

[MS-DRSR]: Directory Replication Service (DRS) Remote Protocol 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.
- 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.

Date	Revision History	Revision Class	Comments
07/10/2007	1.3	Minor	Added informative reference; minor updates to content.
08/17/2007	1.3.1	Editorial	Revised and edited the technical content.
09/21/2007	1.4	Minor	Revised references.
10/26/2007	2.0	Major	Updated and revised the technical content.
01/25/2008	2.0.1	Editorial	Revised and edited the technical content.

Table of Contents

1	Introduction	18
1.1	Glossary	18
1.1.1	Pervasive Concepts	18
1.1.2	Glossary Entries	19
1.2	References	30
1.2.1	Normative References	30
1.2.2	Informative References	32
1.3	Protocol Overview (Synopsis)	32
1.3.1	Methods Categorized by Function	32
1.3.2	Sequencing Issues	33
1.3.3	Most Frequently Used Types	35
1.4	Relationship to Other Protocols	35
1.5	Prerequisites/Preconditions	36
1.6	Applicability Statement	36
1.7	Versioning and Capability Negotiation	36
1.8	Vendor-Extensible Fields	37
1.9	Standards Assignments	37
2	Message Transport	38
2.1	RPC Transport	38
2.2	Protocol Security	38
2.2.1	General Background	38
2.2.2	Service Principal Names for Domain Controllers	38
2.2.3	DC-to-DC Operations	39
2.2.3.1	Security Provider	39
2.2.3.2	SPN for a Target DC in AD/DS	39
2.2.3.3	SPN for a Target DC in AD/LDS	40
2.2.4	Client-to-DC Operations	40
2.2.4.1	Security Provider	40
2.2.4.2	SPN for a Target DC in AD/DS	41
2.2.4.3	SPN for a Target DC in AD/LDS	42
3	Background to Behavior Specifications	43
3.1	Document Organization	43
3.2	Typographical Conventions	43
3.3	State Model	44
3.3.1	Preliminaries	44
3.3.2	Transactions	44
3.3.3	Concrete and Abstract Types	44
3.4	Pseudocode Language	45
3.4.1	Naming Conventions	45
3.4.2	Language Constructs for Concrete Types	46
3.4.3	Language Constructs for Abstract Types	46
3.4.4	Common Language Constructs	48
3.4.5	Access to Objects and Their Attributes	49
3.4.6	Asynchronous Processing	52
3.5	Conventions for Protocol Examples	52
3.5.1	Common Configuration	52
3.5.2	Data Display Conventions	54
3.6	Server and Client Initialization	54
3.6.1	AD/LDS Specifics	55
4	RPC Methods and Their Behavior	56

4.1	drsuapi RPC Interface	56
4.1.1	IDL_DRSAddEntry (Opnum 17)	58
4.1.1.1	Method-Specific Concrete Types.....	58
4.1.1.1.1	DRS_MSG_ADDENTRYREQ.....	58
4.1.1.1.2	DRS_MSG_ADDENTRYREQ_V1	59
4.1.1.1.3	DRS_MSG_ADDENTRYREQ_V2	59
4.1.1.1.4	DRS_MSG_ADDENTRYREQ_V3	59
4.1.1.1.5	DRS_MSG_ADDENTRYREPLY.....	60
4.1.1.1.6	DRS_MSG_ADDENTRYREPLY_V1	60
4.1.1.1.7	DRS_MSG_ADDENTRYREPLY_V2	61
4.1.1.1.8	DRS_MSG_ADDENTRYREPLY_V3	61
4.1.1.1.9	ADDENTRY_REPLY_INFO	62
4.1.1.1.10	DIRERR_DRS_WIRE_V1	62
4.1.1.1.11	ATRERR_DRS_WIRE_V1	63
4.1.1.1.12	PROBLEMLIST_DRS_WIRE_V1.....	63
4.1.1.1.13	INTFORMPROB_DRS_WIRE_V1	63
4.1.1.1.14	NAMERR_DRS_WIRE_V1	64
4.1.1.1.15	REFERR_DRS_WIRE_V1	64
4.1.1.1.16	NAMERESOP_DRS_WIRE_V1.....	64
4.1.1.1.17	DSA_ADDRESS_LIST_DRS_WIRE_V1.....	65
4.1.1.1.18	CONTREF_DRS_WIRE_V1	65
4.1.1.1.19	SECERR_DRS_WIRE_V1	66
4.1.1.1.20	SVCERR_DRS_WIRE_V1	67
4.1.1.1.21	UPDERR_DRS_WIRE_V1.....	67
4.1.1.1.22	SYSERR_DRS_WIRE_V1	67
4.1.1.1.23	DRS_ERROR_DATA.....	68
4.1.1.1.24	DRS_ERROR_DATA_V1	68
4.1.1.1.25	DIRERR Codes	68
4.1.1.1.26	PROBLEM Error Codes	69
4.1.1.2	Method-Specific Abstract Types and Procedures.....	70
4.1.1.2.1	ConstructReplSpn	70
4.1.1.2.2	CreateCrossRef	71
4.1.1.2.3	CreateNtdsDsa	72
4.1.1.2.4	UseCredsForAccessCheck	74
4.1.1.2.5	IsDomainToBeCreated.....	74
4.1.1.2.6	GetDomainNameFromEntinf.....	74
4.1.1.2.7	SetErrorData.....	74
4.1.1.2.8	ClientIpMatch.....	75
4.1.1.2.9	PerformModifyOperation	75
4.1.1.3	Server Behavior of the IDL_DRSAddEntry Method	75
4.1.2	IDL_DRSAddSidHistory (Opnum 20)	78
4.1.2.1	Method-Specific Concrete Types.....	79
4.1.2.1.1	DRS_MSG_ADDSIDREQ.....	79
4.1.2.1.2	DRS_MSG_ADDSIDREQ_V1	79
4.1.2.1.3	DRS_MSG_ADDSIDREPLY.....	80
4.1.2.1.4	DRS_MSG_ADDSIDREPLY_V1	80
4.1.2.1.5	DRS_ADDSID_FLAGS.....	80
4.1.2.2	Method-Specific Abstract Types and Procedures.....	81
4.1.2.2.1	ConnectionCtx.....	81
4.1.2.2.2	ConnectToDC	81
4.1.2.2.3	ConnectToDCWithCreds.....	81
4.1.2.2.4	GenerateFailureAudit	82
4.1.2.2.5	GenerateSuccessAudit.....	82
4.1.2.2.6	GenerateSuccessAuditRemotely	82
4.1.2.2.7	GetKeyLength	82

4.1.2.2.8	GetPDC	82
4.1.2.2.9	HasAdminRights	82
4.1.2.2.10	IsAuditingEnabledForNc.....	83
4.1.2.2.11	IsLocalRpcCall.....	83
4.1.2.2.12	IsNT4SP4OrBetter	83
4.1.2.2.13	IsWellKnownDomainRelativeSid.....	83
4.1.2.2.14	LastRID.....	83
4.1.2.2.15	RemoteQuery.....	83
4.1.2.3	Server Behavior of the IDL_DRSSAddSidHistory Method	84
4.1.3	IDL_DRSSBind (Opnum 0).....	91
4.1.3.1	Client Behavior When Sending the IDL_DRSSBind Request	91
4.1.3.2	Server Behavior of the IDL_DRSSBind Method.....	96
4.1.3.3	Client Behavior When Receiving the IDL_DRSSBind Response	100
4.1.3.4	Examples of the IDL_DRSSBind Method.....	100
4.1.3.4.1	Initial State	100
4.1.3.4.2	Client Request.....	101
4.1.3.4.3	Server Response	102
4.1.3.4.4	Final State.....	103
4.1.4	IDL_DRSSCrackNames (Opnum 12).....	104
4.1.4.1	Method-Specific Concrete Types.....	104
4.1.4.1.1	DRS_MSG_CRACKREQ	104
4.1.4.1.2	DRS_MSG_CRACKREQ_V1	104
4.1.4.1.3	DS_NAME_FORMAT	106
4.1.4.1.4	DS_NAME_RESULT_ITEMW	107
4.1.4.1.5	DS_NAME_RESULTW	108
4.1.4.1.6	DRS_MSG_CRACKREPLY.....	108
4.1.4.1.7	DRS_MSG_CRACKREPLY_V1	108
4.1.4.1.8	DS_NAME_ERROR	108
4.1.4.2	Method-Specific Abstract Types and Procedures.....	109
4.1.4.2.1	CanonicalNameFromCanonicalNameEx	109
4.1.4.2.2	DomainDNSNameFromDomain	110
4.1.4.2.3	DomainFromDomainDNSName	110
4.1.4.2.4	DomainNameFromCanonicalName	110
4.1.4.2.5	DomainNameFromSid	110
4.1.4.2.6	DomainNameFromUPN	110
4.1.4.2.7	DomainNetBIOSNameFromDomain	110
4.1.4.2.8	DomainSidFromSid	111
4.1.4.2.9	LookupName.....	111
4.1.4.2.10	LookupAttr	113
4.1.4.2.11	LookupCanonicalName	114
4.1.4.2.12	GetCanonicalName	114
4.1.4.2.13	LookupSPN	114
4.1.4.2.14	LookupSID	115
4.1.4.2.15	LookupUnknownName.....	116
4.1.4.2.16	LookupUPNAndAltSecID	116
4.1.4.2.17	LookupDomainSyntactically	117
4.1.4.2.18	MapSPN	118
4.1.4.2.19	ParseCanonicalName.....	118
4.1.4.2.20	RetrieveDCSuffixFromDn	119
4.1.4.2.21	UserNameFromUPN	119
4.1.4.2.22	ConstructOutput.....	119
4.1.4.3	Server Behavior of the IDL_DRSSCrackNames Method	120
4.1.4.4	Examples of the IDL_DRSSCrackNames Method.....	124
4.1.4.4.1	Initial State	124
4.1.4.4.2	Client Request.....	125

4.1.4.4.3	Server Response	125
4.1.4.4.4	Final State.....	126
4.1.5	IDL_DRSDomainControllerInfo (Opnum 16)	126
4.1.5.1	Method-Specific Concrete Types.....	126
4.1.5.1.1	DRS_MSG_DCINFOREQ	126
4.1.5.1.2	DRS_MSG_DCINFOREQ_V1	127
4.1.5.1.3	DRS_MSG_DCINFOREPLY	127
4.1.5.1.4	DRS_MSG_DCINFOREPLY_V1	128
4.1.5.1.5	DRS_MSG_DCINFOREPLY_V2	128
4.1.5.1.6	DRS_MSG_DCINFOREPLY_V3	128
4.1.5.1.7	DRS_MSG_DCINFOREPLY_VFFFFFFF	128
4.1.5.1.8	DS_DOMAIN_CONTROLLER_INFO_1W	129
4.1.5.1.9	DS_DOMAIN_CONTROLLER_INFO_2W	129
4.1.5.1.10	DS_DOMAIN_CONTROLLER_INFO_3W	130
4.1.5.1.11	DS_DOMAIN_CONTROLLER_INFO_FFFFFFFW	131
4.1.5.2	Server Behavior of the IDL_DRSDomainControllerInfo Method	132
4.1.5.3	Examples of the IDL_DRSDomainControllerInfo Method	136
4.1.5.3.1	Initial State	136
4.1.5.3.2	Client Request.....	140
4.1.5.3.3	Server Response	140
4.1.5.3.4	Final State.....	141
4.1.6	IDL_DRSExecuteKCC (Opnum 18)	141
4.1.6.1	Method-Specific Concrete Types.....	142
4.1.6.1.1	DRS_MSG_KCC_EXECUTE	142
4.1.6.1.2	DRS_MSG_KCC_EXECUTE_V1	142
4.1.6.2	Method-Specific Abstract Types and Procedures.....	142
4.1.6.2.1	ExecuteKCCTasks	142
4.1.6.3	Server Behavior of the IDL_DRSExecuteKCC Method	143
4.1.7	IDL_DRSFinishDemotion (Opnum 27).....	143
4.1.7.1	Method-Specific Concrete Types.....	144
4.1.7.1.1	DRS_MSG_FINISH_DEMOTIONREQ	144
4.1.7.1.2	DRS_MSG_FINISH_DEMOTIONREQ_V1	144
4.1.7.1.3	DRS_MSG_FINISH_DEMOTIONREPLY	145
4.1.7.1.4	DRS_MSG_FINISH_DEMOTIONREPLY_V1	145
4.1.7.2	Method-Specific Abstract Types and Procedures.....	146
4.1.7.2.1	RemoveADLDSServer.....	146
4.1.7.2.2	RemoveADLDSSCP	146
4.1.7.2.3	RemoveADLDSSPNs.....	146
4.1.7.3	Server Behavior of the IDL_DRSFinishDemotion Method	146
4.1.8	IDL_DRSGetMemberships (Opnum 9).....	148
4.1.8.1	Method-Specific Concrete Types.....	149
4.1.8.1.1	DRS_MSG_REVMEMB_REQ	149
4.1.8.1.2	DRS_MSG_REVMEMB_REQ_V1	149
4.1.8.1.3	REVERSE_MEMBERSHIP_OPERATION_TYPE	150
4.1.8.1.4	DRS_MSG_REVMEMB_REPLY.....	151
4.1.8.1.5	DRS_MSG_REVMEMB_REPLY_V1	151
4.1.8.1.6	SE_GROUP Values	152
4.1.8.2	Method-Specific Abstract Types and Procedures.....	152
4.1.8.2.1	Arc and ArcSet	152
4.1.8.2.2	Closure	152
4.1.8.2.3	DomainOf.....	152
4.1.8.2.4	GetDSNameFromPrimaryGroupId	152
4.1.8.2.5	IsMatchedGroup	153
4.1.8.2.6	Neighbors.....	153
4.1.8.3	Server Behavior of the IDL_DRSGetMemberships Method.....	154

