE\_RESPONSE if t is 1. Set to LOGON\_SAM\_USER\_UNKNOWN if u is not NULL, but x is NULL. Set to LOGON\_SAM\_LOGON\_RESPONSE in other cases.

**UnicodeLogonServer:** Set to the NetBIOS name of the server.

**UnicodeUserName:** Set to u.

**UnicodeDomainName:** Set to the NetBIOS name of the domain.

**NtVersion:** Set to NETLOGON\_NT\_VERSION\_1.

**LmNtToken:** Always set to 0xFF.

**Lm20Token:** Always set to 0xFF.

LdapResult of SearchResultDone entry is set to 0 (*success*).

### 7.3.3.3 Response to Invalid Filter

If the filter is not syntactically valid for any of cases specified above, the server returns an **LDAP** SearchResultEntry with the following form:

The ObjectName of the SearchResultEntry is NULL. Attribute of SearchResultEntry is NULL. And LdapResult of SearchResultDone entry is set to 0 (*success*).

### 7.3.4 NetBIOS Broadcast and NBNS Background

If a server is in a **domain** whose NetBIOS name is d, it registers <d>[1C] records, and <d>[1B] records if it is a PDC, to the NBNS(WINS) server. A client can retrieve those records by either broadcasting or querying against NBNS(WINS) directly.

For more information, see [RFC1001] and [RFC1002].

### 7.3.5 Mailslot "Ping"

This topic describes the usage of mailslot message to verify the aliveness of the domain controller and also check whether that domain controller matches a specific set of requirements. This operation is commonly referred to as "mailslot ping".

Server listens on the \\mailslot\net\netlogon mailslot, as specified in [\[MS-MAIL\]](#); it interprets the message received as structure NETLOGON\_SAM\_LOGON\_REQUEST and does the following processing.

If Opcode is set to LOGON\_PRIMARY\_QUERY and server is not a PDC, the dc just ignores the message without sending a response back to the client.

If DomainSidSize is not zero, it checks whether the default NC has the same Sid, if it does not, the server ignores the message without sending a response back to the client.

If UnicodeUserName is specified, it is processed in the same way as User value in section [7.3.3.2](#).

Let *v* be the NtVer requested by the client in the request.

If the server is configured to respond to "ping" requests in the form of a NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 structure (the way in which the server is configured is outside the state model and is implementation-dependent), and *v* does not have NETLOGON\_NT\_VERSION\_AVOID\_NT4EMUL bit set, server uses structure NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 to send response back.

Else if *v* has NETLOGON\_NT\_VERSION\_5 bit set, server uses structure NETLOGON\_SAM\_LOGON\_RESPONSE to send response back.

For all other cases, server uses structure NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 to send response back.

Let *t* be 0.

- When NetLogon service is in paused state, if *v* does not have NETLOGON\_NT\_VERSION\_PDC bit set or server is not a PDC, let *t* be 1.
- If the value of rootDSE attribute isSynchronized (see [3.1.1.3](#)) is false, let *t* be 1.
- When NetLogon **RPC** server is not initialized, if *v* does not have NETLOGON\_NT\_VERSION\_LOCAL bit set, let *t* be 1.
- If FRS service is in paused state, let *t* be 1.

Then, the server sends a response back to the mailslot named in the client's request. The response message is packed in the structure NETLOGON\_SAM\_LOGON\_RESPONSE or NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 depending on the value of *v*.

- If server uses NETLOGON\_SAM\_LOGON\_RESPONSE to pack the value, it does the following:

**OperationCode:** Set to LOGON\_SAM\_PAUSE\_RESPONSE if *t* is equal to 1. Set to LOGON\_SAM\_USER\_UNKNOWN if UnicodeUserName is not NULL, but *x* is NULL. Set to LOGON\_SAM\_LOGON\_RESPONSE in other cases.

**UnicodeLogonServer:** Set to the NetBIOS name of the server.

**UnicodeUserName:** Set to UnicodeUserName filed in the request NETLOGON\_SAM\_LOGON\_REQUEST message.

**UnicodeDomainName:** Set to the NetBIOS name of the domain.

**DomainGuid:** Set to the **GUID** of the domain.

**SiteGuid:** Always set to NULL **GUID**.

**DnsForestName:** Set to the DNS name of the forest.

**DnsDomainName:** Set to the DNS name of the domain.

**DnsHostName:** Set to the DNS name of the server.

**DcIpAddress:** Set to the IP address of the server.

**Flags:** If server is a PDC, bit DS\_PDC\_FLAG is set; bit DS\_DS\_FLAG is always set; all the other bits of DS\_FLAG are set to 0.

**NtVersion:** Set to NETLOGON\_NT\_VERSION\_5.

**LmNtToken:** Always set to 0xFF.

**Lm20Token:** Always set to 0xFF.

- If server uses NETLOGON\_SAM\_LOGON\_RESPONSE\_NT40 to pack the value, it does the following:

**OperationCode:** Set to LOGON\_SAM\_PAUSE\_RESPONSE if t is equal to 1. Set to LOGON\_SAM\_USER\_UNKNOWN if UnicodeUserName is not NULL, but x is NULL. Set to LOGON\_SAM\_LOGON\_RESPONSE in other cases.

**UnicodeLogonServer:** Set to the NetBIOS name of the server.

**UnicodeUserName:** Set to UnicodeUserName filed in the request NETLOGON\_SAM\_LOGON\_REQUEST message.

**UnicodeDomainName:** Set to the NetBIOS name of the domain.

**NtVersion:** Set to NETLOGON\_NT\_VERSION\_1.

**LmNtToken:** Always set to 0xFF.

**Lm20Token:** Always set to 0xFF.

### 7.3.6 Locate a Domain Controller

There are two ways to locate a domain controller: DNS-based discovery and NetBIOS-based discovery.

If the name specified by the user is a dns name, the client uses DNS-based discovery; if the name is NETBIOS name, the client uses NETBIOS-based discovery.

If DNS-based discovery is used, the client first does DNS Query for the SRV records. If the client wants to find:

- A **LDAP** server hosts NC N, it searches for record named \_ldap\_tcp.N.
- A **LDAP** server hosts NC N in site Y, it searches for record named \_ldap\_tcp.Y.\_sites.N.
- A DC hosts NC N, it searches for record named \_ldap\_tcp.dc.\_msdcs.N.



- A DC hosts NC N in site Y, it searches for record named `_ldap._tcp.Y._sites.dc._msdcs.N`.
- A DC hosts default NC X whose Guid is G and in forest Z, it searches for record named `_ldap._tcp.G.domains._msdcs.Z`.
- A DC that hosts default NC X and is also a PDC, it searches for record named `_ldap._tcp.pdc._msdcs.X`.
- A GC in forest Z, it searches for record named `_gc._tcp.Z`.
- A GC in forest Z in site Y, it searches for record named `_gc._tcp.Y._sites.Z`.
- A server that is running the Kerberos Key Distribution Center service over TCP for default NC X, it searches for record named `_kerberos._tcp.X`.
- A server that is running the Kerberos Key Distribution Center service over UDP for default NC X, it searches for record named `_kerberos._udp.X`.
- A server that is running the Kerberos Key Distribution Center service over TCP for default NC X and in site Y, it searches for record named `_kerberos._tcp.Y._sites.X.`
- A DC that is running the Kerberos Key Distribution Center service over TCP and also hosts default NC X, it searches for record named `_kerberos.tcp.dc._msdcs.X`.
- A DC that is running the Kerberos Key Distribution Center service over TCP and also hosts default NC X and in site Y, it specifies `_kerberos.tcp.Y._sites.dc._msdcs.X`.
- A server that is running Kerberos Password Change service over TCP for default NC X, it specifies `_kpasswd._tcp.X`.
- A server that is running Kerberos Password Change service over UDP for default NC X, it specifies `_kpasswd._udp.X`.

The DNS query returns a list of SRV records that match this query. The target field of the SRV record contains the fully qualified DNS name of the server.

Upon receiving the DNS query results, a client either contacts servers (attempts the intended protocol request) in weighted random order [RFC 2052], or first sends an **LDAP** "Ping" to servers in weighted random order and then attempts the intended protocol request to a server that responded to the "Ping".

To locate a domain controller using NetBIOS based discovery, the client either queries a WINS server or performs broadcasting. To find a domain controller in **domain** fabricom, the client either sends a NetBIOS name query for `<fabricom>[1C]` to the WINS server or broadcasts for `<fabricom>[1C]` record. And if the client wants to find a primary domain controller, it issues a name query for `<fabricom>[1B]` to the WINS server or broadcasts for `<fabricom>[1B]` record.

Upon receiving the list of matching records from WINS or broadcasting, the client either contacts servers (attempt the intended protocol request) or sends a mailslot "Ping" (see [7.3.5](#)) to servers first and then attempts the intended protocol request to a server that responded to the "Ping".

### 7.3.7 Name Compression and Decompression

Server can choose any compression algorithm as long as the compressed stream can be decompressed using the following name decompression algorithm. When server compresses the names for the **LDAP** "Ping" response, if compression fails, the response of the server is documented

in "Response to Invalid filter" (section [7.3.3.3](#)). When server compresses the names for the mailslot "Ping" response, if compression fails, the server does not send any response back to the client.

### Name Decompression Algorithm

```
--
-- On Entry: InputBuffer - a buffer of compressed data, treated as
--             bytes
--             InputBufferSize - The number of bytes in the InputBuffer
--             StringCount - number of strings needed to be
--                           decompressed from InputBuffer
--
-- On Exit:  OutputBuffers - an array of decompressed strings
--           Success - Set to TRUE if decompression succeeds, set to
--                   FALSE if decompression fails.

SET current = 0
SET deCompressedCount = 0
FOR i = 1 to StringCount
    SET dnsNameLen = 0
    SET firstLabel = 0
    allocate a buffer s[InputBufferSize]
    WHILE current < InputBufferSize
        SET labelSize = InputBuffer[current]
        IF labelSize == '\0' THEN
            s[dnsNameLen] = '\0'
            OutputBuffers[deCompressedCount] = s
            deCompressedCount++
            BREAK
        ELSE IF (labelSize & 0xC0) != 0 THEN
            current++
            labelSize = InputBuffer[current]
            IF labelSize > InputBufferSize THEN
                Success = FALSE
                RETURN
            END IF
            current = labelSize
            CONTINUE
        ELSE
            IF (labelSize + current) >= InputBufferSize THEN
                Success = FALSE
                RETURN
            END IF
            IF firstLabel == 0 THEN
                firstLabel = 1
            ELSE
                s[dnsNameLen] = '.'
                dnsNameLen++
            ENDIF

            Append
                substring InputBuffer[current + 1, current + labelSize]
                to s
            dnsNameLen += labelSize
            current = current + 1 + labelSize
        END IF
    END WHILE
    If i <> deCompressedCount THEN
```

```

        Success = FALSE
        RETURN
    ENDIF
END FOR
Success = TRUE
RETURN

```

### 7.3.8 AD/LDS DC Publication

If an AD/LDS DC is running on a computer joined to an AD/DS **domain**, the AD/LDS DC will (if certain conditions are met, set below) create a [serviceConnectionPoint](#) object in the AD/DS forest of the **domain** to which it is joined. Clients can use this [serviceConnectionPoint](#) object to locate this AD/LDS DC.

Let O be the [msDS-ServiceConnectionPointPublicationService](#) object in the AD/LDS forest whose DN is "CN=SCP Publication Service" relative to the [nTDSService](#) object in the config NC (the DN of the [nTDSService](#) object is "CN=Directory Service, CN=Windows NT, CN=Services" relative to the root of the config NC).

An AD/LDS DC will create (or update, if the object already exists) a [serviceConnectionPoint](#) object unless one of the following conditions is true:

- O (the [msDS-ServiceConnectionPointPublicationService](#) object defined above) exists and O![Enabled](#) = false
- O exists and O![msDS-DisableForInstances](#) contains the DN of the [nTDSDSA](#) object of the replica.

If the **LDAP** add or modify operation to create or update the [serviceConnectionPoint](#) object fails for any reason, including lack of permission to create or update the [serviceConnectionPoint](#) object, the AD/LDS DC retries periodically until the operation succeeds.

The created (or updated) [serviceConnectionPoint](#) object S satisfies the following:

- If O exists and O![msDS-SCPContainer](#) is non-null, then the DN of S is "CN={**dsaGuid**}" relative to O![msDS-SCPContainer](#), where **dsaGuid** is the DC's **DSA GUID**. Otherwise, the DN of S is "CN={**dsaGuid**}" relative to the [computer](#) object of the machine running AD/LDS.
- S![serviceDNSNameType](#) = "A"
- S![serviceClassName](#) = "LDAP"
- S![serviceDNSName](#) is the DNS name of the computer on which the AD/LDS DC is running.
- S![serviceBindingInformation](#) contains two values, "ldap://**dnsName:ldapPort**" and "ldaps://**dnsName:ldapsPort**", where **dnsName** is the DNS name of the computer on which the AD/LDS DC is running, **ldapPort** is the port on which the AD/LDS DC is listening for **LDAP** requests, and **ldapsPort** is the port on which the AD/LDS DC is listening for **SSL**/TLS-protected LDAPS requests.
- S![keywords](#) contains the following values:
  - The DSA **GUID**
  - For each value of the [supportedCapabilities](#) attribute of the rootDSE, a string containing that value

- The string "site:**siteName**" where **siteName** is the name of the site in which the AD/LDS DC is located
- The string "instance:**instanceName**" where **instanceName** is a name configured for this AD/LDS DC, unique among all AD/LDS DCs on the machine running the DC.
- If this AD/LDS DC has the Schema Master FSMO role, the string "fsmo:schema"
- If the AD/LDS DC has the Partition Naming FSMO role, the string "fsmo:naming"
- For each NC-replica on the AD/LDS DC, excluding the NC-replica of the **schema NC**:
  - The string "partition:**ncName**" where **ncName** is the DN of the NC
  - The NC **GUID** (that is, the value of the [objectGUID](#) attribute for the root of the NC)
- If O exists, the values (if any) present on O![keywords](#)

For example, suppose an AD/LDS replica is running on a computer whose DNS name is "adlds-01.fabrikam.com", has a DSA **GUID** of {d07c66ed-b55e-4472-b09c-1ae35980}, possesses both FSMO roles, and has a single application NC whose name is "CN=FirstAppNC" and whose **GUID** is {32079ab-9e49-4c4e-ad36-0f2b8a63f12b}. Further assume it is listening on ports 50000 and 50001 for **LDAP** and **LDAPS** traffic, respectively, is located in a site named "Default-First-Site-Name", has an instance name of "TestInstance" and there are no keywords on O![keywords](#). The resulting [serviceConnectionPoint](#) object could be as follows (depending on the DN and **GUID** of the Config NC):

```
S!serviceDnsNameType = "A"
S!serviceClassName = "LDAP"
S!serviceDNSName = "adlds-01.fabrikam.com"
S!serviceBindingInformation = {
    "ldap://adlds-01.fabrikam.com:50000",
    "ldaps://adlds-01.fabrikam.com:50001"
}
S!keywords = {
    "d07c66ed-b55e-4472-b09c-1ae35980",
    "1.2.840.113556.1.4.1851",
    "1.2.840.113556.1.4.1791",
    "site:Default-First-Site-Name",
    "instance:TestInstance",
    "fsmo:schema",
    "fsmo:naming",
    "partition:CN=FirstAppNC",
    "32079ab-9e49-4c4e-ad36-0f2b8a63f12b",
    "partition:CN=Configuration,CN={FD783EE9-0216-4B83-8A2A-60E45AECCB81}",
    "23b65d43-a701-44b9-9e04-a6555df722eb"
}
```

## 7.4 Domain Join

A machine is said to be *joined to a domain* if certain state exists on the machine and in the **domain** NC. The necessary state is specified in the remainder of this section. The state enables the machine and the **domain** to mutually authenticate using various protocols (for example, [\[MS-NRPC\]](#)).

### 7.4.1 State of a Machine Joined to a Domain

The following variables are part of the state of any machine joined to a domain:

- *domain-secret*: A binary sequence of bytes, containing the secret shared between the machine and the domain.
- *machine-account-name*: The [sAMAccountName](#) of the machine's [computer](#) object within the domain.
- *domain-name*: A tuple containing:
  - netbios: The NetBIOS name of the domain
  - dns: The fully qualified DNS name of the domain

If the **domain** has a DNS name, domain-name.dns contains it. If the **domain** has a NetBIOS name, domain-name.netbios contains it. The value of at least one of these variables is not NULL.

- *domain-locator*: Implementation-specific state sufficient to locate a **domain** controller of the **domain**. If the implementation is capable of locating a domain controller given domain-name, then domain-locator can be NULL.

The specific choices made in implementing a machine joined to a **domain** (e.g. for representing these variables and for generating names) are outside the state model. For Windows, machine-account-name equals the machine name (result of GetComputerName) with "\$" appended, and domain-locator is NULL.

### 7.4.2 State in an Active Directory Domain

A machine *m* that is a member of an Active Directory **domain** *d* has a corresponding object *o* in *d*'s **domain** NC. The object *o* is called the *machine account* of the joined machine *m*. The [objectClass](#) attribute of *o* contains the class [computer](#). In addition to [objectClass](#), the following attributes of *o* are significant to the membership of *m* in *d*:

- [userAccountControl](#)
- [sAMAccountName](#)
- [unicodePwd](#)
- [dNSHostName](#)
- [servicePrincipalName](#)

The syntax and other details of these attributes are documented in [\[MS-ADA1\]](#) and [\[MS-ADA3\]](#).

The following predicates are satisfied by the joined machine *m*'s state and the state of object *o*:

- the **domain** *d*'s NetBIOS name equals *m.domain-name.netbios*
- the **domain** *d*'s fully qualified DNS name equals *m.domain-name.dns*
- *o!*[userAccountControl](#) & ADS\_UF\_WORKSTATION\_TRUST\_ACCOUNT ≠ 0
- *o!*[sAMAccountName](#) equals *m.machine-account-name*
- *o!*[unicodePwd](#) equals *m.domain-secret*

Section [7.1.1.2.1.1.4](#) specifies the representation of a domain's NetBIOS name. A domain's fully qualified DNS name is derived from the DN of its root object, as specified in section [3.1.1.1.5](#).

The specific choices made in implementing a machine joined to a **domain** (e.g. for maintaining these variables) are outside the state model. Windows may periodically update m.domain-secret on the client machine and o.domain-secret in the Windows Active Directory. This behavior is not required for a functional **domain** join.

### 7.4.3 Relationship to Protocols

A joined machine's domain-secret can be used by the Netlogon, NTLM and Kerberos authentication protocols as a parameter for machine **Authentication** to the **domain**. Further Netlogon, NTLM and Kerberos authentication protocol documentation can be found in [\[MS-NRPC\]](#), [\[MS-NLMP\]](#), and [\[MS-KILE\]](#) respectively.

## 7.5 Unicode String Comparison

This section provides a precise description of **Unicode** string comparisons.

### 7.5.1 String Comparison by Using Sort Keys

To compare strings you need to get a "sort key" for each string. A binary comparison of the sort keys can then be used to arrange the strings in any desired order.

For example, for two strings StringA and StringB:

```
set SortKeyA to call GetWindowsActiveDirectorySortKey(StringA)
set SortKeyB to call GetWindowsActiveDirectorySortKey(StringB)
set Result to call CompareSortKeys(SortKeyA, SortKeyB)
if Result is "SortKeyA is equal to SortKeyB"
    StringA is considered equal to StringB
else if Result is "SortKeyA is less than SortKeyB"
    StringA is sorted prior to StringB
else
    assert Result must be "SortKeyA is greater than SortKeyB"
    StringA is sorted after StringB
endif

-- procedure CompareSortKeys
-- On Entry: SortKeyA - An array of bytes returned from
--                  GetWindowsActiveDirectorySortKey
--            SortKeyB - An array of bytes returned from
--                  GetWindowsActiveDirectorySortKey
--
-- On Exit:  Result  - A value indicating if SortKeyA
--                  is less than, equal to, or greater
--                  than SortKeyB

set index to 0
while index is less than Length(SortKeyA) and
    index is also less than Length(SortKeyB)
    if SortKeyA[index] is less than SortKeyB[index]
        set Result to "SortKeyA is less than SortKeyB"
        return
    endif
    if SortKeyA[index] is greater than SortKeyB[index]
        set Result to "SortKeyA is greater than SortKeyB"
```

```

        return
    endif
    increment index
end while
if Length(SortKeyA) is equal to Length(SortKeyB)
    set Result to "SortKeyA is equal to SortKeyB"
else if Length(SortKeyA) is less than Length(SortKeyB)
    set Result to "SortKeyA is less than SortKeyB"
else
    assert Length(SortKeyA) must be greater than Length(SortKeyB)
    set Result to "SortKeyA is greater than SortKeyB"
endif
return

```

Any sorting mechanism may be used to arrange these strings by comparing their sort keys.