4.1.9	IDL_DRSGetMemberships2 (Opnum 21)	156
4.1.9.1	Method-Specific Concrete Types	157
4.1.9.1.1	DRS_MSG_GETMEMBERSHIPS2_REQ	157
4.1.9.1.2	DRS_MSG_GETMEMBERSHIPS2_REQ_V1	157
4.1.9.1.3	DRS_MSG_GETMEMBERSHIPS2_REPLY	157
4.1.9.1.4	DRS_MSG_GETMEMBERSHIPS2_REPLY_V1	158
4.1.9.2	Server Behavior of the IDL_DRSGetMemberships2 Method	158
4.1.10	IDL_DRSGetNCChanges (Opnum 3)	158
4.1.10.1	Overview	159
4.1.10.1.1	Cycle Start and Finish	159
4.1.10.1.2	Cycle Goal	160
4.1.10.1.3	Extended Operations	161
4.1.10.2	Method-Specific Concrete Types	161
4.1.10.2.1	DRS_MSG_GETCHGREQ	161
4.1.10.2.2	DRS_MSG_GETCHGREQ_V3	161
4.1.10.2.3	DRS_MSG_GETCHGREQ_V4	162
4.1.10.2.4	DRS_MSG_GETCHGREQ_V5	163
4.1.10.2.5	DRS_MSG_GETCHGREQ_V7	163
4.1.10.2.6	DRS_MSG_GETCHGREQ_V8	164
4.1.10.2.7	DRS_MSG_GETCHGREPLY	165
4.1.10.2.8	DRS_MSG_GETCHGREPLY_V1	165
4.1.10.2.9	DRS_MSG_GETCHGREPLY_V2	166
4.1.10.2.10	DRS_MSG_GETCHGREPLY_V6	166
4.1.10.2.11	DRS_MSG_GETCHGREPLY_V7	167
4.1.10.2.12	COMPRESSED_DATA	168
4.1.10.2.13	DRS_COMP_ALG_TYPE	168
4.1.10.2.14	DRS_COMPRESSED_BLOB	169
4.1.10.2.15	ENCRYPTED_PAYLOAD	169
4.1.10.2.16	EXOP_ERR Codes	170
4.1.10.2.17	EXOP_REQ Codes	171
4.1.10.2.18	PROPERTY_META_DATA	171
4.1.10.3	Method-Specific Abstract Types and Procedures	171
4.1.10.3.1	AbstractLinkValStampFromConcreteLinkValStamp	171
4.1.10.3.2	AbstractPASFromConcretePAS	172
4.1.10.3.3	AbstractUTDFromConcreteUTD	172
4.1.10.3.4	AttributeAndStamp	173
4.1.10.3.5	AttributeStampCompare	173
4.1.10.3.6	ConcretePASFromAbstractPAS	173
4.1.10.3.7	ConcreteUTDFromAbstractUTD	174
4.1.10.3.8	GetNCChangesV6Reply	174
4.1.10.3.9	GetStampsForUpdate	175
4.1.10.3.10	GetWellKnownObject	175
4.1.10.3.11	IsProtectedObject	176
4.1.10.3.12	IsSecretAttribute	176
4.1.10.3.13	IsUserIncluded	176
4.1.10.3.14	LinkValueStampCompare	176
4.1.10.3.15	ObjAtts	177
4.1.10.3.16	ObjAttVal	177
4.1.10.3.17	PerformModifyDNOperation	177
4.1.10.3.18	RemoveAttrVal	178
4.1.10.3.19	SetAttrStamp	178
4.1.10.3.20	SetAttrVal	178
4.1.10.3.21	SetLinkStamp	178
4.1.10.4	Client Behavior When Sending the IDL_DRSGetCChanges Request	178
4.1.10.4.1	ReplicateNCRequestMsg	179

4.1.10.4.2	ReplSingleObjRequestMsg	182
4.1.10.4.3	PerformExtendedOpRequestMsg	184
4.1.10.5	Server Behavior of the IDL_DRSGetNCChanges Method	186
4.1.10.5.1	TransformInput	189
4.1.10.5.2	GetReplChanges	190
4.1.10.5.3	GetReplScope	193
4.1.10.5.4	GetChangesInScope	194
4.1.10.5.5	FilterAttribute	195
4.1.10.5.6	GetResponseSubset	196
4.1.10.5.7	AddObjToResponse	197
4.1.10.5.8	UpdateRevealedList	198
4.1.10.5.9	AddLinkToResponse	199
4.1.10.5.10	EncryptValuesIfNecessary	200
4.1.10.5.11	ProcessFsmoRoleRequest	201
4.1.10.5.12	RevealSecretsPolicy	204
4.1.10.5.13	GetRevealSecretsPolicyForUser	205
4.1.10.5.14	RevealSecretsForUserAllowed	206
4.1.10.5.15	GetRoleScope	206
4.1.10.5.16	SortResponseLinks	207
4.1.10.5.17	TransformOutput	208
4.1.10.6	Client Behavior When Receiving the IDL_DRSGetNCChanges Response	210
4.1.10.6.1	ProcessGetNCChangesReply	210
4.1.10.6.2	PrepareCrossNCMove	212
4.1.10.6.3	AdjustInstanceTypeAttrVal	214
4.1.10.6.4	SetResetInstanceTypeBits	215
4.1.10.6.5	AddObject	215
4.1.10.6.6	UpdateObject	217
4.1.10.6.7	FindBestParentObject	219
4.1.10.6.8	ResolveNameConflict	220
4.1.10.6.9	MakeConflictDN	222
4.1.10.6.10	ProcessLinkValue	222
4.1.10.6.11	UpdateRepsFrom	223
4.1.10.6.12	UpdateUTDandPAS	224
4.1.10.6.13	DecryptValuesIfNecessary	225
4.1.10.6.14	DecompressReplyMessage	227
4.1.10.6.15	DecompressWin2k3	229
4.1.10.6.16	UndeleteObject	230
4.1.10.7	Examples of the IDL_DRSGetNCChanges Method - Add User	230
4.1.10.7.1	Initial State	230
4.1.10.7.2	Client Request	233
4.1.10.7.3	Server Response	234
4.1.10.7.4	Final State	235
4.1.10.8	Examples of the IDL_DRSGetNCChanges Method - Add User to a Group	237
4.1.10.8.1	Initial State	237
4.1.10.8.2	Client Request	239
4.1.10.8.3	Server Response	239
4.1.10.8.4	Final State	240
4.1.10.9	Examples of the IDL_DRSGetNCChanges Method - Change User Password	242
4.1.10.9.1	Initial State	242
4.1.10.9.2	Client Request	244
4.1.10.9.3	Server Response	245
4.1.10.9.4	Final State	246
4.1.11	IDL_DRSGetNT4ChangeLog (Opnum 11)	248
4.1.11.1	Method-Specific Concrete Types	248
4.1.11.1.1	DRS_MSG_NT4_CHGLOG_REQ	248

4.1.11.1.2	DRS_MSG_NT4_CHGLOG_REQ_V1	248
4.1.11.1.3	DRS_MSG_NT4_CHGLOG_REPLY	249
4.1.11.1.4	DRS_MSG_NT4_CHGLOG_REPLY_V1	249
4.1.11.1.5	NT4_REPLICATION_STATE	250
4.1.11.2	Method-Specific Abstract Types and Procedures	251
4.1.11.2.1	IsPDC	251
4.1.11.2.2	GetWindowsErrorCode	251
4.1.11.3	Server Behavior of the IDL_DRSGetNT4ChangeLog Method	251
4.1.11.4	Examples of the IDL_DRSGetNT4ChangeLog Method	254
4.1.11.4.1	Initial State	254
4.1.11.4.2	Client Request	254
4.1.11.4.3	Server Response	255
4.1.11.4.4	Final State	255
4.1.12	IDL_DRSGetObjectExistence (Opnum 23)	255
4.1.12.1	Method-Specific Concrete Types	256
4.1.12.1.1	DRS_MSG_EXISTREQ	256
4.1.12.1.2	DRS_MSG_EXISTREQ_V1	256
4.1.12.1.3	DRS_MSG_EXISTREPLY	257
4.1.12.1.4	DRS_MSG_EXISTREPLY_V1	257
4.1.12.2	Method-Specific Abstract Types and Procedures	257
4.1.12.2.1	GuidSequence	257
4.1.12.3	Client Behavior when Sending the IDL_DRSGetObjectExistence Request	258
4.1.12.4	Server Behavior of the IDL_DRSGetObjectExistence Method	259
4.1.12.5	Client Behavior when Receiving the IDL_DRSGetObjectExistence Response	260
4.1.13	IDL_DRSGetReplInfo (Opnum 19)	260
4.1.13.1	Method-Specific Concrete Types	261
4.1.13.1.1	DRS_MSG_GETREPLINFO_REQ	261
4.1.13.1.2	DRS_MSG_GETREPLINFO_REQ_V1	261
4.1.13.1.3	DRS_MSG_GETREPLINFO_REQ_V2	261
4.1.13.1.4	DS_REPL_INFO Codes	262
4.1.13.1.5	DRS_MSG_GETREPLINFO_REPLY	263
4.1.13.1.6	DS_REPL_NEIGHBORSW	264
4.1.13.1.7	DS_REPL_NEIGHBORW	264
4.1.13.1.8	DS_REPL_CURSORS	266
4.1.13.1.9	DS_REPL_CURSOR	266
4.1.13.1.10	DS_REPL_CURSORS_2	266
4.1.13.1.11	DS_REPL_CURSOR_2	267
4.1.13.1.12	DS_REPL_CURSORS_3W	267
4.1.13.1.13	DS_REPL_CURSOR_3W	267
4.1.13.1.14	DS_REPL_OBJ_META_DATA	268
4.1.13.1.15	DS_REPL_ATTR_META_DATA	268
4.1.13.1.16	DS_REPL_OBJ_META_DATA_2	269
4.1.13.1.17	DS_REPL_ATTR_META_DATA_2	269
4.1.13.1.18	DS_REPL_KCC_DSA_FAILURESW	270
4.1.13.1.19	DS_REPL_KCC_DSA_FAILUREW	270
4.1.13.1.20	DS_REPL_PENDING_OPSW	270
4.1.13.1.21	DS_REPL_OPW	271
4.1.13.1.22	DS_REPL_ATTR_VALUE_META_DATA	272
4.1.13.1.23	DS_REPL_VALUE_META_DATA	272
4.1.13.1.24	DS_REPL_ATTR_VALUE_META_DATA_2	273
4.1.13.1.25	DS_REPL_VALUE_META_DATA_2	273
4.1.13.1.26	DS_REPL_CLIENT_CONTEXTS	274
4.1.13.1.27	DS_REPL_CLIENT_CONTEXT	274
4.1.13.1.28	DS_REPL_SERVER_OUTGOING_CALLS	275
4.1.13.1.29	DS_REPL_SERVER_OUTGOING_CALL	275

4.1.13.2	Method-Specific Abstract Types and Procedures.....	277
4.1.13.2.1	GetDNFromInvocationID	277
4.1.13.2.2	GetDNFromObjectGuid	277
4.1.13.2.3	GetNCs	277
4.1.13.3	Server Behavior of the IDL_DRSGetReplInfo Method	277
4.1.14	IDL_DRSInitDemotion (Opnum 25)	289
4.1.14.1	Method-Specific Concrete Types.....	289
4.1.14.1.1	DRS_MSG_INIT_DEMOTIONREQ	289
4.1.14.1.2	DRS_MSG_INIT_DEMOTIONREQ_V1	290
4.1.14.1.3	DRS_MSG_INIT_DEMOTIONREPLY	290
4.1.14.1.4	DRS_MSG_INIT_DEMOTIONREPLY_V1	290
4.1.14.2	Server Behavior of the IDL_DRSInitDemotion Method	291
4.1.15	IDL_DRSInterDomainMove (Opnum 10)	291
4.1.15.1	Method-Specific Concrete Types.....	292
4.1.15.1.1	DRS_MSG_MOVEREQ	292
4.1.15.1.2	DRS_MSG_MOVEREQ_V1	292
4.1.15.1.3	DRS_MSG_MOVEREQ_V2	293
4.1.15.1.4	DRS_MSG_MOVEREPLY	293
4.1.15.1.5	DRS_MSG_MOVEREPLY_V1.....	294
4.1.15.1.6	DRS_MSG_MOVEREPLY_V2.....	294
4.1.15.2	Method-Specific Abstract Types and Procedures.....	294
4.1.15.2.1	AttrIsBacklink	294
4.1.15.2.2	AttrIsConstructed	294
4.1.15.2.3	AttrIsNonReplicated	295
4.1.15.2.4	AuthorizationInfoFromClientCredentials.....	295
4.1.15.2.5	ImpersonateAuthorizationInfo	295
4.1.15.2.6	IsApplicationNC.....	295
4.1.15.2.7	RevertToSelf.....	295
4.1.15.3	Server Behavior of the IDL_DRSInterDomainMove Method	295
4.1.15.4	Examples of the IDL_DRSInterDomainMove Method	299
4.1.15.4.1	Initial State	299
4.1.15.4.2	Client Request.....	300
4.1.15.4.3	Server Response	302
4.1.15.4.4	Final State.....	302
4.1.16	IDL_DRSQuerySitesByCost (Opnum 24)	304
4.1.16.1	Method-Specific Concrete Types.....	304
4.1.16.1.1	DRS_MSG_QUERYSITESREQ	304
4.1.16.1.2	DRS_MSG_QUERYSITESREQ_V1	304
4.1.16.1.3	DRS_MSG_QUERYSITESREPLY	305
4.1.16.1.4	DRS_MSG_QUERYSITESREPLY_V1.....	305
4.1.16.1.5	DRS_MSG_QUERYSITESREPLYELEMENT_V1.....	305
4.1.16.2	Method-Specific Abstract Types and Procedures.....	306
4.1.16.2.1	ValidateSiteRDN.....	306
4.1.16.2.2	WeightedArc and WeightedArcSet.....	306
4.1.16.2.3	MinWeightPath	306
4.1.16.3	Server Behavior of the IDL_DRSQuerySitesByCost Method	307
4.1.17	IDL_DRSRemoveDsDomain (Opnum 15).....	309
4.1.17.1	Method-Specific Concrete Types.....	310
4.1.17.1.1	DRS_MSG_RMDMNREQ	310
4.1.17.1.2	DRS_MSG_RMDMNREQ_V1.....	310
4.1.17.1.3	DRS_MSG_RMDMNREPLY	310
4.1.17.1.4	DRS_MSG_RMDMNREPLY_V1	311
4.1.17.2	Method-Specific Abstract Types and Procedures.....	311
4.1.17.2.1	HasNCReplicated	311
4.1.17.3	Server Behavior of the IDL_DRSRemoveDsDomain Method	311

4.1.18	IDL_DRSRemoveDsServer (Opnum 14)	313
4.1.18.1	Method-Specific Concrete Types	313
4.1.18.1.1	DRS_MSG_RMSVRREQ	313
4.1.18.1.2	DRS_MSG_RMSVRREQ_V1	314
4.1.18.1.3	DRS_MSG_RMSVRREPLY	314
4.1.18.1.4	DRS_MSG_RMSVRREPLY_V1	314
4.1.18.2	Server Behavior of the IDL_DRSRemoveDsServer Method	315
4.1.19	IDL_DRSReplicaAdd (Opnum 5)	317
4.1.19.1	Method-Specific Concrete Types	318
4.1.19.1.1	DRS_MSG_REPADD	318
4.1.19.1.2	DRS_MSG_REPADD_V1	318
4.1.19.1.3	DRS_MSG_REPADD_V2	318
4.1.19.2	Server Behavior of the IDL_DRSReplicaAdd Method	319
4.1.20	IDL_DRSReplicaDel (Opnum 6)	321
4.1.20.1	Method-Specific Concrete Types	322
4.1.20.1.1	DRS_MSG_REPDEL	322
4.1.20.1.2	DRS_MSG_REPDEL_V1	322
4.1.20.2	Server Behavior of the IDL_DRSReplicaDel Method	323
4.1.21	IDL_DRSReplicaDemotion (Opnum 26)	324
4.1.21.1	Method-Specific Concrete Types	325
4.1.21.1.1	DRS_MSG_REPLICA_DEMOTIONREQ	325
4.1.21.1.2	DRS_MSG_REPLICA_DEMOTIONREQ_V1	325
4.1.21.1.3	DRS_MSG_REPLICA_DEMOTIONREPLY	326
4.1.21.1.4	DRS_MSG_REPLICA_DEMOTIONREPLY_V1	326
4.1.21.2	Method-Specific Abstract Types and Procedures	326
4.1.21.2.1	ReplicationPartners()	326
4.1.21.2.2	BindToDSA()	327
4.1.21.2.3	UnbindFromDSA()	327
4.1.21.2.4	AbandonAllFSMORoles()	327
4.1.21.2.5	ReplicateOffChanges()	328
4.1.21.3	Server Behavior of the IDL_DRSReplicaDemotion Method	329
4.1.22	IDL_DRSReplicaModify (Opnum 7)	330
4.1.22.1	Method-Specific Concrete Types	330
4.1.22.1.1	DRS_MSG_REPMOD	330
4.1.22.1.2	DRS_MSG_REPMOD_V1	331
4.1.22.2	Server Behavior of the IDL_DRSReplicaModify Method	331
4.1.23	IDL_DRSReplicaSync (Opnum 2)	333
4.1.23.1	Method-Specific Concrete Types	333
4.1.23.1.1	DRS_MSG_REPSYNC	333
4.1.23.1.2	DRS_MSG_REPSYNC_V1	333
4.1.23.2	Server Behavior of the IDL_DRSReplicaSync Method	334
4.1.24	IDL_DRSReplicaVerifyObjects (Opnum 22)	335
4.1.24.1	Method-Specific Concrete Types	336
4.1.24.1.1	DRS_MSG_REPVERIFYOBJ	336
4.1.24.1.2	DRS_MSG_REPVERIFYOBJ_V1	336
4.1.24.2	Method-Specific Abstract Types and Procedures	336
4.1.24.2.1	GetRemoteUTD	336
4.1.24.2.2	ObjectExistsAtDC	336
4.1.24.3	Server Behavior of the IDL_DRSReplicaVerifyObjects Method	337
4.1.25	IDL_DRSUnbind (Opnum 1)	338
4.1.25.1	Server Behavior of the IDL_DRSUnbind Method	338
4.1.26	IDL_DRSUpdateRefs (Opnum 4)	339
4.1.26.1	Method-Specific Concrete Types	339
4.1.26.1.1	DRS_MSG_UPDREFS	339
4.1.26.1.2	DRS_MSG_UPDREFS_V1	339

4.1.26.2	Server Behavior of the IDL_DRSUpdateRefs Method	340
4.1.26.3	Examples of the IDL_DRSUpdateRefs Method	341
4.1.26.3.1	Initial State	341
4.1.26.3.2	Client Request.....	341
4.1.26.3.3	Server Response	342
4.1.26.3.4	Final State.....	342
4.1.27	IDL_DRSVerifyNames (Opnum 8)	342
4.1.27.1	Method-Specific Concrete Types.....	343
4.1.27.1.1	DRS_MSG_VERIFYREQ	343
4.1.27.1.2	DRS_MSG_VERIFYREQ_V1	343
4.1.27.1.3	DRS_MSG_VERIFYREPLY	344
4.1.27.1.4	DRS_MSG_VERIFYREPLY_V1.....	344
4.1.27.2	Server Behavior of the IDL_DRSVerifyNames Method	345
4.1.27.3	Examples of the IDL_DRSVerifyNames Method	348
4.1.27.3.1	Initial State	348
4.1.27.3.2	Client Request.....	348
4.1.27.3.3	Server Response	349
4.1.27.3.4	Final State.....	349
4.1.28	IDL_DRSWriteSPN (Opnum 13)	349
4.1.28.1	Method-Specific Concrete Types.....	350
4.1.28.1.1	DRS_MSG_SPNREQ	350
4.1.28.1.2	DRS_MSG_SPNREQ_V1	350
4.1.28.1.3	DRS_MSG_SPNREPLY.....	351
4.1.28.1.4	DRS_MSG_SPNREPLY_V1	351
4.1.28.1.5	DS_SPN_OPERATION	351
4.1.28.2	Server Behavior of the IDL_DRSWriteSPN Method	352
4.2	dsaop RPC Interface	354
4.2.1	IDL_DSAPrepareScript (Opnum 0)	354
4.2.1.1	Method-Specific Concrete Types.....	354
4.2.1.1.1	DSA_MSG_PREPARE_SCRIPT_REQ.....	354
4.2.1.1.2	DSA_MSG_PREPARE_SCRIPT_REQ_V1	355
4.2.1.1.3	DSA_MSG_PREPARE_SCRIPT_REPLY.....	355
4.2.1.1.4	DSA_MSG_PREPARE_SCRIPT_REPLY_V1	355
4.2.1.2	Method-Specific Abstract Types and Procedures	356
4.2.1.2.1	GetKeyLengthHandleT.....	356
4.2.1.2.2	PrepareScriptInProgress	356
4.2.1.2.3	PrepareScriptVerifyScript.....	356
4.2.1.2.4	PrepareScriptHashBody	356
4.2.1.2.5	PrepareScriptHashSignature	356
4.2.1.2.6	PrepareScriptGeneratePassword	357
4.2.1.3	Server Behavior of the IDL_DSAPrepareScript Method	357
4.2.2	IDL_DSASExecuteScript (Opnum 1).....	358
4.2.2.1	Method-Specific Concrete Types.....	359
4.2.2.1.1	DSA_MSG_EXECUTE_SCRIPT_REQ	359
4.2.2.1.2	DSA_MSG_EXECUTE_SCRIPT_REQ_V1	359
4.2.2.1.3	DSA_MSG_EXECUTE_SCRIPT_REPLY.....	360
4.2.2.1.4	DSA_MSG_EXECUTE_SCRIPT_REPLY_V1	360
4.2.2.2	Method-Specific Abstract Types and Procedures	360
4.2.2.2.1	ExecuteScriptInProgress.....	360
4.2.2.2.2	ExecuteScript.....	360
4.2.2.3	Server Behavior of the IDL_DSASExecuteScript Method	360
5	Common Data Types, Variables, and Procedures	362
5.1	AbstractPTFromConcretePT	362
5.2	AccessCheckAttr	362

5.3	AccessCheckCAR	363
5.4	AccessCheckObject.....	363
5.5	AccessCheckWriteToSpnAttribute	363
5.6	AmIRODC	364
5.7	AmILHServer.....	365
5.8	ATTR	365
5.9	ATTRBLOCK.....	365
5.10	AttributeStamp	365
5.11	AttributeSyntax	366
5.12	AttrStamp	366
5.13	ATTRTYP	366
5.14	AttrtypFromSchemaObj	366
5.15	ATTRVAL.....	367
5.15.1	Concrete Value Representations	367
5.15.1.1	INT32.....	367
5.15.1.2	INT64.....	368
5.15.1.3	OctetString	368
5.15.1.4	String8	368
5.15.1.5	String16	368
5.15.1.6	SECURITY_DESCRIPTOR	368
5.15.1.7	SID	368
5.15.2	Abstract Value Representations	368
5.15.2.1	Object(DS-DN)	369
5.15.2.2	Object(DN-String).....	370
5.15.2.3	Object(DN-Binary)	370
5.15.2.4	Object(Access-Point)	370
5.15.2.5	Object(OR-Name)	370
5.15.2.6	String(NT-Sec-Desc)	371
5.15.2.7	String(Sid)	371
5.15.2.8	String(Teletex)	371
5.15.3	Converting Between Abstract and Concrete Value Representations	371
5.15.3.1	Boolean	373
5.15.3.2	Enumeration and Integer	373
5.15.3.3	LargeInteger	374
5.15.3.4	Object(Presentation-Address).....	374
5.15.3.5	Object(Replica-Link) String (Octet)	374
5.15.3.6	String(IA5) String(Printable) String(Numeric) String(Teletex).....	374
5.15.3.7	String(Unicode)	375
5.15.3.8	String(Object-Identifier)	375
5.15.3.9	String(UTC-Time) and String(Generalized-Time)	375
5.15.3.10	Object(DS-DN)	375
5.15.3.11	Object(DN-Binary).....	377
5.15.3.12	Object(DN-String)	378
5.15.3.13	Object(OR-Name)	379
5.15.3.14	Object(Access-Point)	379
5.15.3.15	String(Sid)	379
5.15.3.16	String(NT-Sec-Desc).....	379
5.15.4	ATTRTYP-to-OID Conversion	381
5.16	ATTRVALBLOCK	386
5.17	ATTRVALFromValue.....	386
5.18	BOOL.....	386
5.19	BYTE	387
5.20	CHANGE_LOG_ENTRIES.....	387
5.21	CHANGELOG_ENTRY.....	387
5.22	CheckGroupMembership	387

5.23	ClientAuthorizationInfo	387
5.24	ClientExtensions	388
5.25	ConcretePTFromAbstractPT	388
5.26	ConfigNC.....	388
5.27	dc, DC	388
5.28	DefaultNC	389
5.29	DefaultNCOfDC	389
5.30	DescendantObject	389
5.31	DN	390
5.32	DNBinary	390
5.33	DomainNameFromNT4AccountName.....	390
5.34	DRS_EXTENSIONS	390
5.35	DRS_EXTENSIONS_INT.....	390
5.36	DRS_HANDLE	394
5.37	DRS_OPTIONS.....	394
5.38	DRS_SecBuffer	395
5.39	DRS_SecBufferDesc.....	397
5.40	DRS_SPN_CLASS	397
5.41	DS_REPL_OP_TYPE	397
5.42	DSABObj.....	397
5.43	DSName	398
5.44	DSNAME	398
5.44.1	DSNAME Equality.....	400
5.45	DSTIME	400
5.46	DWORD	401
5.47	ENTINF	401
5.48	ENTINF_GetValue.....	401
5.49	ENTINF_SetValue	402
5.50	ENTINF_EnumerateAttributes	402
5.51	ENTINFLIST.....	402
5.52	Expunge	403
5.53	FILETIME	403
5.54	FilteredGCPAS	403
5.55	FilteredPAS	404
5.56	FindChar	404
5.57	FindCharRev	405
5.58	FOREST_TRUST_INFORMATION	405
5.58.1	Record	406
5.58.2	Determining If a Name Is In a Trusted Forest	409
5.59	FOREST_TRUST_RECORD_TYPE	414
5.60	ForestRootDomainNC.....	414
5.61	FullReplicaExists	414
5.62	GCPAS	415
5.63	GetFilteredAttributeSet	415
5.64	GetNCType	416
5.65	GetAttrVals	417
5.66	GetCallerAuthzInfo	417
5.67	GetDefaultObjectCategory	417
5.68	GetDSNameFromDN	417
5.69	GetForestFunctionalLevel	418
5.70	GetFSMORoleOwner	418
5.71	GetInstanceNameFromSPN.....	418
5.72	GetObjectNC	418
5.73	GetProxyEpoch	419
5.74	GetProxyType	419