## 7.5.2 Unisort.txt Data

Windows Server 2003 and Windows XP get their sorting data from a source file named Unisort.txt (for more information, see [MS-ASRT]). Unisort.txt is a UTF-8 file. All machine readable data is in ASCII, although some comments contain UTF-8 data. Code points are labeled using UTF-16 values.

The file is arranged in a group of sections of records of tab delimited fields. Optional comments begin with a semicolon. Each section contains a label and perhaps a subsection label.

SORTKEY						← Section label
DEFAULT		52086				← Subsection label with record count
0x0001	6	3	2	2	;Start Of Heading	← Record for character U+0001
0x0002	6	4	2	2	;Start Of Text	
0x0003	6	5	2	2	;End Of Text	
...						
0x0041	14	2	2	18	;Latin Capital Letter A	← "highlighted row"
0x0042	14	9	2	18	;Latin Capital Letter B	
0x0043	14	10	2	18	;Latin Capital Letter C	
↑	↑	↑	↑	↑	↑	
Field 1	Field 2	Field 3	Field 4	Field 5	Comment	

**Figure 3: Unisort.txt file arrangement**

Note that labels are any field that does not begin with a numeric (0xNNNN) value. Blank lines and characters following a semicolon are ignored.

This document will use the following notation to describe the processing of the file:

"open" will be used to indicate that queries for records in a specific section will be made. To open the above section with the SORTKEY label and DEFAULT sublabel the following syntax will be used. The open section will be accessible using the "DefaultTable" name.

```

open section DefaultTable
    where name is SORTKEY\DEFAULT from unisort.txt

```

"select" will assign a line from the data file to be referenced by the assigned variable name. To select the highlighted row above this document will use this notation. The selected row will be accessible using the name "CharacterRow".

```
set UnicodeChar to 0x0041
select record CharacterRow from DefaultTable
where field 1 matches UnicodeChar
```

Values from selected records will be referenced by field number. The following would select the individual data fields from the selected row:

```
set CharacterWeight.ScriptMember to CharacterRow.Field2
set CharacterWeight.PrimaryWeight to CharacterRow.Field3
set CharacterWeight.DiacriticWeight to CharacterRow.Field4
set CharacterWeight.CaseWeight to CharacterRow.Field5
```

Some sections of the data file are referenced by a locale LCID (locale identifier).

SORTTABLES						
...						
COMPRESSION				19		← 19 Locales have compressions
LCID		0x0000041a		;Croatian		
TWO		9		← 9 Records in this subsection		
0x0064	0x017e	14	29	4	2	;d z Hacek
0x0044	0x017e	14	29	4	18	;D z Hacek
0x0044	0x017d	14	29	4	26	;D Z Hacek
...						
LCID		0x00000405		;Czech		
TWO		3		← Czech as 3 TWO character compressions		
0x0063	0x0068	14	46	2	2	;ch
0x0043	0x0068	14	46	2	18	;Ch
0x0043	0x0048	14	46	2	26	;CH
↑	↑	↑	↑	↑	↑	↑
Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Comment
...						

Figure 4: Record selection

To select the highlighted record, notation such as this will be used:

```
set Character1 to 0x0043
set Character2 to 0x0068
set SortLocale to 0x0405

open section CompressionTable
where name is SORTTABLES\COMPRESSION\LCID[SortLocale]\TWO from
unisort.txt
select record CompressionRow from CompressionTable
where field 1 matches Character1 and field 2 matches Character2
```



```

set CharacterWeight.ScriptMember to CompressionRow.Field3
set CharacterWeight.PrimaryWeight to CompressionRow.Field4
set CharacterWeight.DiacriticWeight to CompressionRow.Field5
set CharacterWeight.CaseWeight to CompressionRow.Field6

```

### 7.5.3 GetWindowsActiveDirectorySortKey Pseudo Code

```

Structure CharacterWeightType
(
    ScriptMember:      8 bit integer
    PrimaryWeight:     8 bit integer
    DiacriticWeight:   8 bit integer      // AD ignores diacritic weight
    CaseWeight:        8 bit integer      // AD ignores case weight
)

Structure UnicodeWeightType
(
    ScriptMember:      8 bit integer
    PrimaryWeight:     8 bit integer
)

Structure SpecialWeightType
(
    Position :         16 bit integer
    ScriptMember :      8 bit integer
    PrimaryWeight :     8 bit integer
)

Structure ExtraWeightType
(
    W6: 8 bit integer
    W7: 8 bit integer
)

set constant LCID_KOREAN to 0x0412
set constant LCID_KOREAN_UNICODE_SORT to 0x010412
set constant LCID_HUNGARIAN to 0x040e

set constant SORTKEY_SEPARATOR to 0x01
set constant SORTKEY_TERMINATOR to 0x00

set global KoreanScriptMap to InitKoreanScriptMap

//
// Script Member Values.
//
set constant UNSORTABLE to 0
set constant NONSPACE_MARK to 1
set constant EXPANSION to 2
set constant FAREAST_SPECIAL to 3
set constant JAMO_SPECIAL to 4
set constant EXTENSION_A to 5

set constant PUNCTUATION to 6

set constant SYMBOL_1 to 7
set constant SYMBOL_2 to 8

```

```

set constant SYMBOL_3      to 9
set constant SYMBOL_4      to 10
set constant SYMBOL_5      to 11

set constant LATIN         to 14
set constant KANA          to 34
set constant IDEOGRAPH     to 128

set constant MAX_SPECIAL_CASE to SYMBOL_5
set constant FIRST_SCRIPT  to LATIN

set constant MAX_SCRIPTS   to 256

//
// Values for CJK Unified Ideographs Extension A range.
// 0x3400 thru 0x4dbf
//
set constant SCRIPT_MEMBER_EXT_A to 254 // SM for Extension A
set constant PRIMARY_WEIGHT_EXT_A to 255 // AW for Extension A

//
// Bitmask values.
//
// Case Weight (CW) - 8 bits:
// bit 0 => width
// bit 1,2 => small kana, sei-on
// bit 3,4 => upper/lower case
// bit 5 => kana
// bit 6,7 => compression
//
set constant COMPRESS_3_MASK to 0xc0 // compress 3-to-1 or 2-to-1
set constant COMPRESS_2_MASK to 0x80 // compress 2-to-1

set constant CASE_UPPER_MASK to 0xe7 // zero out case bits
set constant CASE_KANA_MASK to 0xdf // zero out kana bit
set constant CASE_WIDTH_MASK to 0xfe // zero out width bit

//
// Masks to isolate the various bits in the case weight.
//
// NOTE: Bit 2 must always equal 1 to avoid getting a byte value
// of either 0 or 1.
//
// NOTE: that since Active Directory Ignores Kana and Width, those
// will end up being CASE_EXTRA_WEIGHT_MASK
set constant CASE_EXTRA_WEIGHT_MASK to 0xc4
set constant ISOLATE_KANA to
(~CASE_KANA_MASK) | CASE_EXTRA_WEIGHT_MASK
set constant ISOLATE_WIDTH to
(~CASE_WIDTH_MASK) | CASE_EXTRA_WEIGHT_MASK

// Active Directory always uses the same case masks (0xc6)
set constant AD_CASE_MASK to
(CASE_UPPER_MASK & CASE_KANA_MASK & CASE_WIDTH_MASK);

//
// Values for fareast special case primary weights.
//

```

```

set constant PW_REPEAT      to 0
set constant PW_CHO_ON      to 1
set constant MAX_SPECIAL_PW to PW_CHO_ON

//
// PW Mask for Cho-On:
// Leaves bit 7 on in PW, so it becomes Repeat if it follows Kana N.
//
set constant CHO_ON_PW_MASK to 0x87

//
// Special weight values
//
set constant MAP_INVALID_WEIGHT to 0xff

//
// Some Significant Values for Korean Jamo.
// The L, V & T syllables in the 0x1100 Unicode range can be composed
// to characters in the 0xac00 range. See The Unicode Standard for
// details
//
set constant NLS_CHAR_FIRST_JAMO to
    0x1100 // Beginning of the jamo range
set constant NLS_CHAR_LAST_JAMO to
    0x11f9 // End of the jamo range
set constant NLS_CHAR_FIRST_VOWEL_JAMO to
    0x1160 // First Vowel Jamo
set constant NLS_CHAR_FIRST_TRAILING_JAMO to
    0x11a8 // First Trailing Jamo

set constant NLS_JAMO_VOWEL_COUNT to
    21 // Number of vowel jamo (V)
set constant NLS_JAMO_TRAILING_COUNT to
    28 // Number of trailing jamo (L)
set constant NLS_HANGUL_FIRST_COMPOSED to
    0xac00 // Beginning of composed range

//
// Values for Unicode Weight extra weights (e.g. Jamo (old Hangul)).
//
set constant ScriptMember_Extra_UnicodeWeight to
    255 // SM for extra UW weights

// Leading Weight/Vowel Weight/Trailing Weight
// according to the current Jamo class.
//
Structure JamoSortInfoType
(
    OldHangulFlag : Boolean // true for an old Hangul sequence
    FillerUsed : Boolean // true if U+1160 (Hangul Jungseong
                        // Filler) used
    LeadingIndex : 8 bit integer // index to the prior modern Hangul
                        // syllable (L)
    VowelIndex : 8 bit integer // index to the prior modern Hangul
                        // syllable (V)
    TrailingIndex : 8 bit integer // index to the prior modern Hangul
                        // syllable (T)
    LeadingWeight : 8 bit integer // Weight to offset from other old

```

```

                                // hangul (L)
VowelWeight : 8 bit integer    // Weight to offset from other old
                                // hangul (V)
TrailingWeight : 8 bit integer// Weight to offset from other old
                                // hangul (T)
)

// This is the raw data record type from the data table
Structure JamoStateDataType
(
  OldHangulFlag : Boolean      // true for an old Hangul sequence
  LeadingIndex : 8 bit integer  // index to the prior modern Hangul
                                // syllable (L)
  VowelIndex : 8 bit integer    // index to the prior modern Hangul
                                // syllable (V)
  TrailingIndex : 8 bit integer // index to the prior modern Hangul
                                // syllable (T)
  ExtraWeight : 8 bit integer   // weight to distinguish from old
                                // Hangul
  TransitionCount : 8 bit integer // number of additional records in
                                // this state
  JamoRecord : data record     // Current record in unisort.txt Jamo
                                // table:
                                // SORTTABLES\JAMOSORT\[Character] section
)

```

#### 7.5.4 GetWindowsActiveDirectorySortKey

```

-- GetWindowsActiveDirectorySortKey
--
-- On Entry:  SourceString  - Unicode String to compute a sort key
--              for
--              SortLocale   - Locale to determine correct linguistic
--              sort
--
-- On Exit:   SortKey       - Byte array containing the computed sort
--              key.
--
Procedure GetWindowsActiveDirectorySortKey( IN  SourceString: Unicode
                                           String,
                                           IN  SortLocale:   LCID,
                                           OUT SortKey:      BYTE
                                           String )

// Compute flags for sort conditions conditions
if SortLocale is LCID_KOREAN or SortLocale is LCID_KOREAN_UNICODE_SORT
  set IsKoreanLocale to true
  if KoreanScriptMap is null
    call InitKoreanScriptMap
  end if
else
  set IsKoreanLocale to false
endif

//
// Allocate buffer to hold different levels of sort key weights.
// UnicodeWeights/ExtraWeights/SpecialWeights will be eventually to

```

```

// be collected together, in that order, into the returned Sortkey
// byte string.
//
// Maximum expansion size is 3 times the input size
//

// Unicode Weight => 4 word (16 bit) length (extension A and Jamo
// need extra words)
set UnicodeWeights to new empty string of UnicodeWeightType

// Extra Weight => 4 byte length (4 weights, 1 byte each) FE Special
set ExtraWeights to new empty string of ExtraWeightType

// Special Weight => dword length (2 words each of 16 bits)
set SpecialWeights to new empty string of SpecialWeightType

//
// Go through the string, code point by code point,
// testing for compressions and double compressions
//

// loop presumes 0 based index for source string
for SourceIndex is 0 to Length(SourceString) -1

    //
    // Get weights
    // CharacterWeights will contain all of the weight information
    // for the character tested.
    set CharacterWeights to
        call GetCharacterWeights(SortLocale, SourceString[SourceIndex])

    set ScriptMember to CharacterWeight.ScriptMember

    // Special case weights have script members less than
    // MAX_SPECIAL_CASE (11)
    if ScriptMember is greater than MAX_SPECIAL_CASE

        //
        // No special case on character, but must check for
        // compression characters and double compression
        // characters.
        //

        set HasDoubleCompression to
            call TestDoubleCompression(SortLocale,
                                      SourceString,
                                      SourceIndex)

        set Result to
            call GetCompression(CharacterWeight)
        case Result of
            "3 to 1 Compression" :
                if HasDoubleCompression is true

                    // Test for double 3 to 1 compression (eg ddzs)
                    // Need SourceIndex + 1 to skip first d
                    set TempWeights to
                        call Is3To1Compression(SortLocale,

```

```

SourceString,
SourceIndex + 1)

if TempWeights is not null
    set CharacterWeight to TempWeights
    set UnicodeWeight to
        call CorrectUnicodeWeight(CharacterWeight,
                                   IsKoreanLocale)
    // It was a double compression, so
    // add weight twice
    append UnicodeWeight to UnicodeWeights
    append UnicodeWeight to UnicodeWeights

    // Add 3 to source index for 3 compression characters
    // An additional 1 for the initial character will be
    // added in the outer loop
    set SourceIndex to SourceIndex + 3
next SourceIndex
end if

end if

// Not a double compression, test for
// normal 3 to 1 compression
set TempWeights to
    call Is3To1Compression(SortLocale,
                           SourceString,
                           SourceIndex)

if TempWeights is not null
    set CharacterWeight to TempWeights
    set UnicodeWeight to
        call CorrectUnicodeWeight(CharacterWeight,
                                   IsKoreanLocale)

    append UnicodeWeight to UnicodeWeights
    // Add 2 to source index for 2 of 3 compression
    // characters
    // An additional 1 for the initial character will be
    // added in the outer loop
    set SourceIndex to SourceIndex + 2
next SourceIndex
end if

// If 3 to 1 compressions aren't found, try 2 to 1 fall
// through to case "2 to 1 Compression"

"2 to 1 Compression" :
if HasDoubleCompression is true
    // Test for double 2 to 1 compression (eg ddz)
    // Need SourceIndex + 1 to skip first d
    set TempWeights to
        call Is2To1Compression(SortLocale,
                                SourceString,
                                SourceIndex + 1)

    if TempWeights is not null
        set CharacterWeight to TempWeights
        set UnicodeWeight to

```

```

        call CorrectUnicodeWeight(CharacterWeight,
                                   IsKoreanLocale)

        // It was a double compression, so add weight twice
        append UnicodeWeight to UnicodeWeights
        append UnicodeWeight to UnicodeWeights

        // Add 2 to source index for 2 compression characters
        // An additional 1 for the initial character will be
        // added in the outer loop
        set SourceIndex to SourceIndex + 2
        next SourceIndex
    end if
end if

// Not a double compression, test for normal 2 to 1
// compression
set TempWeights to
    call Is2To1Compression(SortLocale,
                           SourceString,
                           SourceIndex)

if TempWeights is not null
    set CharacterWeight to TempWeights
    set UnicodeWeight to
        call CorrectUnicodeWeight(CharacterWeight,
                                   IsKoreanLocale)

    append UnicodeWeight to UnicodeWeights
    // Add 1 to source index for 1 of 2 compression
    // characters
    // An additional 1 for the initial character will be
    // added in the outer loop
    set SourceIndex to SourceIndex + 1
    next SourceIndex
end if

// If 2 to 1 compressions aren't found, try 2 to 1 fall through
// to case OTHERS
OTHERS :
    set UnicodeWeight to
        CorrectUnicodeWeight(CharacterWeight,
                              IsKoreanLocale)
    Append UnicodeWeight to UnicodeWeights

end case

else

    call SpecialCaseHandler(SourceString,
                           SourceIndex,
                           UnicodeWeights,
                           ExtraWeights,
                           SpecialWeights,
                           SortLocale,
                           IsKoreanLocale)

end if

```

```

end for

// Store the Unicode Weights in the destination buffer.
for each UnicodeWeight in UnicodeWeights

    // Copy Unicode weight to destination buffer.
    append UnicodeWeight.ScriptMember to SortKey
    append UnicodeWeight.PrimaryWeight to SortKey
end for

// Copy Separator to destination buffer.
append SORTKEY_SEPARATOR to SortKey

//
// Diacritic Weights are not used for Active Directory.
//

//
// Copy Separator to destination buffer.
//
append SORTKEY_SEPARATOR to SortKey

//
// Case Weights are not used for Active Directory
//

//
// Copy Separator to destination buffer.
//
append SORTKEY_SEPARATOR to SortKey

//
// Store the Extra Weights in the destination buffer for
// Asia Special.
//
// Eliminate unnecessary XW.
// Copy extra weights to destination buffer.
//
if Length(ExtraWeights) is greater than 0
    // NORM_IGNORE_NONSPACE is set for Active Directory,
    // so just add separators for W4 & W5.
    append 0xff to SortKey
    append 0x02 to SortKey

    // Append W6 group to SortKey
    // Trim unused values from the end of the string
    set EndExtraWeight to Length(ExtraWeights) - 1
    while EndExtraWeight greater than 0 and
        ExtraWeightSeparator[EndExtraWeight].W6 == 0xe4
        decrement EndExtraWeight
    end while

    set ExtraWeightIndex to 0
    while ExtraWeightIndex is less than or equal to EndExtraWeight
        append ExtraWeightSeparator[ExtraWeightIndex].W6 to SortKey
        increment ExtraWeightIndex
    end while

```



```

// Append W6 separator
append 0xff to SortKey

// Append W7 group to SortKey
// Trim unused values from the end of the string
set EndExtraWeight to Length(ExtraWeights) - 1

while EndExtraWeight greater than 0 and
    ExtraWeightSeparator[EndExtraWeight].W7 == 0xe4
    decrement EndExtraWeight
end while

set ExtraWeightIndex to 0
while ExtraWeightIndex is less than or equal to EndExtraWeight
    append ExtraWeightSeparator[ExtraWeightIndex].W7 to SortKey
    increment ExtraWeightIndex
end while

// Append W7 separator
append 0xff to SortKey
end if

//
// Copy Separator to destination buffer.
//
append SORTKEY_SEPARATOR to SortKey

//
// Store the Special Weights in the destination buffer.
//
// - Copy special weights to destination buffer.
//
for each SpecialWeight in SpecialWeights
    set Byte1 to SpecialWeight.Position >> 8           // High byte (most
                                                         // significant)
    set Byte2 to SpecialWeight.Position & 0xff         // Low byte (least
                                                         // significant)
    append Byte1 to SortKey
    append Byte2 to SortKey
    append SpecialWeight.Script to SortKey
    append SpecialWeight.Weight to SortKey
end for

//
// Copy terminator to destination buffer.
//
append SORTKEY_TERMINATOR to SortKey

return SortKey

```

## 7.5.5 TestDoubleCompression

```

-- TestDoubleCompression
--
-- On Entry:  SortLocale    - Locale to use for linguistic data
--            SourceString  - Unicode String to look for double
--                               compression in

```

```

--          SourceIndex    - Index of character in string to look
--                          for start of double compression
--
-- On Exit:   Result        - Set to true if a double compression was
--                          found
--

Procedure TestDoubleCompression(IN  SortLocale : LCID,
                               IN  SourceString : Unicode String,
                               IN  SourceIndex : 32 bit integer,
                               OUT Result : Boolean )

// Double compressions only happen to Hungarian
// Note that this can be found in unisort.txt in the
// SORTTABLES\DOUBLECOMPRESSION section, however since
// there's only 1 locale we just hard code it here.
if SortLocale not equal to LCID_HUNGARIAN)
    set Result to false
    return
end if

// first test to make sure more data is available
if SourceIndex + 1 is greater than or equal to Length(SourceString)
    set Result to false
    return
end if

// CMP_MASKOFF_CW (e7) is not necessary since it was already masked
// off
set FirstWeight to
    call GetCharacterWeights(SortLocale, SourceString[SourceIndex])
set SecondWeight to
    call GetCharacterWeights(SortLocale, SourceString[SourceIndex + 1])

if FirstWeight is equal to SecondWeight
    set Result to true
else
    set Result to false
end if
return

```

### 7.5.6 GetCompression

```

-- GetCompression
--
-- On Entry:  CharacterWeight - Weights structure to test for a
--                          compression
--
-- On Exit:   Result          - Type of compression found:
--                          "No Compression"
--                          "3 to 1 compression"
--                          "2 to 1 compression"
--
Procedure GetCompression(IN CharacterWeight : CharacterWeightType,
                        OUT Result )

case CharacterWeight.CaseWeight & COMPRESS_3_MASK