5.75	GetServiceClassFromSPN	419
5.76	GetServiceNameFromSPN	419
5.77	groupType Bit Flags.....	419
5.78	GUID	420
5.79	GuidFromString	420
5.80	GuidToString	420
5.81	handle_t	420
5.82	instanceType Bit Flags	420
5.83	Is2PartSPN	421
5.84	Is3PartSPN	421
5.85	IsBuiltinPrincipal	421
5.86	IsDomainNameInTrustedForest.....	421
5.87	IsDCAccount.....	422
5.88	IsGC.....	422
5.89	IsGetNCChangesPermissionGranted.....	422
5.90	IsGUIDBasedDNSName	423
5.91	IsMemberOfBuiltinAdminGroup	423
5.92	IsRevealFilteredAttribute.....	423
5.93	IsRevealSecretRequest	424
5.94	IsValidServiceName.....	425
5.95	KCCFailedConnections	425
5.96	KCCFailedLinks	425
5.97	LARGE_INTEGER.....	426
5.98	LDAP_CONN_PROPERTIES.....	426
5.99	LDAPConnections	426
5.100	LinkStamp	427
5.101	LinkValueStamp	427
5.102	LocalAttidFromRemoteAttid	428
5.103	LONG.....	428
5.104	LONGLONG.....	428
5.105	LPWSTR	428
5.106	MakeAttid.....	428
5.107	MakeProxyValue.....	428
5.108	MasterReplicaExists	429
5.109	MD5_CTX	429
5.110	MD5Final	429
5.111	MD5Init.....	429
5.112	MD5Update	429
5.113	MergeUTD	429
5.114	MTX_ADDR.....	430
5.115	NCType Bits.....	430
5.116	NetworkAddress	431
5.117	NewPrefixTable	431
5.118	Nt4ReplicationState	431
5.119	NT4SID	431
5.120	NTDSTRANSPORT_OPT Values	432
5.121	NULLGUID	432
5.122	ObjExists.....	432
5.123	OID	432
5.124	OID_t	432
5.125	OidFromAttid	432
5.126	parent.....	432
5.127	PARTIAL_ATTR_VECTOR_V1_EXT	433
5.128	partialAttributeSet	433
5.129	PartialGCReplicaExists.....	433

5.130	PAS_DATA.....	433
5.131	PdcChangeLog	434
5.132	PerformAddOperation	434
5.133	PerformAddOperationAsSystem.....	435
5.134	PerformModifyOperation.....	435
5.135	PrefixTable	435
5.136	PrefixTableEntry.....	435
5.137	PROPERTY_META_DATA_EXT	436
5.138	PROPERTY_META_DATA_EXT_VECTOR.....	436
5.139	proxiedObjectName Value Format	436
5.140	RDN	437
5.141	rdnType	437
5.142	RemoveObj	437
5.143	REPLENTINFLIST	437
5.144	ReplicationQueue	438
5.145	REPLTIMES	438
5.146	replUpToDateVector, ReplUpToDateVector	439
5.147	REPLVALINF.....	439
5.148	REPS_FROM.....	440
5.149	REPS_TO.....	443
5.150	repsFrom, RepsFrom.....	445
5.151	repsTo, RepsTo	447
5.152	Rid	448
5.153	Right	448
5.154	RIGHT Values	448
5.155	RPCClientContexts.....	448
5.156	RPCOutgoingContexts	448
5.157	sAMAccountType Values	449
5.158	SCHEMA_PREFIX_TABLE	449
5.159	SchemaNC	450
5.160	SchemaObj.....	450
5.161	ServerExtensions	450
5.162	SID	450
5.163	SidFromStringSid	450
5.164	StampLessThanOrEqualUTD.....	450
5.165	StartsWith.....	451
5.166	STATUS Codes	451
5.167	StringSidFromSid	451
5.168	SubString.....	451
5.169	Syntax	452
5.170	SYNTAX_ADDRESS	452
5.171	SYNTAX_DISTNAME_BINARY	452
5.172	systemFlags Values	454
5.173	UCHAR.....	454
5.174	ULARGE_INTEGER	454
5.175	ULONG.....	455
5.176	ULONGLONG.....	455
5.177	UNICODE_STRING.....	455
5.178	UPTODATE_CURSOR_V1	455
5.179	UPTODATE_CURSOR_V2	455
5.180	UPTODATE_VECTOR_V1_EXT	456
5.181	UPTODATE_VECTOR_V2_EXT	456
5.182	userAccountControl Bits	456
5.183	UserNameFromNT4AccountName	457
5.184	USHORT.....	457

5.185	USN.....	457
5.186	USN_VECTOR	458
5.187	UUID	458
5.188	Value.....	458
5.189	VALUE_META_DATA_EXT_V1	458
5.190	ValueFromATTRVAL	458
5.191	WCHAR	459
6	Security	460
6.1	Security Considerations for Implementers	460
6.2	Index of Security Parameters	460
7	Appendix A: Full IDL	461
8	Appendix B: Windows Behavior	489
9	Index.....	493

1 Introduction

This document specifies the Directory Replication Service (DRS) Remote Protocol, which is an RPC protocol for replication and management of data in Active Directory (AD).

The protocol consists of two RPC interfaces named [drsuapi](#) and [dsaop](#). The name of each drsuapi method begins with "IDL_DRS", while the name of each **dsaop** method begins with "IDL_DSA". This protocol was originally implemented in Windows 2000 Server and is implemented in all subsequent server releases. It is not implemented in Windows 3.1, Windows NT 3.51, Windows NT 4.0, Windows XP, or Windows Vista.

Some functionality exposed by these RPC protocols is also available using the **Lightweight Directory Access Protocol (LDAP)** protocol ([\[MS-ADTS\]](#) section 3.1.1.3); the overlap is described in section [1.4](#).

The special typographical conventions used in this document are described in section [3.2](#).

State is included in the state model for this specification only as necessitated by the requirement that a licensee's implementation of Windows server protocols must be capable of receiving messages and responding 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, all behaviors are required elements of the protocol. Any specified behavior not explicitly qualified with MAY or SHOULD is to be treated as if it were specified as a MUST behavior.

1.1 Glossary

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

1.1.1 Pervasive Concepts

This specification uses [KNUTH1] section 2.3.4.2 as a reference for the graph-related 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.

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 term "replica" suggests, two replicas often 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 (that is, **replication**) to converge the values of the remaining attributes.

When members of a set of replicas are considered to be the same, it is common to say "an object" as shorthand referring to the set of corresponding objects in the replicas.

Object Class: A set of restrictions on the construction and update of objects. An object class must be specified when an object is created. 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.

Parent Object: See **Object**.

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

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

1.1.2 Glossary Entries

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

Ancestor Object
AttributeStamp
Authentication
Authentication Level
Back Link Value
Binary Large Object (BLOB)
Constructed Attribute
Container
Control Access Right (CAR)
Digest
Directory
Discretionary Access Control List (DACL)
Domain Controller (DC)
Dynamic Endpoint
Expunge
Forest
Forward Link Value
FSMO Role Owner
Full NC Replica
Garbage Collection
Global Catalog (GC)
Global Catalog Server (GC Server)
Interface Definition Language (IDL)
LDAP Connection
Lightweight Directory Access Protocol (LDAP)
Link Attribute
Link Value
LinkValueStamp
Local Domain Controller (Local DC)

Microsoft Interface Definition Language (MIDL)
Network Data Representation (NDR)
Non-Replicated Attribute
NULL GUID
Opnum
Originating Update
Partial Attribute Set (PAS)
PDC Role Owner
Prefix Table
Remote Procedure Call (RPC)
Replication Latency
RPC Transport
Schema
Security Principal
Security Provider

The following terms are specific to this document:

Abstract Type: A type used in this specification whose representation need not be standardized for interoperability because the type's use is internal to the specification. See **Concrete Type**.

Access Control Entry (ACE): An entry in an **access control list (ACL)**. An ACE contains a set of access rights and a **security identifier (SID)** that identifies the **principal** (including **group principals**) for whom the rights are allowed, denied, or audited.

Access Control List (ACL): A sequence of **access control entries (ACEs)** that describes the rules for authorizing access to some resource; for example, an **object** or set of **objects**.

Active Directory (AD): 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 Windows 2000 and is available as part of Windows 2000 Server 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 3.1, Windows NT 3.51, Windows NT 4.0, or Windows XP. For more information, see [\[MS-SECO\]](#) section 2.2.2 and [\[MS-ADTS\]](#).

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".)

Application NC: A specific type of **naming context (NC)**. An **application NC** does not contain **security principal objects** and does not appear in the **GC**. The root of an **application NC** is an **object of class domainDNS**. See **domainDNS**.

Attribute: (Note: This definition is a specialization of the **Attribute** entry in [Pervasive Concepts](#).) An identifier for a single- or multivalued data element 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**. **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**.

Back Link Attribute: A computed **attribute** whose values include **object** references (for example, an **attribute** of syntax Object(DS-DN)). The 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 **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 [MS-ADTS] section [3.1.1.1.6](#).

Binary OID: An **OID** in a BER-encoded binary format, as specified in [\[ITU690\]](#) section 8.19.

Built-in Domain: The **SID** namespace that is defined by the fixed **SID** S-1-5-32. The built-in domain contains **groups** that define roles on a local computer, such as "Backup Operators".

Built-in Domain SID: The fixed **SID** S-1-5-32 of the built-in domain.

Built-in Principal: A **security principal** within the **built-in domain** whose **SID** is identical in every **domain**.

Canonical Name: A syntactic transformation of an Active Directory **distinguished name (DN)** into something resembling a path 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/". See **domainDNS**.

Checksum: A value that is the summation of a byte stream. By comparing the checksums computed from a data item at two different times, it can be quickly assessed whether the data items are different.

Class: See **Object Class**.

Compression Chunk (or Chunk): When compression is used for **replication** data, the data is divided into smaller units, called "compression chunks", that are suitable for the particular algorithm. The chunk size is specific to the compression algorithm being employed.

Computer Object, [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**.

Concrete Type: A type used in this specification whose representation must be standardized for interoperability. Specific cases include types in the IDL definition of an RPC interface, types sent over RPC but whose representation is unknown to RPC, and types stored as byte strings in directory attributes.

Configuration Naming Context (Config NC): A specific type of **NC**, containing configuration information. A **forest** has a single **config NC**, which is 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**. A [crossRef](#) describes the properties of an **NC**, such as its DNS name, operational settings, and so on.

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**.

Cyclic Redundancy Check (CRC): A broad class of functions used for detecting errors in the transmission of data; for example, **CRC** is one method of generating a **checksum**.

Default Naming Context (Default NC): Part of the state of a **DC**. A **DC's default NC** is the **NC** of its **default NC replica**. A **DC's default NC** contains the **DC's** [computer](#) **object**.

Default Naming Context Replica (Default NC Replica): Part of the state of a **DC**. A **DC's default NC replica** is a **domain NC**. On a **read-only DC**, the **default NC replica** is a **filtered partial NC replica**; otherwise, it is a **full NC replica**, hosted by the DC.

Directory Object (or Object): An **Active Directory (AD) object**, which is a specialization of **object** as defined in the Pervasive Concepts section of this glossary. The identifying **attribute** is [objectGUID](#), and the **parent-identifying attribute** (not exposed as an **LDAP attribute**) is [parent](#). **Active Directory objects** are similar to **LDAP** entries, as defined in [\[RFC2251\]](#); the differences are specified in [MS-ADTS] section [3.1.1.3.1](#).

Distinguished Name (DN): A human-readable name for an **object**; every **object** has a **DN**. **Active Directory DNs** are **LDAP DNs** [\[RFC2251\]](#), restricted as specified in [MS-ADTS] section [3.1.1.3.1.2.1](#). The **DN** of an **object** is the **object's RDN** followed by ",", followed by the **DN** of its **parent**; for example: "cn=Peter Houston, ou=NTDEV, dc=microsoft, dc=com". See **Canonical Name**.

Domain: A unit of security administration and delegation in a Microsoft Windows network. For more information, see [MS-SECO] section [2.2](#), and [MS-ADTS].

Domain Naming Context (Domain NC): A type of **NC** that represents a **domain**. A **domain NC** can contain **security principal objects**; no other type of **NC** can contain **security principal objects**. **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](#). See **domainDNS**.

Domain Security Identifier (Domain SID): The **SID** of the root object of a **domain NC**. The **RID** portion of the **domain SID** is always zero. Every **security principal object** in a **domain NC** has an [objectSid](#) attribute equal to the **domain SID** except for the **RID** portion.

domainDNS: A specific **object class**. The root of a **domain NC** or an **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 by using DNS.

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

DSA Object: An **object of class** [nTDSDSA](#), always located in the **config NC**. This object represents a **DC** in the **forest**.

DSName: A DSName is 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). Given a DSName, an object can be identified within a set of **NC replicas** according to the matching rules defined in section [5.43](#).