```

```

        COMPRESS_3_MASK : set Result = "3 to 1 compression"
        COMPRESS_2_MASK : set Result = "2 to 1 compression"
        OTHERS : set Result = "No compression"
    end case

    return

```

### 7.5.7 CorrectUnicodeWeight

```

-- CorrectUnicodeWeight
--
-- On Entry:  CharacterWeight - Weights structure to get Unicode
--                               weight of
--             IsKoreanLocale - True if this locale must adjust for
--                               Korean mapped scripts behavior.
--
-- On Exit:   UnicodeWeight  - Corrected Unicode Weight
--
Procedure CorrectUnicodeWeight(
    IN CharacterWeight : CharacterWeightType,
    IN IsKoreanLocale : Boolean,
    OUT UnicodeWeight : UnicodeWeightType )

Set UnicodeWeight to call MakeUnicodeWeight(
    CharacterWeight.ScriptMember,
    CharacterWeight.PrimaryWeight,
    IsKoreanLocale)

return UnicodeWeight

```

### 7.5.8 MakeUnicodeWeight

```

-- MakeUnicodeWeight
--
-- On Entry:  ScriptMember      - Script member to use for Unicode
--                               weight
--             PrimaryWeight     - Primary weight to use for Unicode
--                               weight
--             IsKoreanLocale    - True if this locale must adjust for
--                               Korean mapped scripts behavior.
--
-- On Exit:   UnicodeWeight     - Corrected Unicode Weight
--
Procedure MakeUnicodeWeight(IN ScriptMember : 8 bit byte,
    IN PrimaryWeight : 8 bit byte,
    IN IsKoreanLocale : Boolean,
    OUT UnicodeWeight : UnicodeWeightType )

if IsKoreanLocale is true
    set UnicodeWeight.ScriptMember to
        KoreanScriptMap[ScriptMember]
else
    set UnicodeWeight.ScriptMember to ScriptMember
end if
set UnicodeWeight.PrimaryWeight to PrimaryWeight

```

```
return UnicodeWeight
```

## 7.5.9 GetCharacterWeights

```
-- GetCharacterWeights
--
-- On Entry:  SortLocale      - Locale to use for linguistic data
--            SourceCharacter - Unicode Character to return weight
--                               for
--
-- On Exit:   Result          - A structure containing the weights
--                               for this character
--
Procedure GetCharacterWeights(IN SortLocale : LCID,
                             IN SourceCharacter : Unicode Character
                             OUT Result : CharacterWeightType )

// Search for the character in the exception table
open section ExceptionTable
where name is SORTTABLES\EXCEPTION\LCID[SortLocale] from unisort.txt
select record CharacterRow from ExceptionTable
where field 1 matches SourceCharacter

if CharacterRow is null
    // Not found, search for the character in the default table
    open section DefaultTable
    where name is SORTKEY\DEFAULT from unisort.txt
    select record CharacterRow from DefaultTable
    where field 1 matches SourceCharacter

    if CharacterRow is null
        // Not found in default table either, check expansions
        set Expansion to GetExpandedCharacters(SourceCharacter)
        if Expansion is not null
            // Has an expansion, set appropriate weights
            set Result.ScriptMember to EXPANSION
        else
            // No expansion, set appropriate weights
            set Result.ScriptMember to UNSORTABLE
        end if
        set Result.PrimaryWeight to 0
        set Result.DiacriticWeight to 0
        set Result.CaseWeight to 0

        return Result
    end if
end if

set Result.ScriptMember to CharacterRow.Field2
set Result.PrimaryWeight to CharacterRow.Field3
set Result.DiacriticWeight to CharacterRow.Field4
set Result.CaseWeight to CharacterRow.Field5

// Some fields need correction
// Active Directory does not use case weight but we need some bits
// to test for double compression
```

```

// For Active Directory we are able to mask the case weight on read
// (0xc6)
set Result.CaseWeight to Result.CaseWeight & AD_CASE_WEIGHT

return Result

```

### 7.5.10 GetExpansionWeights

```

-- GetExpansionWeights
--
-- On Entry:  SourceCharacter - Character to look up expansions for
--            SortLocale     - Locale to get sort weights for
--
-- On Exit:   Weights        - String of 2 or 3 weights for this
--                           character
--
Procedure GetExpansionWeights(
    IN SourceCharacter : Unicode Character,
    IN SortLocale      : LCID,
    OUT Weights        : CharacterWeightType String)

set Weights to new empty string of CharacterWeightType
set ExpandedCharacters to call GetExpandedCharacters(SourceCharacter)

// Append first weight
set Weight to
    call GetCharacterWeights(SortLocale, ExpandedCharacters[0])
append Weight to Weights

// Get second weight, it may expand again
set Weight to
    call GetCharacterWeights(SortLocale, ExpandedCharacters[1])
if Weight.ScriptMember is EXPANSION
    // second weight expands again, get new expansion
    // note that this can only happen once, as it does
    // with the U+fb03 ffi (ffi ligature)
    set ExpandedCharacters to
        call GetExpandedCharacters(ExpandedCharacters[1])

    // Append second expansion's first weight
    set Weight to
        call GetCharacterWeights(SortLocale, ExpandedCharacters[0])
    append Weight to Weights

    // Get second weight for second expansion, it will not expand again
    set Weight to
        call GetCharacterWeights(SortLocale, ExpandedCharacters[1])
end if

// Finish appending second weight to weights string
append Weight to Weights

return Result

```

### 7.5.11 GetExpandedCharacters

```
-- GetExpandedCharacters
--
-- On Entry:  SourceCharacter - Character to look for in expansion
--                                     table
--
-- On Exit:   Result          - Array of 2 unicode characters for the
--                                     expansion or null if no expansion
--                                     found
--
-- NOTE: Look for default table characters first, some entries in the
-- expansion table are only used in exception tables for some locales
-- (ie: 0x00c4 Å)
Procedure GetExpandedCharacters(IN SourceCharacter : Unicode Character
                                OUT Result : Unicode Character[2] )

// Search for the expansion in the expansion table
open section ExpansionTable
where name is SORTTABLES\EXPANSION from unisort.txt
select record ExpansionRow from ExceptionTable
where field 1 matches SourceCharacter

if ExpansionRow is null
    set Result to null
    return Result
end if

set Result[0] to ExpansionRow.Field2
set Result[1] to ExpansionRow.Field3

return Result
```

### 7.5.12 Is3To1Compression

```
-- Is3To1Compression
--
-- On Entry:  SortLocale      - Locale to use for linguistic data
--              SourceString   - Unicode String to search for
--                               compression in
--              SourceIndex    - Position in string to search for
--
-- On Exit:   Result          - A structure containing the weights
--                               for the compression, or null if no
--                               compression is found.
--
Procedure Is3To1Compression (IN SortLocale : LCID,
                              IN SourceString : Unicode String,
                              IN SourceIndex : 32 bit int,
                              OUT Result : CharacterWeightType )

// Make sure there is enough data left
if SourceIndex + 3 is greater than or equal to SourceString.Length
    set Result to null
    return
end if
```

```

// Search for the character in the 3-1 compression table
open section CompressionTable
where name is SORTTABLES\COMPRESSION\LCID[SortLocale]\THREE from
    unisort.txt
// Compression table may not be found if locale does not have them
if CompressionTable is null
    set Result to null
    return
end if

select record CompressionRow from CompressionTable
where field 1 matches SourceString[SourceIndex] and
where field 2 matches SourceString[SourceIndex + 1] and
where field 3 matches SourceString[SourceIndex + 2]

// If this sequence is not a compression we will not find one
if CompressionRow is null
    set Result to null
    return
end if

// Found a compression, get its weights
set Result.ScriptMember to CompressionRow.Field4
set Result.PrimaryWeight to CompressionRow.Field5

// These 2 are ignored by Active Directory
set Result.DiacriticWeight to CompressionRow.Field6
set Result.CaseWeight to CompressionRow.Field7
return Result

```

### 7.5.13 Is2To1Compression

```

-- Is2To1Compression
--
-- On Entry:  SortLocale    - Locale to use for linguistic data
--            SourceString  - Unicode String to search for
--                               compression in
--            SourceIndex   - Position in string to search for
--
-- On Exit:   Result        - A structure containing the weights
--                               for the compression, or null if no
--                               compression is found.
--
Procedure Is2To1Compression (IN SortLocale : LCID,
                             IN SourceString : Unicode String,
                             IN SourceIndex : 32 bit int,
                             OUT Result : CharacterWeightType )

// Make sure there is enough data left
if SourceIndex + 2 is greater than or equal to SourceString.Length
    set Result to null
    return
end if

// Search for the character in the 2-1 compression table
open section CompressionTable
where name is SORTTABLES\COMPRESSION\LCID[SortLocale]\TWO from

```

```

        unisort.txt
// Compression table may not be found if locale does not have them
if CompressionTable is null
    set Result to null
    return
end if

select record CompressionRow from CompressionTable
where field 1 matches SourceString[SourceIndex] and
where field 2 matches SourceString[SourceIndex + 1]

// If this sequence is not a compression we will not find one
if CompressionRow is null
    set Result to null
    return
end if

// Found a compression, get its weights
set Result.ScriptMember to CompressionRow.Field3
set Result.PrimaryWeight to CompressionRow.Field4

// These 2 are ignored by Active Directory
set Result.DiacriticWeight to CompressionRow.Field5
set Result.CaseWeight to CompressionRow.Field6

return Result

```

#### 7.5.14 SpecialCaseHandler

```

-- SpecialCaseHandler
--
-- On Entry:  SourceString  - Source Unicode String
--            SourceIndex   - Current Index within source string
--            UnicodeWeights - String of UnicodeWeightType to append
--                           additional weight(s) to
--            ExtraWeights   - String of ExtraWeightType to append
--                           extra weight(s) to if needed
--            SpecialWeights - String of SpecialWeightType to append
--                           special weight(s) to if needed
--            SortLocale     - Locale to use for linguistic sorting
--                           data
--            IsKoreanLocale - True if this locale needs Korean
--                           special casing of the ScriptMember
--                           value
--
-- On Exit:   SourceIndex   - Index of last character processed,
--                           caller must increment to continue
--                           loop
--                           Korean Jamo cases can increment this
--                           beyond its input value
--            UnicodeWeights - The UnicodeWeight of the processed
--                           character(s)
--                           is appended to this string.
--            ExtraWeights   - The ExtraWeight, if any, of the
--                           processed character(s) is appended to
--                           this string.
--            SpecialWeights - The Special Weight, if any, of the

```



```

--                                processed
--                                character(s) is appended to this
--                                string.
--
Procedure SpecialCaseHandler (IN    SourceString : Unicode String
                              INOUT SourceIndex : 32 bit integer
                              INOUT UnicodeWeights : UnicodeWeightType
                                              String,
                              INOUT ExtraWeights : ExtraWeightType
                                              String,
                              INOUT SpecialWeights : SpecialWeightType
                                              String,
                              IN    SortLocale : LCID,
                              IN    IsKoreanLocale : Boolean )

// Get the weight for the current character
set CharacterWeight to SourceString[SourceIndex]

case CharacterWeight.ScriptMember of

    UNSORTABLE :
        // Character is unsortable, so skip it
        return
    NONSPACE_MARK :
        // Character is a nonspacing mark, but Active Directory
        // ignores diacritics, so skip it
        return
    EXPANSION :
        // Expansion character, each character has 2 weights, store
        // each weight separately
        set Weights to
            call GetExpansionWeights(SourceString[SourceIndex],
                                     SortLocale)

        // Store the appropriate weights, there should be 2 or 3
        for each Weight in Weights
            // Store the weight of the first character of the
            // expansion
            set UnicodeWeight to CorrectUnicodeWeight(Weight,
                                                         IsKoreanLocale)
            append UnicodeWeight to UnicodeWeights
        end for

        return

    PUNCTUATION :
        // Active Directory uses the "string sort" method, so
        // punctuation is treated like symbols. Fall through to the
        // Symbol behavior
        fall through to case SYMBOL_1
    SYMBOL_1 :
    SYMBOL_2 :
    SYMBOL_3 :
    SYMBOL_4 :
    SYMBOL_5 :
        // Character is a symbol, store Unicode Weights
        // Active Directory does not use Diacritic or Case Weights
        set UnicodeWeight to CorrectUnicodeWeight(Weights[0],

```

```

IsKoreanLocale)
append UnicodeWeight to UnicodeWeights
return

FAREAST_SPECIAL :
// Get the primary and case weight of the current code point
set PrimaryWeight to UnicodeWeight.PrimaryWeight
set ExtraWeight to UnicodeWeight.CaseWeight
// Mask off the bits we don't want
// In Active Directory this will result in either 0xc6 or 0xc4
set ExtraWeight to (ExtraWeight & AD_CASE_MASK) |
CASE_EXTRA_WEIGHT_MASK

// Special case Repeat and Cho-On
// PrimaryWeight = 0 => Repeat
// PrimaryWeight = 1 => Cho-On
// PrimaryWeight = 2+ => Kana
if PrimaryWeight is less than or equal to MAX_SPECIAL_PW
// If the script member of the previous character is
// invalid, then give the special character
// invalid weight (highest possible weight) so that it
// will sort AFTER everything else.
set PreviousIndex to SourceIndex - 1
set UnicodeWeight.ScriptMember to MAP_INVALID_WEIGHT
set UnicodeWeight.PrimaryWeight to MAP_INVALID_WEIGHT
while PreviousIndex is greater than or equal to 0
set PreviousWeight to
call GetCharacterWeights(SortLocale,
SourceString[PreviousIndex])
if PreviousWeight.ScriptMember is
less than FAREAST_SPECIAL
if PreviousWeight.ScriptMember is
not equal to EXPANSION
// UNSORTABLE or NONSPACE_MARK
// Ignore these since we're trying to get the
// previous ScriptMember/PrimaryWeight
decrement PreviousIndex
continue while PreviousIndex
end if
else if PreviousWeight.ScriptMember is equal to
FAREAST_SPECIAL
if PreviousWeight.PrimaryWeight is
less than or equal to
MAX_SPECIAL_PW
// Handle case where two special chars follow
// each other. Keep going back in the string
decrement PreviousIndex
continue while PreviousIndex
end if
set UnicodeWeight to call MakeUnicodeWeight(
KANA,
PreviousWeight.PrimaryWeight,
IsKoreanLocale)
// Only build weights W6 & W7 if the previous
// character is KANA.
//
// Active Directory ignores W4 & W5
// Always:

```

```

//      W6 = previous CW & ISOLATE_KANA

set PreviousExtraWeight to PreviousWeight.CaseWeight

// Mask off the bits we don't want
// In Active Directory this will result in 0xc6 or
// 0xc4
set PreviousExtraWeight to CASE_EXTRA_WEIGHT_MASK |
    (PreviousExtraWeight & AD_CASE_MASK)

// We ignore kana and width with the Active Directory
// Flags, so these are merely CASE_EXTRA_WEIGHT_MASK
set ExtraWeight.W6 to CASE_EXTRA_WEIGHT_MASK
set ExtraWeight.W7 to CASE_EXTRA_WEIGHT_MASK

// Repeat is already done, which is:
// UW = previous UW (set above)
// W5 = ignored in Active Directory
// W7 = previous CW & ISOLATE_WIDTH (done above)

if PrimaryWeight is not equal to PW_REPEAT
    // Cho-On:
    // UW = previous UW & CHO_ON_UW_MASK
    // W5 = ignored in Active Directory
    // W7 = current CW & ISOLATE_WIDTH
    // (done above)
    set UnicodeWeight.PrimaryWeight to
        UnicodeWeight.PrimaryWeight & CHO_ON_PW_MASK
end if
// Append the calculated ExtraWeight
append ExtraWeight to ExtraWeights

else
    // The previous weight is not FAREAST_SPECIAL, so just
    // store the previous weight
    set UnicodeWeight to
        CorrectUnicodeWeight(PreviousWeight,
                               IsKoreanLocale)
end if
end while

// Append the weight we found
append UnicodeWeight to UnicodeWeights

// Active Directory does not use Diacritic or Case Weights
// so no further work is necessary here
else
    //Kana
    //      ScriptMember = KANA
    //      PrimaryWeight = current PrimaryWeight
    //      W4 & W5 are unused in Active Directory
    //      W6 = current CaseWeight & ISOLATE_KANA
    //      W7 = current CaseWeight & ISOLATE_WIDTH
    set UnicodeWeight to
        call MakeUnicodeWeight(KANA,
                               CharacterWeight.PrimaryWeight,
                               IsKoreanLocale)
    append UnicodeWeight to UnicodeWeights

```

```

        // Note: ExtraWeight is always be 0xc4 or 0xc6 in Active
        // Directory
        // So W6 & W7 always end up CASE_EXTRA_WEIGHT_MASK
        set TempExtraWeight.W6 to ExtraWeight & ISOLATE_KANA;
        set TempExtraWeight.W7 to ExtraWeight & ISOLATE_WIDTH;
        append TempExtraWeight to ExtraWeights
    end if
    // Active Directory does not use Diacritic or Case Weights
    return

JAMO_SPECIAL :
    // See if it's a leading Jamo
    if (call IsJamoLeading(SourceString[SourceIndex])) is true

        // If the characters beginning at SourceIndex are a
        // valid old Hangul composition, create the SortKey
        // according to the old Hangul rule

        set OldHangulCount to
            call MapOldHangulSortKey(SourceString,
                                    SourceIndex,
                                    SortLocale,
                                    UnicodeWeights,
                                    IsKoreanLocale)

        if OldHangulCount is greater than 0

            // Decrement OldHangulCount because the caller's loop
            // will increment the SourceIndex as well
            decrement OldHangulCount
            set SourceIndex to SourceIndex + OldHangulCount

        return
    end if
end if

// Otherwise, fall back to the normal behavior
//
// No special case on the character, so store the Jamo's
// weights. We store the real script member in the diacritic
// weight in the tables since both the diacritic weight and
// the case weight are not used in Korean

// For example, from unisort.txt:
//
// 0x1101  4   84   83   2   ; ㅓ Choseong Ssangkiyeok
//
// Field 2 has a value of 4 to trigger the code case for
// JAMO_SPECIAL.
// Field 3 (84) is the real primary weight for this Jamo.
// Field 4 (83) is the real script member for this Jamo.
set UnicodeWeight to
    call MakeUnicodeWeight(CharacterWeight.DiacriticWeight,
                          CharacterWeight.PrimaryWeight,
                          IsKoreanLocale)
append UnicodeWeight to UnicodeWeights
return

```

```

EXTENSION_A :
    // Extension A gives us two weights
    // UnicodeWeight = SM_EXT_A, AW_EXT_A, AW, DW
    // Diacritic and Case Weights are ignored by Active Directory

    // First Weight
    UnicodeWeight = MakeUnicodeWeight(SCRIPT_MEMBER_EXT_A,
                                      PRIMARY_WEIGHT_EXT_A,
                                      IsKoreanLocale)

    append UnicodeWeight to UnicodeWeights

    // Since the script member is our flag for this EXTENSION_A
    // special case, the real weights are in fields 2 & 3.
    //
    // Example:
    // From unisort.txt:
    // 0x3400 5 16 2 2 ; □ CJK Unified Ideographs Extension A
    //
    // Field 2 is the script member.
    // Field 3 is the primary weight.

    // Second Weight
    UnicodeWeight = MakeUnicodeWeight(
        CharacterWeight.PrimaryWeight,
        CharacterWeight.DiacriticWeight,
        false)

    append UnicodeWeight to UnicodeWeights

    return
end case

```

### 7.5.15 GetPositionSpecialWeight

```

-- GetPositionSpecialWeight
--
-- On Entry:  Position - Position to calculate weight for
--
-- On Exit:   Weight    - Resulting weight
--
Procedure GetPositionSpecialWeight(IN Position : 32 bit integer
                                  OUT Weight : 16 bit integer )

    // We need to add some bits (0x8003) to adjust the weight and because
    // some bits are expected. Since we're setting 0x3, we rotate source
    // index 2 bits so as to not lose the precision.

    // Note that if SourceIndex is larger than 0x1FFF, then some bits will
    // be lost on the conversion to 16 bits. Presumably if a string is
    // over 8191 characters long, they will differ well before this point,
    // so the lost information is irrelevant.

    set Weight to (SourceIndex << 2) | 0x8003
    return Weight