Dynamic Object: An object with a "time-to-die" attribute, [msDS-Entry-Time-To-Die](#). **Active Directory** "garbage-collects" a dynamic object immediately after the time-to-die of the object has passed. The **constructed attribute** [entryTTL](#) gives a dynamic object's current "time-to-live"; that is, the difference between the current time and [msDS-Entry-Time-To-Die](#). See [\[RFC2589\]](#).

Encryption Key: One of the input parameters to an encryption algorithm. An encryption algorithm takes as input a clear-text message and a key, and results in a cipher-text message. The corresponding decryption algorithm takes a cipher-text message, and the key, and results in the original clear-text message.

Endpoint: A network-specific address of a server process for remote procedure calls. The actual name of the endpoint depends on the **RPC protocol sequence** being used. For example, for the NCACN_IP_TCP **RPC protocol sequence**, an endpoint might be TCP port 1025. For more information, see [\[C706\]](#).

Entry: A term often used as a synonym for object, but not in this document.

Extended Canonical Name: Same as a **canonical name**, except that the rightmost forward slash ('/') is replaced with a newline character.

Extended Operation: A special **replication cycle** in which a client **DC** requests an action on a FSMO role; for example, a change in the FSMO role owner. FSMO role abandon and **FSMO role transfer** are examples of extended operations.

Filtered Attribute Set (FAS): The subset of attributes that are not replicated to **filtered partial NC replicas** and **filtered GC partial NC replicas**. A particular **filtered attribute set** is part of the state of the forest, and is used to control the attributes that replicate to a **read-only DC**. 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** it contains. The subset of 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; that is, it does not accept **originating updates**.

Filtered 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 all the attributes of the **objects**, excluding those attributes in the **filtered attribute set**. A **filtered partial NC replica** is not writable; that is, it does not accept **originating updates**.

Flexible Single Master Operation: See **FSMO**.

Forest Root Domain NC: The **domain NC** within a forest that is the parent of the **config NC**. The DNS name of the forest root domain NC serves as the **forest's** name.

Forward Link Attribute: A specific type of attribute. The values of a forward link attribute include object references (for example, syntax Object(DS-DN)). The **forward link values** can be used to compute the values of a related attribute, that is a **back link attribute**, on other objects. A forward link attribute can exist with no corresponding **back link attribute**, but not vice versa. See [MS-ADTS] section 3.1.1.1.6.

FSMO (Flexible Single Master Operation): 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**.

FSMO Role: A set of objects that can be updated in only one **NC replica** at any given time. For more information, see [MS-ADTS] section [3.1.1.1.11](#). See **FSMO Role Owner**.

FSMO Role Abandon: A request to a **DC D**. The effect is for *D* to request the current owner of a specified FSMO role to transfer the role to *D* (see **FSMO Role Transfer**). Abandon is typically initiated by the current role owner in anticipation of being unable to host the role; for example, because the **DC** is being decommissioned.

FSMO Role Object: The object in the **domain** or forest that represents a specific **FSMO role**. This object is a member 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 that the client can try the request again on another **DC**.

GC Partial Attribute Set (PAS): The subset of attributes that replicate to **GC partial NC replicas**. A particular **GC 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**. A **GC partial NC replica** is not writable, for example, it does not accept **originating updates**.

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 GROUP_TYPE_ACCOUNT_GROUP is present in *G*'s [groupType](#) attribute. A global group contributes to the creation of **security contexts** if GROUP_TYPE_SECURITY_ENABLED is present in *G*'s [groupType](#) attribute; in this case the **group** is valid for inclusion within access control lists (ACLs) anywhere in the forest.

Globally Unique Identifier (GUID): A 128-bit value used in cross-process communication to identify entities such as client and server interfaces and RPC objects. For more information, see [\[C706\]](#). A string representation of GUIDs, commonly called the "dashed-string" representation, is specified in [\[RFC4122\]](#) section 3.

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

Group: See **Group Object**.

Group Object, [group](#) Object: An **object of class [group](#)**, representing a set of objects. 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](#)). Representation of group subsets is called "nested group membership". 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 example, used to represent e-mail distribution lists).

Group Principal: A **group** representing a collection of **security principals**. A **group principal** can be used in an **ACE** to collectively grant or deny permissions to all the **security principals** in that **group**.

Invocation ID: A unique identifier for a function that maps from **update sequence numbers (USNs)** to updates to the **NC replicas** of a DC.

Knowledge Consistency Checker (KCC): An internal Windows component of **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.

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

Lost and Found Container: A **container** holding objects in a given **NC** that do not have parent objects due to add and delete operations that originated on different DCs. The **container** is a child of the **NC** root and has **RDN** `cn=LostAndFound` in **domain NCs** and `cn=LostAndFoundConfig` in **config NCs**.

MD5 Hash: A hashing algorithm developed by RSA Data Security, Inc., and defined in [\[RFC1321\]](#).

Mixed Mode: A state of an **Active Directory domain** that supports DCs running Windows NT Server 4.0. Mixed mode does not allow organizations to take advantage of new **Active Directory** features such as **universal groups**, nested group membership, and inter-domain group membership. See **Native Mode**.

MSZIP Compression Algorithm: A compression algorithm specified in [\[RFC1591\]](#) that is used between Windows 2000 DCs.

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; the **SID** is present if and only if the NC is a **domain NC**. **Active Directory** allows NCs to be organized into a tree structure.

Native Mode: A state of an **Active Directory domain** in which all current and future DCs run Windows 2000 Server or higher; no DCs run Windows NT Server 4.0. Native mode allows organizations to take advantage of new **Active Directory** features such as **universal groups**, nested group membership, and inter-domain group membership. See **Mixed Mode**.

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

[nTDSDSA Object](#): See **DSA Object**.

Object Class Name: The [LDAPDisplayName](#) of the [classSchema](#) object of a class. This document consistently uses **object class names** to denote classes; for example, [user](#) and [group](#) are both **object class names**. The correspondence between LDAP display names and numeric

OIDs in the **Active Directory schema** is specified in the following appendices of [MS-ADTS]: [\[MS-ADSC\]](#), [\[MS-ADA1\]](#), [\[MS-ADA2\]](#), and [\[MS-ADA3\]](#).

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

Object of Class x (or x Object): An object *O* such that one of the values of its **objectClass** attributes is *x*. For example, if *O*'s **objectClass** contains the value [user](#), *O* is an object of class [user](#). This is often contracted to "[user](#) object".

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

objectClass: The attribute on an object that holds the **object class name** of each **object class** of the object.

objectGUID: The identifying attribute on an object, in the sense of the Pervasive Concepts definition of object. The value of an object's **objectGUID** is a GUID assigned when the object was created and is immutable thereafter. The integrity of 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 an object's **objectSid** is assigned when the **security principal object** was 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.

Partial NC Replica: An **NC replica** that contains a **schema**-specified subset of **attributes** for the objects it contains. A **partial NC 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**.

PDC Emulator: A **DC** that is designated to track changes made to the accounts of all computers in a **domain**. The PDC emulator 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 emulator**.

Primary Domain Controller (PDC): See **PDC Emulator**.

Principal: See **Security Principal**.

Read Permission: Authorization to read an **attribute** of an object. For more information, see [MS-ADTS] section [5.1.3](#).

Read-Only Domain Controller (RODC or Read-Only DC): An AD/DS **DC** that performs no 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".

Relative Identifier (RID): The last item in the series of sub-authority values in a **SID** (see [MS-DTYP] section [2.4.2](#)). Differences in the RID are what distinguish the different **SIDs** generated within a **domain**.

Replicated Attribute: An **attribute** whose values are replicated. See **Replication**.

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

Replication: The process of propagating the effects of all originating writes, to any replica of an NC, to all replicas of the NC. If originating writes cease and replication continues, all replicas converge to a common application-visible state.

Replication Cycle: See **Cycle**.

Replication Epoch: A state variable of a **DC** that changes when a **DC** is no longer compatible for replication with its former partners. A server receiving a replication request tests the client's replication epoch against its own, and refuses the request if the two are not equal.

RID Allocation Pool: The set of RIDs that a **domain NC** replica can assign to new objects having the [objectSid](#) **attribute** without obtaining more RIDs from the **domain NC's RID available pool**. See **Relative Identifier (RID)**, **objectSid**.

RID Available Pool: The set of RIDs for a domain NC that have not been assigned to the RID allocation pool of any replica of the NC. The RID available pool is represented by the values of attributes within the **domain NC's** RID Master FSMO role.

Root Domain: See **Forest Root Domain NC**.

RPC Protocol Sequence: A character string that represents a valid combination of an RPC protocol, a network layer protocol, and a transport layer protocol. For example, the protocol sequence NCACN_IP_TCP describes a Network Computing Architecture (NCA) connection over the Internet Protocol (IP) with a Transmission Control Protocol (TCP) as transport. For more information, see [\[C706\]](#) and [MS-RPCE] section [2.1](#).

RPC Session Key: See **Session Key**.

Salt: An additional random quantity, specified as input to an encryption function, that is used to increase the strength of the encryption.

Schema Naming Context (Schema NC): A specific type of NC that contains schema objects representing the schema. 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.

Secret Data: An implementation-specific set of attributes on **objects of class** [user](#) that contain security-sensitive information about the security principal.

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: A data structure containing the security information associated with a securable entity, such as an 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 Identifier (SID): An identifier for a security principal object. The SID is composed of an account authority portion (typically corresponding to a **domain**) and an 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 a SID is specified in [\[MS-SECO\]](#) section **2.1.2**. See **Relative Identifier (RID)**.

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](#) classes are examples of security principal object classes (though not every [group](#) object is a security principal object). See **domainDNS**, **objectSid**, **Computer Object**, **Group Object**, **User Object**.

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

Service Account Object: The security principal object that corresponds to the principal running a service. For a typical service (including some configurations of an AD/LDS **DC**), this is a [user](#) object; for a service running as Local System or Network Service (including all AD/DS **DCs** and the default configuration of an AD/LDS **DC**), this is the [computer](#) object of the computer.

Service Principal Name (SPN): The name a client uses to identify a service for mutual **authentication**. (For more information, see [\[RFC1964\]](#) section 2.1.1.) An SPN consists of either two parts or three parts, each separated by a forward slash ('/'). The first part is the *service class name*, the second part is the *instance name*, and the third part (if present) is the *service name*. For example, "ldap/dc-01.fabrikam.com/fabrikam.com" is a three-part SPN where "ldap" is the service class name, "dc-01.fabrikam.com" is the instance name, and "fabrikam.com" is the service name.

Session Key: A cryptographic key negotiated by the client and the server side based on a shared secret.

SHA1 Hash: A hashing algorithm defined in [\[FIPS180\]](#) that was developed by the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA).

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 a user logs in, an **Active Directory** client finds a **DC** that is in the same site as the client, or near the same site if there is no **DC** in the site. See **Knowledge Consistency Checker (KCC)**, **Site Object**.

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

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

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 actual data values.

STATUS Code: A 32-bit quantity where zero represents success and nonzero represents failure. The specific failure codes used in this specification are documented in section 5, [STATUS codes](#).

Tombstone: A deleted object in the directory that remains in storage until a configured amount of time (the **tombstone lifetime**) has passed, after which the object is permanently deleted. By keeping the tombstone in existence for the **tombstone lifetime**, the deleted state of the object has time to replicate to all DCs.

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

Unicode: An industry standard representation for text and symbols from the world's writing systems. UTF-16 is a 16-bit, variable-width encoding of Unicode; UTF-8 is an 8-bit, variable-width encoding.

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 GROUP_TYPE_UNIVERSAL_GROUP is present in *G*'s [groupType attribute](#). A universal group contributes to the creation of security contexts if GROUP_TYPE_SECURITY_ENABLED is present in *G*'s [groupType attribute](#); in this case, the group is valid for inclusion within ACLs anywhere in the forest.

Universally Unique Identifier (UUID): See **GUID**.

Up-To-Date Vector: The representation of an assertion about the presence of originating updates from different sources in an NC replica. See **Replication Cycle, Update Sequence Number (USN)**.

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. For more information, see [MS-ADTS] section [3.1.1.1.9](#). See **Invocation ID**.

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

Well-Known Endpoint: A network-specific address that is known between client and server instances. See also **Endpoint**. For more information, see [\[C706\]](#).

Windows Error Code: A 32-bit quantity where zero represents success and nonzero represents failure. Specific failure codes are documented in [\[MS-ERREF\]](#).

Writable NC Replica: An NC replica that accepts originating updates. See **Full NC Replica, Partial NC Replica**.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified 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. 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.

[C311] The Open Group, "DCE 1.1: Authentication and Security Services--Document Number C311", October 1997, <http://www.opengroup.org/onlinepubs/9668899/>

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

[FIPS180] Federal Information Processing Standards Publication, "Secure Hash Standard", FIPS PUB 180-1, April 1995, <http://www.itl.nist.gov/fipspubs/fip180-1.htm>

[ISO/IEC 13239] International Organization for Standardization, "Information technology -- Telecommunications and information exchange between systems -- High-level data link control (HDLC) procedures", <http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=37010&ICS1=35&ICS2=100&ICS3=20&showrevision=y&scopelist=CATALOGUE>, ISO/IEC 13239:2002, July 2007.

Note There is a charge to download the specification.

[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-ADTS] Microsoft Corporation, "[Active Directory Technical 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-NRPC] Microsoft Corporation, "[Netlogon Remote Protocol Specification](#)", July 2006.