```

## 7.5.16 MapOldHangulSortKey

```
-- MapOldHangulSortKey
--
-- On Entry:  SourceString  - Unicode String to test
--            SourceIndex   - Index of leading Jamo to start from
--            SortLocale    - Locale to use for linguistic sort data
--            UnicodeWeights - String to store any Unicode weight
--                               found for this character(s)
--
-- On Exit:   CharactersRead - Number of old Hangul found
--            UnicodeWeights - Any Unicode weights found are appended
--
Procedure MapOldHangulSortKey(IN SourceString : Unicode String,
                             IN SourceIndex  : 32 bit integer,
                             IN SortLocale   : LCID,
                             INOUT UnicodeWeights : String of
                                     UnicodeWeightType,
                             IN IsKoreanLocale : Boolean,
                             OUT CharactersRead : 32 bit integer )

set CurrentIndex to SourceIndex
set JamoSortInfo to empty JamoSortInfoType

// Get any Old Hangul Leading Jamo composition for our Leading Jamo
set JamoClass to call GetJamoComposition(SourceString,
                                         SourceIndex,
                                         "Leading Jamo Class",
                                         JamoSortInfo )

if JamoClass is equal to "Vowel Jamo Class"

    // We have a Vowel Jamo, try to find an Old Hangul Vowel Jamo
    // composition.
    set JamoClass to call GetJamoComposition(SourceString,
                                             SourceIndex,
                                             "Vowel Jamo Class",
                                             JamoSortInfo )

end if

if JamoClass is equal to "Trailing Jamo Class"
    // We have a Trailing Jamo, try to find an Old Hangul Trailing Jamo
    // composition.
    call GetJamoComposition(SourceString,
                           SourceIndex,
                           "Trailing Jamo Class",
                           JamoSortInfo )
end if

// If we have a valid leading and vowel sequence and this is old
// Hangul...
if JamoSortInfo.OldHangulFlag is true


    // Compute the modern hangul syllable prior to this composition
    // Users formula from Unicode 3.0 Section 3.11 p54
    // "Hangul Syllable Composition"
    // This converts a U+11.. sequence to a U+AC00 character

    set ModernHangul to
```

```

        (JamoSortInfo.LeadingIndex * NLS_JAMO_VOWELCOUNT +
         JamoSortInfo.VowelIndex) *
        NLS_JAMO_TRAILING_COUNT +
        JamoSortInfo.TrailingIndex +
        NLS_HANGUL_FIRST_SYLLABLE

    if JamoSortInfo.FillerUsed is true
        // If the filler is used, sort before the modern Hangul,
        // instead of after
        decrement ModernHangul

        // If we fall off the modern Hangul syllable block...
        if ModernHangul is less than NLS_HANGUL_FIRST_SYLLABLE
            // Sort after the previous character (Circled Hangul
            // Kiyeok A )
            set ModernHangul to 0x326e
        end if
        // Shift the leading weight past any old Hangul that sorts
        // after this modern Hangul
        set JamoSortInfo.LeadingWeight to
            JamoSortInfo.LeadingWeight + 0x80
    end if

    // Store the weights
    set CharacterWeight to call GetCharacterWeights(ModernHangul)
    set UnicodeWeight to call CorrectUnicodeWeight(CharacterWeight,
                                                    IsKoreanLocale)

    append UnicodeWeight to UnicodeWeights

    // Add additional weights
    set UnicodeWeight to
        MakeUnicodeWeight(ScriptMember_Extra_UnicodeWeight,
                          JamoSortInfo.LeadingWeight,
                          false)
    append UnicodeWeight to UnicodeWeights
    set UnicodeWeight to
        MakeUnicodeWeight(ScriptMember_Extra_UnicodeWeight,
                          JamoSortInfo.VowelWeight,
                          false)
    append UnicodeWeight to UnicodeWeights
    set UnicodeWeight to
        MakeUnicodeWeight(ScriptMember_Extra_UnicodeWeight,
                          JamoSortInfo.TrailingWeight,
                          false)
    append UnicodeWeight to UnicodeWeights

    // Return the characters consumed
    set CharactersRead to CurrentIndex - SourceIndex
    return CharactersRead

end if

// Otherwise it is not a valid old Hangul composition and we do not do
// anything with it

set CharactersRead to 0
return CharactersRead

```

## 7.5.17 GetJamoComposition

```
-- GetJamoComposition
--
-- On Entry:  SourceString - Unicode String to test
--            CurrentIndex - Index of leading Jamo to start from
--            JamoClass    - Class of Jamo to look for
--            JamoSortInfo - Information about the current sequence
--
-- On Exit:   JamoSortInfo - Updated with information about the new
--            sequence
--            SourceIndex  - Updated to next character if Jamo are
--                        found
--            NewJamoClass - New class to look for next
--
-- NOTE: This function assumes the character at
--       SourceString[SourceIndex] is a leading Jamo.
--       Ie: IsJamo() returned true
--
Procedure GetJamoComposition (IN      SourceString : Unicode String,
                             INOUT   CurrentIndex : 32 bit integer,
                             IN      JamoClass    : enumeration,
                             INOUT   JamoSortInfo : JamoSortInfoType,
                             OUT     NewJamoClass : enumeration )

set CurrentCharacter to SourceString[CurrentIndex]

// Get the Jamo information for the current character
set JamoStateData to call GetJamoStateData(CurrentCharacter)
set JamoSortInfo to call UpdateJamoSortInfo(JamoClass, JamoStateData,
                                           JamoSortInfo)

// Move on to the next character
increment CurrentIndex

while CurrentIndex is less than Length(SourceString)
    set CurrentCharacter to SourceString[CurrentIndex]

    if call IsJamo(CurrentCharacter) is not true
        // The current character is not a Jamo, we are done checking
        // for a Jamo composition
        set NewJamoClass to "Invalid Jamo Sequence"
        return
    end if

    if CurrentCharacter is equal to 0x1160
        set JamoSortInfo.FillerUsed to true
    end if

    // Get the Jamo class of it
    if call IsJamoLeading(CurrentCharacter) is true
        set NewJamoClass to "Leading Jamo Class"
    else if call IsJamoTrailing(CurrentCharacter) is true
        set NewJamoClass to "Trailing Jamo Class"
    else
        set NewJamoClass to "Vowel Jamo Class"
    end if
end while
```



```

if JamoClass is not equal to NewJamoClass
    return NewJamoClass
end if

// Push the current Jamo (SourceString[CurrentIndex]) into the
// state machine to check if we have a valid old Hangul
// composition.
// During the check we will also update the sortkey result in
// JamoSortInfo

// Find the new record
set JamoStateData to call FindNewJamoState(CurrentCharacter,
                                           JamoStateData)

// we did not find a valid old Hangul composition for the current
// character so return the current Jamo class (JamoClass and
// NewJamoClass are identical)
if JamoStateData is null
    return NewJamoClass
end if

// We found a match, so update our info.
set JamoSortInfo to call UpdateJamoSortInfo(JamoClass,
                                           JamoStateData,
                                           JamoSortInfo)

// We are still in a valid old Hangul composition.
// Go check the next character.
increment CurrentIndex

end while CurrentIndex

set NewJamoClass to "Invalid Jamo Sequence"
return NewJamoClass

```

### 7.5.18 GetJamoStateData

```

-- GetJamoStateData
--
-- On Entry:  Character      - Unicode Character to get Jamo
--                               information for
--
-- On Exit:   JamoStateData - Jamo state information from the data
--                               file
--
-- Jamo State information looks like this in the database:
--
-- SORTTABLES
-- ...
--      JAMOSORT      395
-- ...
--      0x1172      4
--      0x1172 0x00 0x00 0x11 0x00 0x38 0x03 ; U+1172
--      0x1161 0x01 0x00 0x00 0x00 0x00 0x01 ; U+1172,1161
--      0x1175 0x01 0x00 0x11 0x1b 0x3a 0x00 ; U+1172,1161,1175
--      0x1169 0x01 0x00 0x11 0x1b 0x3f 0x00 ; U+1172,1169

```

```

Procedure GetJamoStateData (IN  Character : Unicode Character,
                           OUT JamoStateData : JamoStateDataType )

// Get the Jamo section for this character.
// if Character was 0x1172, this would access the following section:
// 0x1172  4
// 0x1172 0x00 0x00 0x11 0x00 0x38 0x03 ; U+1172          record 0
// 0x1161 0x01 0x00 0x00 0x00 0x00 0x01 ; U+1172,1161      record 1
// 0x1175 0x01 0x00 0x11 0x1b 0x3a 0x00 ; U+1172,1161,1175 record 2
// 0x1169 0x01 0x00 0x11 0x1b 0x3f 0x00 ; U+1172,1169      record 3
//      |   |   |   |   |   |   |
// Field 1   2   3   4   5   6   7      Comment

open section JamoSection
where name is SORTTABLES\JAMOSORT\[Character] from unisort.txt

// Now open the first record
select record JamoRecord from JamoSection
where record index is 0

// Now gather the information from that record.
set JamoStateData.OldHangulFlag   to JamoRecord.Field2
set JamoStateData.LeadingIndex    to JamoRecord.Field3
set JamoStateData.VowelIndex      to JamoRecord.Field4
set JamoStateData.TrailingIndex   to JamoRecord.Field5
set JamoStateData.ExtraWeight     to JamoRecord.Field6
set JamoStateData.TransitionCount to JamoRecord.Field7

// Remember the record
set JamoStateData.DataRecord to JamoRecord

return JamoStateData

```

### 7.5.19 FindNewJamoState

```

-- FindNewJamoState
--
-- On Entry:  JamoCharacter - Unicode Character to get Jamo
--              information for
--              JamoStateData - Current Jamo state information
--
-- On Exit:   JamoStateData - New Jamo state record from the data
--              file null if an appropriate state
--              record is not found.
--
Procedure FindNewJamoState(IN      JamoCharacter : Unicode Character
                          INOUT   JamoStateData : JamoStateDataType )

// The current JamoStateData.DataRecord points to the base record.
// There are JamoStateData.TransitionCount following records that may
// match the input JamoCharacter, we're looking for the first one
set DataRecord to JamoStateData.DataRecord

while JamoStateData.TransitionCount is greater than 0

    // advance to the next record in the data and test if
    // it is the correct record for JamoCharacter

```

```

advance DataRecord to next record in data table
if DataRecord.Field1 is equal to JamoCharacter
  // Found a record, get its info and return it
  // Now gather the information from that record.
  set JamoStateData.OldHangulFlag to JamoRecord.Field2
  set JamoStateData.LeadingIndex to JamoRecord.Field3
  set JamoStateData.VowelIndex to JamoRecord.Field4
  set JamoStateData.TrailingIndex to JamoRecord.Field5
  set JamoStateData.ExtraWeight to JamoRecord.Field6
  set JamoStateData.TransitionCount to JamoRecord.Field7

  // Remember the record
  set JamoStateData.DataRecord to JamoRecord

  return JamoStateData
end if
end while

// record not found, return null
set JamoStateData to null
return JamoStateData

```

## 7.5.20 UpdateJamoSortInfo

```

-- UpdateJamoSortInfo
--
-- On Entry:  JamoClass      - The current Jamo Class
--            JamoStateData - Information about the new character
--                      state
--            JamoSortInfo  - Information about the character state
--
-- On Exit:   JamoSortInfo  - Updated with information about the new
--                      state based on JamoClass and
--                      JamoSortData
--
Procedure UpdateJamoSortInfo(IN JamoClass : enumeration,
                           IN JamoStateData : JamoStateDataType
                           INOUT JamoSortInfo : JamoSortInfoType )

// Record if this is a Jamo unique to old Hangul
set JamoSortInfo.OldHangulFlag to
  JamoSortInfo.OldHangulFlag | JamoStateData.OldHangulFlag

// Update the indices if the new ones are higher than the current
// ones.
if JamoStateData.LeadingIndex is greater than
  JamoSortInfo.LeadingIndex
  set JamoSortInfo.LeadingIndex to JamoStateData.LeadingIndex;
end if

if JamoStateData.VowelIndex is greater than JamoSortInfo.VowelIndex
  set JamoSortInfo.VowelIndex to JamoStateData.VowelIndex;
end if

if JamoStateData.TrailingIndex is greater than
  JamoSortInfo.TrailingIndex
  set JamoSortInfo.TrailingIndex to JamoStateData.TrailingIndex;

```

```

end if

// Update the extra weights according to the current Jamo class.
case JamoClass
    "Leading Jamo Class" :
        if JamoStateData.ExtraWeight is greater than
            JamoSortInfo.LeadingWeight
            set JamoSortInfo.LeadingWeight to JamoStateData.ExtraWeight
        end if

        "Vowel Jamo Class" :
            if JamoStateData.ExtraWeight is greater than
                JamoSortInfo.VowelWeight
                set JamoSortInfo.VowelWeight to JamoStateData.ExtraWeight
            end if

        "Trailing Jamo Class" :
            if JamoStateData.ExtraWeight is greater than
                JamoSortInfo.TrailingWeight
                set JamoSortInfo.TrailingWeight to
                    JamoStateData.ExtraWeight
            end if
    end case

return JamoSortInfo

```

### 7.5.21 IsJamo

```

-- IsJamo
--
-- On Entry:  SourceCharacter - Unicode Character to test
--
-- On Exit:   Result           - true if SourceCharacter is in the
--                               Jamo range
--
Procedure IsJamoLeading(IN SourceCharacter : Unicode Character
                      OUT Result : Boolean )

if (SourceCharacter is greater than or equal to NLS_CHAR_FIRST_JAMO)
    and
    (SourceCharacter is less than or equal to NLS_CHAR_LAST_JAMO)
    set Result to true
else
    set Result to false
end if

return Result

```

### 7.5.22 IsJamoLeading

```

-- IsJamoLeading
--
-- On Entry:  SourceCharacter - Unicode Character to test
--
-- On Exit:   Result           - true if SourceCharacter is a leading

```

```

--                                     Jamo
--
-- NOTE: Only call this if the character is known to be a Jamo
--       syllable
--       This function only helps distinguish between the different
--       types of Jamo, so only call it if IsJamo() has returned true.
--
Procedure IsJamoLeading(IN SourceCharacter : Unicode Character
                      OUT Result : Boolean )

if SourceCharacter is less than NLS_CHAR_FIRST_VOWEL_JAMO
    set Result to true
else
    set Result to false
end if

return Result

```

### 7.5.23 IsJamoTrailing

```

-- IsJamoTrailing
--
-- On Entry:  SourceCharacter - Unicode Character to test
--
-- On Exit:   Result          - true if this is a trailing Jamo
--
-- NOTE: Only call this if the character is known to be a Jamo
--       syllable
--       This function only helps distinguish between the different
--       types of Jamo, so only call it if IsJamo() has returned true.
--
Procedure IsJamoTrailing(IN SourceCharacter : Unicode Character
                       OUT Result : Boolean )

if SourceCharacter is greater than or equal to
    NLS_CHAR_FIRST_VOWEL_JAMO
    set Result to true
else
    set Result to false
end if

return Result

```

### 7.5.24 InitKoreanScriptMap

```

-- InitKoreanScriptMap
--
-- On Entry:  global KoreanScriptMap - presumed to be null
--
-- On Exit:   global KoreanScriptMap - initialized to map scripts to
--                                     Korean sort values
--
-- This procedure initializes the Korean script mapping table used to
-- cause the ideographic scripts to sort prior to other scripts for
-- the Korean sort

```

```

--
Procedure InitKoreanScriptMap
set KoreanScriptMap to new array of 256 null bytes

// Initialize the "scripts" prior to first script (Latin, script 14)
for counter is 0 to FIRST_SCRIPT - 1
    set KoreanScriptMap[counter] to counter
end for counter

// For Korean the Ideographs sort to the first script,
// so start with that index
set NewScript to FIRST_SCRIPT

// Test if the IDEOGRAPH script is part of a multiple weights script

// for convenience we're hard coding the information from the
// unisort.txt section SORTTABLES\MULTIPLEWEIGHTS
// IDEOGRAPHS are 128 through 241, we map them to FIRST_SCRIPT through
// 127

for counter is IDEOGRAPH to 241
    set KoreanScriptMap[counter] to NewScript
    increment NewScript
end for

// Now set the remaining unset scripts the next NewScript value
for counter is 0 to MAX_SCRIPTS - 1
    // If the value has not been set yet, set it to the next value
    if KoreanScriptMap[counter] is null
        set KoreanScriptMap[counter] to NewScript
        increment NewScript
    end if
end for counter

```

## 8 Index

### A

[Abstract data model](#)  
Access checks - reads ([section 3.1.1.4.3](#), [section 3.1.1.4.4](#))  
[Active Directory schema](#)  
[AD/LDS DC publication](#)  
[AD/LDS special objects](#)  
[AD/LDS users](#)  
[Add operation](#)  
[Applicability](#)  
[Attributes](#)  
[Authentication security](#)  
[Authorization security](#)

### B

[Background tasks](#)  
[Bind proxies](#)

### C

[Capability negotiation](#)  
[Constants - DC](#)  
[constructed attributes - reads](#)  
[Continuations - reads](#)

### D

[Data model - abstract](#)  
DC  
    [AD/LDS DC publication](#)  
    [constants](#)  
    [DNS record registrations](#)  
    [LDAP Ping](#)  
    [locating](#)  
    [Mailslot Ping](#)  
    [name compression/decompression](#)  
    [NBNS background](#)  
    [NetBIOS broadcast](#)  
    [publishing and locating](#)  
    [structures](#)  
[Definitions - reads](#)  
[Defunct attributes and classes - effect on reads](#)  
[Delete operation](#)  
[DNS record registrations](#)  
Domain controller  
    [AD/LDS DC publication](#)  
    [constants](#)  
    [DNS record registrations](#)  
    [LDAP Ping](#)  
    [locating](#)  
    [Mailslot Ping](#)  
    [name compression/decompression](#)  
    [NBNS background](#)  
    [NetBIOS broadcast](#)  
    [publishing and locating](#)  
    [structures](#)  
Domain join

[Active Directory state](#)  
[machine state](#)  
[overview](#)  
[relationship to protocols](#)  
[DS\\_REPL\\_ATTR\\_META\\_DATA\\_BLOB packet](#)  
[DS\\_REPL\\_CURSOR\\_BLOB packet](#)  
[DS\\_REPL\\_KCC\\_DSA\\_FAILUREW\\_BLOB packet](#)  
[DS\\_REPL\\_OPW\\_BLOB packet](#)  
[DS\\_REPL\\_QUEUE\\_STATISTICSW\\_BLOB packet](#)  
[DS\\_REPL\\_VALUE\\_META\\_DATA\\_BLOB packet](#)  
[dSHeuristics packet](#)  
[DynamicObject invariants](#)

### E

EXAMPLEPACKET packet ([section 2.2.2](#), [section 7.1.6.8.1](#))  
[Examples](#)  
[Extended access checks - reads](#)

### F

[Fields - vendor-extensible](#)  
Forest invariants  
    [dynamicObject invariant](#)  
    [introduction](#)  
    [overview](#)  
[Format of referent of pmsgOut.V1.pLog packet](#)  
[FSMO roles](#)

### G

[Glossary](#)

### I

[Index of version-specific behavior](#)  
[Informative references](#)  
[Introduction](#)

### K

KCC  
    [described](#)  
    [overview](#)  
    [references](#)  
Knowledge Consistency Checker  
    [described](#)  
    [overview](#)  
    [references](#)

### L

[LDAP attributes](#)  
[LDAP conformance](#)  
[LDAP extensions](#)  
[LDAP Ping](#)  
[LDAP protocol](#)  
[LDAP security](#)

[LSAPR\\_AUTH\\_INFORMATION packet](#)

## M

[Mailslot Ping](#)

Messages

[overview](#)

[security](#)

[syntax](#)

[transport](#)

[Modify DN](#)

[Modify operation](#)

[msDS-TrustForestTrustInfo Attribute packet](#)

## N

[Name compression/decompression](#)

[NBNS background](#)

[NC - schema](#)

[NetBIOS broadcast](#)

[NETLOGON SAM LOGON REQUEST packet](#)

[NETLOGON SAM LOGON RESPONSE packet](#)

[NETLOGON SAM LOGON RESPONSE EX packet](#)

[NETLOGON SAM LOGON RESPONSE NT40 packet](#)

[Normative references](#)

[NT4 replication support](#)

## O

Objects

[introduction](#)

[overview](#)

[trust objects](#)

[Overview](#)

## P

[Preconditions](#)

[Prerequisites](#)

[Proxies - bind](#)

## R

Reads

[access checks](#)

[constructed attributes](#)

[continuations](#)

[definitions](#)

[defunct attributes and classes effect](#)

[extended access checks](#)

[introduction](#)

[overview](#)

[referrals](#)

[Record packet](#)

References

[informative](#)

[normative](#)

[overview](#)

[Referrals - reads](#)

[Relationship to other protocols](#)

[Replication - NT4 support](#)

[rootDSE attributes](#)

[rootDSE modify operations](#)

## S

[SCHEDULE packet](#)

[SCHEDULE\\_HEADER packet](#)

[Schema NC](#)

Security

[authentication](#)

[authorization](#)

[LDAP](#)

[messages](#)

[overview](#)

[Security descriptor invariants](#)

[Standards assignments](#)

[State model](#)

[Structures - DC](#)

[Syntax](#)

## T

[Transport](#)

[Trust objects](#)

## U

[Unicode string comparisons](#)

Updates

[add operation](#)

[delete operation](#)

[generally](#)

[modify DN](#)

[modify operation](#)

[overview](#)

## V

[Vendor-extensible fields](#)

[Versioning](#)

[Version-specific behavior](#)