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

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

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

[RC4] The RC4 Encryption Algorithm. RSA Data Security, Inc.,
<http://www.rsa.com/node.aspx?id=1204>

Note To obtain this stream cipher that is licensed by RSA Data Security, you need to contact this company.

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

[RFC1321] Rivest, R. "The MD5 Message-Digest Algorithm", RFC 1321, April 1992,
<http://www.ietf.org/rfc/rfc1321.txt>

[RFC1327] Hardcastle-Kille S., "Mapping Between X.400(1988) / ISO 10021 and RFC 822", RFC 1327, May 1992, <http://www.ietf.org/rfc/rfc1327.txt>

[RFC1591] Postel, J., "Domain Name System Structure and Delegation", RFC 1591, March 1994,
<http://www.ietf.org/rfc/rfc1591.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>

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996, <http://www.ietf.org/rfc/rfc1951.txt>

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

[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>

[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>

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001,
<http://www.ietf.org/rfc/rfc2821.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>

1.2.2 Informative References

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

[NELSON] Nelson, G., "Systems Programming with Modula-3", Englewood Cliffs, NJ: Prentice Hall, 1991, ISBN: 0135904641.

1.3 Protocol Overview (Synopsis)

This document specifies the Directory Replication Service Remote Protocol, an RPC protocol for replication between domain controllers and management of Active Directory. The protocol consists of two RPC interfaces, named [drsuapi](#) and [dsaop](#). The name of each **drsuapi** method begins with "IDL_DRS", while the name of each **dsaop** method begins with "IDL_DSA".

1.3.1 Methods Categorized by Function

The Directory Replication Service (DRS) Remote Protocol contains 28 methods. These methods are diverse in function and fall into the following categories:

- Context handle methods: [IDL_DRSBind](#), [IDL_DRSUnbind](#). These methods create and destroy RPC context handles that maintain volatile state used by [drsuapi](#) methods. The [dsaop](#) methods do not use context handles.
- Replication methods: [IDL_DRSGetNCChanges](#), [IDL_DRSReplicaSync](#), [IDL_DRSReplicaVerifyObjects](#), [IDL_DRSGetReplInfo](#). The **IDL_DRSGetNCChanges** method replicates directory changes from the server to the client. The **IDL_DRSReplicaSync** and **IDL_DRSReplicaVerifyObjects** methods cause the server to call **IDL_DRSGetNCChanges** on the client. The **IDL_DRSGetReplInfo** method is used to gather information about the replication state of the server.
- Cross-domain move method: [IDL_DRSInterDomainMove](#). This method is used in the server implementation of LDAP Modify DN when the DN modification moves an object from one domain NC to another.
- Lookups: [IDL_DRSVerifyNames](#), [IDL_DRSCrackNames](#), [IDL_DRSGetMemberships](#), [IDL_DRSGetMemberships2](#). These methods perform specialized directory lookups. They are all used by a DC client; the **IDL_DRSCrackNames** method is commonly used by a non-DC client.
- DC Locator support methods: [IDL_DRSDomainControllerInfo](#), [IDL_DRSQuerySitesByCost](#). These methods retrieve information about the domain controllers in a domain or forest and information about the cost of connections between different sites.
- Windows NT 4.0 Replication support method: [IDL_DRSGetNT4ChangeLog](#). This method is used in the implementation of Active Directory support for replication to Windows NT 4.0 backup domain controllers (BDCs), specifically in the implementation of moving the PDC Emulator FSMO role from one DC to another without triggering a full sync of Windows NT 4.0 BDCs (see [\[MS-NRPC\]](#) section 3.5).
- Knowledge Consistency Checker support methods: [IDL_DRSUpdateRefs](#), [IDL_DRSReplicaAdd](#), [IDL_DRSReplicaDel](#), [IDL_DRSReplicaModify](#), [IDL_DRSExecuteKCC](#). These methods are used by the Knowledge Consistency Checker ([\[MS-ADTS\]](#) section 7.2) and by administrator tools to manage replication topology.
- Administrator-tool support methods: [IDL_DRSAddEntry](#), [IDL_DRSAddSidHistory](#), [IDL_DRSRemoveDsServer](#), [IDL_DRSRemoveDsDomain](#), [IDL_DRSGetObjectExistence](#),

[IDL_DSAPrepareScript](#), [IDL_DSAExecuteScript](#), [IDL_DRSSWriteSPN](#). These methods are used by administrator tools to perform various specialized functions.

The specification of each method in section 4, [RPC Methods and Their Behavior](#), includes an *Informative summary of behavior* that provides a detailed introduction to the method.

1.3.2 Sequencing Issues

The sequencing issues in this RPC protocol are as follows:

- For server and client initialization, see section [3.6](#).
- The [drsuapi RPC interface](#) is a "context handle"-based RPC interface; [\[C706\]](#) specifies what this means. A client obtains a [DRS_HANDLE](#) for a particular DC by calling [IDL_DRSSBind](#), then calls any other **drsuapi** method on that DC, passing the **DRS_HANDLE** as the first parameter. The client's **DRS_HANDLE** remains valid for making method calls until the client calls [IDL_DRSSUnbind](#), or until the server unilaterally invalidates the **DRS_HANDLE** (for example, by crashing).

The only state associated with a **DRS_HANDLE** is the state established by **IDL_DRSSBind**. This state is immutable for as long as the **DRS_HANDLE** remains valid. Therefore if a client creates two binding handles to the same DC by using the same parameters to **IDL_DRSSBind**, the server behavior of a **drsuapi** method is not affected by the client's choice of binding handle passed to the method.

Because the state associated with a **DRS_HANDLE** is immutable so long as the **DRS_HANDLE** remains valid, there are no special considerations involved in making concurrent method calls using the same **DRS_HANDLE**; the client is free to make concurrent method calls.

- Two methods use the "cookie" design pattern. In this pattern, the client sends an initial request containing a designated null value for a certain parameter. The server response contains a value that is opaque to the client, or contains the designated null value. If the value is not null and the response indicates that another client request is required to complete some higher-level operation, the client sends the opaque value returned by the server in the next request rather than sending the designated null value. The exchange of requests and responses continues until some response indicates that the higher-level operation is complete.

The two methods that follow this pattern are:

- [IDL_DRSSGetNCChanges](#): In this instance of the "cookie" pattern, the server returns a "cookie" in the response that completes the higher-level operation. The client may use this cookie at the start of the next higher-level operation. The higher-level operation is a complete replication cycle that improves the client's up-to-date vector. See section [4.1.10.1](#) for an explanation of replication cycles.
- [IDL_DRSSGetNT4ChangeLog](#): In this instance of the "cookie" pattern, the server returns a "cookie" in the response that completes the higher-level operation. The client does not use this cookie at the start of the next higher-level operation. The client supplies the designated null value at the start of the next higher-level operation. The higher-level operation is the retrieval of a complete Windows NT 4.0 change log. See the informative summary of this method in section [4.1.11.3](#).
- Successfully processing an [IDL_DSAPrepareScript](#) request generates a password and stores that password locally on the server. The server returns this password in the **IDL_DSAPrepareScript** response. When a server is in this state (that is, when it holds the

password created by **IDL_DSAPrepareScript**), it processes an [IDL_DSAExecuteScript](#) request that includes this password, otherwise it rejects the request.

- [IDL_DRSInitDemotion](#) is called before the other demotion methods: [IDL_DRSReplicaDemotion](#) and [IDL_DRSFinishDemotion](#).
- Otherwise, all method requests are independent, apart from their dependencies on the state of the directory. The potential dependencies are varied and understanding them requires understanding the state model specified in [\[MS-ADTS\]](#) section 3.1.1. Here are some examples:
 - Successfully processing an [IDL_DRSAddEntry](#) request can create a [crossRef](#) object. When the directory is in this state (that is, when it holds the [crossRef](#) object), an [IDL_DRSRemoveDsDomain](#) request can successfully remove that [crossRef](#) object (subject to other conditions specified with **IDL_DRSRemoveDsDomain**).
 - Successfully processing an **IDL_DRSAddEntry** request can create an [nTDSDSA](#) object. When the directory is in this state (that is, when it holds the [nTDSDSA](#) object), an [IDL_DRSRemoveDsServer](#) request can successfully remove that [nTDSDSA](#) object.
 - Successfully processing an [IDL_DRSReplicaAdd](#) request creates a [repsFrom](#) value on a server. When a server is in this state (that is, when it holds the [repsFrom](#) value), it has the information needed to make an **IDL_DRSGetNCChanges** request on the DC that is specified in **IDL_DRSReplicaAdd**.
 - Successfully processing an [IDL_DRSUpdateRefs](#) request creates a [repsTo](#) value on a server. When a server is in this state (that is, when it holds the [repsTo](#) value), it has the information needed to make an [IDL_DRSReplicaSync](#) request on the DC that is specified in **IDL_DRSUpdateRefs**.

State-based sequencing issues also exist between methods specified in this document and LDAP because LDAP provides another way to change the state of the directory.

- One method, [IDL_DRSGetReplInfo](#), has a parameter of both input and output, `dwEnumerationContext`. This parameter is defined for the following:
 - `dwInVersion=2`, and
 - `InfoType=DS_REPL_INFO_METADATA_FOR_ATTR_VALUE`, or `DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE`, or `DS_REPL_INFO_CURSORS_2_FOR_NC`, or `DS_REPL_INFO_CURSORS_3_FOR_NC`.

For the first call to this method for a specific `InfoType`, the client sets `dwEnumerationContext` in `pmsgIn` to zero. The server returns an implementation-specific value for `dwEnumerationContext` in `pmsgOut`. On subsequent calls to this method with the same `InfoType`, the client sets the input `dwEnumerationContext` in `pmsgIn` to the last value of that field returned from the server. The purpose of this field is to allow the client to gather all the requested information, but in more than one server call. The final call is identified when the method returns `ERROR_NO_MORE_ITEMS`. See the server implementation section for [IDL_DRSGetReplInfo](#) ([4.1.13.3](#)) for exact details.

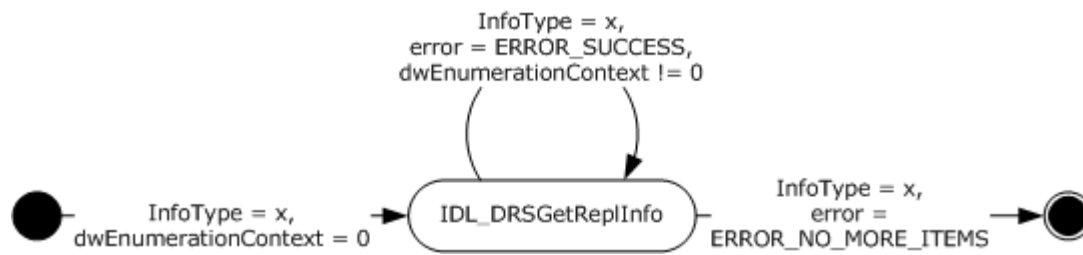


Figure 1: Using dwEnumerationContext

1.3.3 Most Frequently Used Types

The role of the [DRS_HANDLE](#) type, described in the previous section ([Sequencing Issues](#)), plays a central role in capability negotiation, as explained in the specification of IDL_DRSBind.

The type that is most central to this protocol is [DSNAME](#). [DSNAME](#) is the concrete type for the abstract [DSNAME](#) specified in [MS-ADTS] section 3.1.1.1.5. A [DSNAME](#) identifies an object. Nearly every method in the DRS protocol contains a [DSNAME](#) either in its request or its response.

Another basic type in the DRS protocol is [ENTINF](#). An [ENTINF](#) structure contains the [DSNAME](#) of an object (or object to be) and a list of attributes with associated values—[ATTRBLOCK](#). [ENTINF](#) and [ATTRBLOCK](#) are used in the following ways:

- To communicate some or all of the current state of an object:
 - In the response to an [IDL_DRSGetNCChanges](#) request, where multiple [ENTINF](#) structures are embedded in a [REPLENTINFLIST](#) structure. The request plus the stamps on the object determine what information about the object is included in the response.
 - In the response to an [IDL_DRSVerifyNames](#) request, which includes an [ENTINF](#) structure. The request specifies what information about the object is included in the response.
- To specify the state of an object to be created by a method:
 - In an [IDL_DRSAddEntry](#) request, where the request message is essentially an [ENTINF](#) structure but is not declared as such: The [DSNAME](#) and the [ATTRBLOCK](#) structures are separate top-level fields of the request.
 - In an [IDL_DRSInterDomainMove](#) request, where an [ENTINF](#) structure specifies the state of the object that is being created in another domain (unlike LDAP Add, with a specified [objectGUID](#)).
- To specify the subset of the current state of an object to be returned in a response:
 - In an [IDL_DRSVerifyNames](#) request, where an [ATTRBLOCK](#) structure specifies attributes but not their values.

1.4 Relationship to Other Protocols

This protocol includes replication based on the IP protocol—[IDL_DRSGetNCChanges](#). Active Directory (AD) also supports replication based on the SMTP protocol; SMTP-based replication is specified in [\[MS-SRPL\]](#).

Some of the AD state exposed by this protocol is also exposed by the AD implementation of LDAP; see [\[MS-ADTS\]](#) section 3.1.1.

Some methods in this protocol are exposed, in modified form, via LDAP. The LDAP versions are specified in [\[MS-ADTS\]](#) section 3.1.1.3.

- RootDSE constructed attributes: [msDS-ReplAllInboundNeighbors](#), [msDS-ReplConnectionFailures](#), [msDS-ReplLinkFailures](#), [msDS-ReplPendingOps](#), [msDS-ReplAllOutboundNeighbors](#), [msDS-ReplQueueStatistics](#) (these expose some functionality of [IDL_DRSGetReplInfo](#)), [dnsHostName](#), [dsServiceName](#), [isGlobalCatalogReady](#), [serverName](#) (these expose some functionality of [IDL_DRSDomainControllerInfo](#)).
- RootDSE modify operations: [becomeDomainMaster](#), [becomeInfrastructureMaster](#), [becomePdc](#), [becomeRidMaster](#), [becomeSchemaMaster](#), [replicateSingleObject](#), [removeLingeringObject](#). The last two operations expose some functionality of [IDL_DRSGetNCChanges](#).
- Object constructed attributes: [canonicalName](#) (this exposes some functionality of [IDL_DRSCrackNames](#)), [msDS-NCReplInboundNeighbors](#), [msDS-NCReplCursors](#), [msDS-ReplAttributeMetaData](#), [msDS-ReplValueMetaData](#), [msDS-NCReplOutboundNeighbors](#) (these expose some functionality of [IDL_DRSGetReplInfo](#)), [tokenGroups](#), [tokenGroupsNoGCAcceptable](#), [tokenGroupsGlobalAndUniversal](#) (these expose some functionality of [IDL_DRSGetMemberships](#) and [IDL_DRSGetMemberships2](#)).
- Controls: LDAP_SERVER_DIRSYNC_OID

The LDAP Control LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID is related to [IDL_DRSInterDomainMove](#) in that the LDAP client specifies via this control the DC whose [IDL_DRSInterDomainMove](#) method should be called (from the LDAP server implementation of Modify DN) to perform the move.

Some methods in this protocol have completely functional equivalents in LDAP:

- The function of [IDL_DRSWriteSPN](#) can be performed as an LDAP Modify of the [servicePrincipalName](#) attribute.
- The function of creating a [crossRef](#) object with IDL_DRSAddEntry can be performed as an LDAP Add of a [crossRef](#) object.

1.5 Prerequisites/Preconditions

This protocol is based on RPC and therefore has the prerequisites identified in [\[MS-RPCE\]](#) as common to all RPC interfaces.

Security configuration for usage of RPC is described further in section [2.2](#).

1.6 Applicability Statement

This protocol is appropriate for replicating (DC-to-DC only) and managing objects in a directory service, and for overall management of the directory service.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports:** RPC can be implemented on top of TCP and other protocol sequences as described in section [2.1](#).
- **Protocol Versions:** Each of the protocol interfaces described in this document has a single version number: 4.0 for [drsuapi](#) and 1.0 for [dsaop](#).

- **Security and Authentication Methods:** See [\[MS-RPCE\]](#) section 1.7.
- **Capability Negotiation:** This protocol does explicit capability negotiation as described in [IDL DRSBind \(section 4.1.3\)](#) behavior.

1.8 Vendor-Extensible Fields

This protocol does not define any vendor-extensible fields.

1.9 Standards Assignments

Parameter	Value	Reference
RPC Interface UUID for drsuapi methods	e3514235-4b06-11d1-ab04-00c04fc2dcd2	Section 4.1.1 - section 4.1.28
RPC Interface UUID for dsaop methods	7c44d7d4-31d5-424c-bd5e-2b3e1f323d22	Section 4.2.1 - section 4.2.2

2 Message Transport

The following sections discuss RPC transport and security considerations for this protocol. Common data types are defined and discussed in section 5. See section 3 for more details about the organization of this protocol specification.

2.1 RPC Transport

This protocol uses the following RPC protocol sequence: RPC over TCP as defined in [\[MS-RPCE\]](#). A server MAY listen on additional RPC protocol sequences. A client SHOULD attempt to connect using the RPC-over-TCP protocol sequence. [<1>](#)

This protocol uses RPC dynamic endpoints, as described in [\[C706\]](#) part 4.

Implementations MUST use the universally unique identifiers (UUIDs) as specified in section 1.9. The RPC version number is 4.0 for the [drsuapi](#) interface and 1.0 for the [dsaop](#) interface.

2.2 Protocol Security

This section describes the security mechanisms used for this RPC-based protocol.

2.2.1 General Background

A DC authenticates using its service account. In AD/DS, this is Local System on the machine running the DC, represented by the [computer](#) object of the machine. In AD/LDS, a DC's service account is configured by the administrator.

In AD/DS, connections for DC-to-DC communications MUST use mutual authentication and encryption of protocol traffic. Mutual authentication is provided by Kerberos (see [\[MS-KILE\]](#) section 3.3.1).

In AD/LDS, mutual authentication for DC-to-DC communications is not required. When a connection is established, the client uses the GSS Negotiate protocol, which first attempts to use Kerberos, and if Kerberos is unavailable, attempts NTLM (which does not give mutual authentication). If the [msDS-ReplAuthenticationMode](#) attribute on the config NC root equals 2, all DCs in the AD/LDS forest require mutual authentication for DC-to-DC communications.

Connections from a non-DC client to a DC do not require mutual authentication. Therefore, NTLM is an acceptable security provider in addition to Kerberos.

When a connection is established, the non-DC client uses the GSS Negotiate protocol, which first attempts to use Kerberos and then, if Kerberos is unavailable, attempts NTLM (which does not give mutual authentication).

2.2.2 Service Principal Names for Domain Controllers

In the absence of a trusted naming service, which maps service names to servers providing a given service, the client of a distributed service must authenticate a *service*, not a server. The client produces a service principal name (SPN), which is a name for the service it wants a connection to, and the authentication system verifies that the server is a provider of the named service.

Kerberos verifies the services provided by a server by reading the [servicePrincipalName](#) attribute of the server's [computer](#) object. The [servicePrincipalName](#) attribute contains a set of Unicode strings; each string is an SPN. If the client produces an SPN that is not present on the [computer](#) object of

the server it has requested a connection to, the mutual authentication fails and so does the connection attempt.

Each DC maintains the values of the [servicePrincipalName](#) attribute on its own [computer](#) object.

For the protocols specified in this document, the SPN produced by a client differs for DC-to-DC communications and non-DC-client-to-DC communications. The specific SPNs produced by a client in each scenario are described in the following sections.

2.2.3 DC-to-DC Operations

This section describes the security and mutual authentication requirements for those DRS protocol operations that involve DC-to-DC interactions. [<2>](#)

2.2.3.1 Security Provider

If mutual authentication is required, a DC client MUST request authentication, specifying the "Kerberos" security provider (RPC_C_AUTHN_GSS_KERBEROS). Regardless of whether mutual authentication is required, a DC client MUST request integrity and encryption of the RPC messages by specifying an authentication level of "packet privacy" (RPC_C_AUTHN_PKT_PRIVACY).

A DC client MUST authenticate the target DC by constructing an SPN for the service it is using. A DC client constructs an SPN as described in the following section.

2.2.3.2 SPN for a Target DC in AD/DS

Two different scenarios are possible when an AD/DS DC wants to connect to another DC for a DRS protocol operation:

- A DC wants to connect to a DC in a particular domain.
- A DC wants to connect to a GC server in the forest.

The scenario determines how the DC constructs an SPN for the service it is using:

- A DC wants to connect to a DC in a particular domain. The DC constructs the following SPN:
 - "<DRS interface GUID>/<DSA GUID>/<DNS domain name>"
- A DC wants to connect to a GC server in the forest. The DC constructs the following SPN:
 - "GC/<DNS hostname>/<DNS forest name>"

In the preceding SPN descriptions:

- "GC" is a literal string that represents a service class,
- The forward slash ('/') is the literal separator between parts of the SPN,
- <DRS interface GUID> is the fixed DRS RPC interface GUID, which has the well-known value of "E3514235-4B06-11D1-AB04-00C04FC2DCD2",
- <DSA GUID> is the DSA GUID of the target DC,
- <DNS domain name> is the DNS name of the domain of the target DC,
- <DNS hostname> is the DNS host name of the target DC, and

- <DNS forest name> is the DNS name of the forest of the target DC.

For example, the two SPNs that can be used for a DC named "dc1" with DSA GUID A5FF6869-AB5A-11D2-91E2-08002BA3ED3B in the contoso.com domain and forest are as follows:

- "E3514235-4B06-11D1-AB04-00C04FC2DCD2/A5FF6869-AB5A-11D2-91E2-08002BA3ED3B/contoso.com"
- "GC/dc1.contoso.com/contoso.com"

To allow mutual authentication to occur in DC-to-DC protocol operations, an AD/DS RODC MUST store the form of SPN that begins with "GC/", and other AD/DS DCs MUST store both forms of SPN, as values of the [servicePrincipalName](#) attribute of the DC's [computer](#) object. Additional forms that must be stored for client-to-DC protocol operations are described in section [2.2.4.2](#).

2.2.3.3 SPN for a Target DC in AD/LDS

When an AD/LDS DC wants to connect to another DC for a DRS protocol operation, it uses either of the following SPN forms:

- <DRS interface GUID>-ADAM/<DNS hostname>:<LDAP port>
- <DRS interface GUID>-ADAM/<NetBIOS hostname>:<LDAP port>

In the preceding SPN descriptions:

- <DRS interface GUID> is the fixed DRS RPC interface GUID, which has the well-known value of "E3514235-4B06-11D1-AB04-00C04FC2DCD2",
- "-ADAM/" is a literal string,
- <DNS hostname> is the full DNS host name of the target DC,
- <NetBIOS hostname> is the NetBIOS host name of the target DC,
- The colon (':') is the literal separator between the host name and port number, and
- <LDAP port> is the LDAP port on which the target DC listens.

If an AD/LDS DC runs on a machine joined to an AD domain, and NTDSDSA_OPT_DISABLE_SPN_REGISTRATION is not present in the [options](#) attribute of its [nTDSDSA](#) object ([\[MS-ADTS\]](#) section 7.1.1.2.2.1.2.1.1), the AD/LDS DC MUST store these two forms of SPN as values of the [servicePrincipalName](#) attribute of the DC's service account object. This action allows mutual authentication to occur in DC-to-DC protocol operations. Additional forms that must be stored for client-to-DC protocol operations are described in section [2.2.4.3](#).

2.2.4 Client-to-DC Operations

This section describes the security and mutual authentication requirements for those DRS protocol operations that involve client-to-DC interactions.

2.2.4.1 Security Provider

To request authentication, a client program specifies the "GSS Negotiate" security provider (RPC_C_AUTHN_GSS_NEGOTIATE). To request integrity and encryption of the RPC messages, a client program specifies an authentication level of "packet privacy" (RPC_C_AUTHN_PKT_PRIVACY).

To authenticate the target DC, a client program constructs an SPN for the service it is using, and negotiates Kerberos as the security provider. A client constructs an SPN as described in the following sections.

2.2.4.2 SPN for a Target DC in AD/DS

Three scenarios are possible when a client wants to connect to an AD/DS DC for a DRS protocol operation:

- A client wants to connect to a particular DC by using its host name, regardless of the domain it contains.
- A client wants to connect to a DC in a particular domain.
- A client wants to connect to a GC server in the forest.

The scenario determines how the client constructs an SPN for the service it is using:

- A client wants to connect to a particular DC by using its host name, regardless of the domain it contains. The client constructs any of the following three SPNs:
 - "ldap/<NetBIOS hostname>"
 - "ldap/<DNS hostname>"
 - "ldap/<DSA GUID based DNS hostname>"

The SPN that a client constructs depends on the information that the client has available. For example, some clients have only a NetBIOS name for a DC, while others have only a DNS name for a DC.

- A client wants to connect to a DC in a particular domain. The client constructs either of the following two SPNs:
 - "ldap/<DNS hostname>/<NetBIOS domain name>"
 - "ldap/<DNS hostname>/<DNS domain name>"

The SPN that a client constructs depends on the information that the client has available. For example, some clients have only a NetBIOS name for a domain, while others have only a DNS name for a domain.

- A client wants to connect to a GC server in the forest:
 - "GC/<DNS hostname >/<DNS forest name>"

In the preceding SPN descriptions:

- "ldap" and "GC" are literal strings representing service classes,
- The forward slash ('/') is the literal separator between parts of the SPN,
- <NetBIOS hostname> is the NetBIOS host name of the target DC,
- <DNS hostname> is the DNS host name of the target DC,
- <NetBIOS domain name> is the NetBIOS name of the domain of the target DC,

- <DNS domain name> is the DNS name of the domain of the target DC,
- <DSA GUID based DNS hostname> is the DNS host name of the target DC, constructed in the form "<DSA GUID>._msdcs.<DNS forest name>", and
- <DNS forest name> is the DNS name of the forest of the target GC server.

Continuing the previous example, the two- and three-part SPNs that can be used for a DC named "dc1" in the contoso.com domain are as follows:

- "ldap/DC1"
- "ldap/dc1.contoso.com"
- "ldap/6B352A21-8622-4F6D-A5A9-45CE9D7A5FB7._msdcs.contoso.com"
- "ldap/dc1.contoso.com/CONTOSO"
- "ldap/dc1.contoso.com/contoso.com"
- "GC/dc1.contoso.com/contoso.com"

To allow mutual authentication to occur in client-to-DC protocol operations, an AD/DS DC MUST store these six forms of SPN as values of the [servicePrincipalName](#) attribute of the DC's [computer](#) object. The GC SPN for client-to-DC is identical to the GC SPN for DC-to-DC. Therefore, when the requirements of this section are added to the requirements of section [2.2.3.2](#), an AD/DS RODC MUST store six, and other AD/DS DCs MUST store seven, [servicePrincipalName](#) values in all.

2.2.4.3 SPN for a Target DC in AD/LDS

When a client wants to connect to an AD/LDS DC for a DRS operation, it uses either of the following SPN forms:

- ldap/<DNS hostname>:<LDAP port>
- ldap/<NetBIOS hostname>:<LDAP port>

In the preceding SPN descriptions:

- "ldap" is the literal string representing the service class,
- The forward slash ('/') is the literal separator between parts of the SPN,
- <DNS hostname> is the full DNS host name of the target DC,
- <NetBIOS hostname> is the NetBIOS host name of the target DC,
- The colon (':') is the literal separator between the host name and port number, and
- <LDAP port> is the LDAP port on which the target DC listens.

If an AD/LDS DC runs on a computer joined to an external AD domain, and NTDSDSA_OPT_DISABLE_SPN_REGISTRATION is not present in the [options](#) attribute of its [nTDSDSA](#) object in AD/LDS (see [\[MS-ADTS\]](#) section 7.1.1.2.2.1.2.1.1), the AD/LDS DC MUST store these two forms of SPN as values of the [servicePrincipalName](#) attribute of the DC's service account object in the external AD domain. This action allows mutual authentication to occur in client-to-AD/LDS DC protocol operations. When the requirements of this section are added to the requirements of section [2.2.3.3](#), an AD/LDS DC that stores SPNs stores four [servicePrincipalName](#) values in all.

3 Background to Behavior Specifications

The following sections provide a background to understanding the specification of client and server behavior.

3.1 Document Organization

This document groups information that is relevant to only one RPC method with the specification of the behavior for that method. Information that is relevant to only one RPC method includes the IDL for the method itself, definitions for all types used exclusively by the method, and examples specific to the method.

Most methods specified in this document have no special client considerations. In such cases the entire specification of the method behavior is the specification of server behavior.

In cases where client behavior is specified, the client behavior in preparing a request is specified in the section immediately preceding the section that specifies server behavior, and the client behavior in processing a response is specified in the section immediately following the section that specifies server behavior. This ordering follows the flow of processing a request.

The behavior specification for some methods is followed immediately by one or more examples that show a request as seen by the server implementation of the method, the corresponding response created by the server implementation of the method, and the effect of server request processing on the state of the directory. Section [3.5.1](#) specifies the initial state used by all examples.

In cases where a type used in a method request or response is common to several methods, that type is placed in [Common Data Types, Variables and Procedures \(section 5\)](#). This section is arranged alphabetically, and so its table of contents serves as an index. This section is placed after the section that contains method behavior specifications, because typically a reader will reference the common types while reading the method specifications, and not the other way around.

3.2 Typographical Conventions

Sections of this document are not self-contained; they contain both forward and backward references, all of which are hyperlinked. In addition, the following typographical convention is used to indicate the special meaning of certain names:

- Underline, as in [instanceType](#): the name of an attribute or object class whose interpretation is specified in the following documents:
 - [\[MS-ADA1\]](#) Attribute names whose initial letter is A through L.
 - [\[MS-ADA2\]](#) Attribute names whose initial letter is M.
 - [\[MS-ADA3\]](#) Attribute names whose initial letter is N through Z.
 - [\[MS-ADSC\]](#) Object class names.
 - [\[MS-ADLS\]](#) Object class names and attribute names for AD/LDS.

No special typographical convention is used for names that represent elements of sets; for example, DRS_WRIT_REP. The name of the set type (for example, [DRS_OPTIONS](#)) is always clear from context, and the elements of each set type are defined with the set type. Similarly, no special typographical convention is used for names that represent Windows error codes; for example, ERROR_INVALID_PARAMETER.

3.3 State Model

3.3.1 Preliminaries

[\[MS-ADTS\]](#) section 3.1.1.1 is a prerequisite to the remainder of this specification.

3.3.2 Transactions

The specifications of client and server method behavior in this document do not mention transaction boundaries because all methods use transactions in a systematic way, as described in the remainder of this section.

In server processing of a normal method, a transaction begins implicitly on the first access to the database that represents the persistent state of the DC, and ends implicitly before a method returns. When a new logical thread of control is created (see Asynchronous Processing in section [3.4.6](#)), the originating thread implicitly ends its transaction before it returns and the new logical thread of control implicitly begins an unconnected transaction as described above.

If a transaction fails, and the method return would otherwise have been successful, the Windows error code returned by the method is in one of the following sets:

- **Retryable:** ERROR_DS_DRA_BUSY, ERROR_DS_OUT_OF_VERSION_STORE. There is a significant chance that retrying the request will succeed.
- **Implementation limit:** ERROR_DS_MAX_OBJ_SIZE_EXCEEDED. This error is returned when an implementation-specific, fixed size limit is exceeded. Retrying will not succeed, but the system is functioning normally.
- **Resource limit:** ERROR_DISK_FULL, ERROR_NO_SYSTEM_RESOURCES. Retrying will not succeed; an administrator must increase available resources.
- **Corruption:** ERROR_DS_KEY_NOT_UNIQUE, ERROR_DS_OBJ_NOT_FOUND, ERROR_DISK_OPERATION_FAILED. Retrying will not succeed; an administrator must repair the database that represents the persistent state of the DC or restore the database from backup.

If server processing of a normal method performs some updates and then detects an error condition, it terminates the current transaction before returning the error code that describes the error condition. If the transaction termination encounters an error condition, the method does not report the transaction-related error condition. Instead, the method reports the original error condition.

When the specification includes a client preparing a method request or processing a method response, the pattern is similar. When a client that is a DC prepares a method request, it implicitly begins a transaction on the first access to the database that represents the persistent state of the client DC and commits this transaction before sending the request. When a client that is a DC processes a method response, it implicitly begins a transaction on the first access to the database that represents the persistent state of the DC and commits this transaction when processing of the response is complete. A client transaction is never in progress while the client waits for the server to respond to a method request. There is no use of distributed transactions.

3.3.3 Concrete and Abstract Types

This protocol specification involves both concrete and abstract types.

A **concrete type** is a type whose representation must be standardized for interoperability. In this protocol specification, three cases apply:

- Types in the IDL definition of the [drsuapi](#) and [dsaop](#) RPC interfaces that determine the format of network requests and responses.
- Types that are hand marshaled onto the network, such as types that are sent in **drsuapi** and **dsaop** requests and responses as octet strings whose actual structure is hidden from the IDL compiler. The hand marshaling and corresponding hand unmarshaling are performed by the implementation of Active Directory (AD) and by clients of the **drsuapi** and **dsaop** RPC interfaces.
- Types that are hand marshaled into directory attributes, such as types that are stored in the directory as octet strings. The hand marshaling and corresponding hand unmarshaling are performed by the implementation of AD and by clients of the AD LDAP interface [\[MS-ADTS\]](#) section 3.1.1.3.

Concrete types in the first category are specified by the C / IDL type declaration. Concrete types in the second and third categories are specified pictorially. Some types are in multiple categories and are specified both ways.

All other types in the specification are *abstract*, meaning that their use is internal to the specification. Abstract types are based on the standard mathematical concepts set, sequence, directed graph, and tuple.

This specification introduces the notion of an abstract attribute. An *abstract attribute* is an AD attribute that has an abstract type for use in pseudocode. An abstract attribute can have a specified concrete representation, required for interoperability; in that case, the abstract attribute's type definition specifies the correspondence between information in the abstract type and in the concrete type. This relieves the specification pseudocode from concerns with storage allocation, packing variable-length information into structures, and so on.

Pseudocode deals with a mixture of concrete and abstract types. The notations and conventions for each are specified in section [3.4](#).

3.4 Pseudocode Language

3.4.1 Naming Conventions

Identifiers for concrete types, structure fields, and constants are used unchanged. The names of concrete types are often uppercase, with underscore characters ('_') to mark the divisions between words.

- Examples: [REPS_FROM](#), [DRS_MSG_UPDREFS](#)

Identifiers for object classes and attributes are LDAP display names from [\[MS-ADA1\]](#), [\[MS-ADA2\]](#), [\[MS-ADA3\]](#), and [\[MS-ADSC\]](#). These identifiers start with a lowercase letter; there are no capitalization conventions for the letters that follow the initial lowercase letter.

- Examples: [repsFrom](#), nTDSDSA

Identifiers for types and procedures introduced for specification purposes always start with an uppercase letter, and start each word after the first word with an uppercase letter (Pascal casing).

- Examples: RepsFrom, [ValidateSiteRDN](#)

Identifiers for variables introduced for specification purposes always start with a lowercase letter, and start each word after the first word with an uppercase letter (camel case).

- Examples: [dc](#), vSet

3.4.2 Language Constructs for Concrete Types

Concrete types support structure assignments between types that are not identical. For example:

```
reqV1: DRS_MSG_REPADD_V1
reqV2: DRS_MSG_REPADD_V2

reqV2 := reqV1
```

Such an assignment is shorthand for a field-by-field assignment for fields with the same name in the two structures. The preceding example is equivalent to the following:

```
reqV1: DRS_MSG_REPADD_V1
reqV2: DRS_MSG_REPADD_V2

reqV2.pNC := reqV1.pNC
reqV2.rtSchedule := reqV1.rtSchedule
reqV2.ulOptions := reqV1.ulOptions
```

The ADR built-in function returns the address of a variable. The ADDRESS OF type constructor creates a pointer type. These are needed occasionally when dealing with concrete structures.

Pseudocode does not perform storage allocation for concrete response structures. An implementation is free to allocate any amount of memory sufficient to contain the structures within the response.

3.4.3 Language Constructs for Abstract Types

The language includes the conventional types *Boolean* and *Integer*.

The notation [first .. 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 that corresponds 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 sequence type can be *open* (index type not specified) or *closed* (index type specified):

- type DSNameSeq = sequence of [DSName](#)
- type Digest = sequence [0 .. 15] of byte

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

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

Therefore:

- $s := []$

sets a sequence-valued variable s to the empty sequence. A sequence of bytes can be written in the more compact string form shown in the following example:

- `s := "\x55\x06\x02"`

A *unicodestring* is a sequence of 16-bit Unicode characters.

If *S* is a sequence, and $j \geq i$, then $S[i .. j]$ is a new sequence of length $j - i + 1$, whose first element has value $S[i]$, second element has value $S[i + 1]$, ... , and final element has value $S[j]$. The index set of the new sequence is $[0 .. j - i]$. If $j < i$ then $S[i .. j]$ is the empty sequence.

A *tuple* is a set of name-value pairs: $[name_1: value_1, name_2: value_2, \dots, name_n: value_n]$ where $name_k$ is an identifier and $value_k$ is the value bound to that identifier. Tuple types are defined as in the following example:

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

This example defines `DSName` as a tuple type with a `DN`-valued field dn, a `GUID`-valued field guid, and an `SID`-valued field sid.

A *tuple constructor* is written as in this example:

- `dsName: DSName`
- `dsName := [dn: "cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com"]`

Fields that are unspecified in a tuple constructor are assigned null values in the resulting tuple.

Access to the named fields of a tuple uses dot notation. Continuing the example:

- `d: DN; g: GUID; s: SID`
- `d := dsName.dn`
- `g := dsName.guid`
- `s := dsName.sid`

The preceding assignments set the variable *d* to "cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com", and variables *g* and *s* to null values.

A *tuple deconstructor* can be written anywhere a tuple-valued variable can occur. The preceding assignments are equivalent to the following:

- `[dn: d, guid: g, sid: s] := dsName;`

The language includes *sets*. If *S* is a set, $\text{number}(S)$ is the cardinality of the set *S*.

A fixed-size set can be constructed using the notation:

- `{one element, another element, ... , yet another element}`

Therefore:

- `S := {}`

sets a set-valued variable *S* to the empty set.

If *S* is a set, the predicate *x in S* is true if *x* is a member of *S*. Therefore, the value of the expression:

- `13 in {1, 2, 3, 5, 7, 11}`

is false.

If A and B are sets, $A + B$ is the set union of A and B, $A \cap B$ is the set intersection of A and B, and $A - B$ is the set difference of A and B.

The specification uses [KNUTH1] section 2.3.4.2 as a reference for the graph-related terms *directed graph*, *oriented tree*, *vertex*, *arc*, *initial vertex*, and *final vertex*. In pseudocode, graphs are described in terms of their vertex and arc sets, and individual arcs are represented as tuples.

The language supports coercion between abstract and concrete types when the correspondence between the two is clear. For example, if *stringSet* is a set of unicodestring and *stringArrayPtr* is a pointer to an array of pointers to null-terminated Unicode strings, the assignment:

- `stringSet := stringArrayPtr^`

populates the abstract set of strings by copying from the concrete array of strings.

3.4.4 Common Language Constructs

Syntax of standard control structures:

```
if boolean-expr then
  stmts
else
  stmts
endif

if boolean-expr then
  stmts
else if boolean-expr then : disambiguated by indentation
  stmts
endif

foreach var in set-or-sequence-expr
  stmts
endfor

for var := first-value to last-value
  stmts
endfor

while boolean-expr
  stmts
endwhile

return expr
```

Other constructs used (inspired by Modula-3; for more information, see [NELSON]):

```
: declare a procedure
: with typed args and result
procedure name(arg: type, arg: type, ... , arg: type): type

: declare a procedure
: with call-by-reference args
procedure name(var arg: type, ... , var arg: type): type
```

```

: cast a variable or an expression value
: to a different type
loophole(expr, type)

var: type      : declare a variable with a type
var := expr    : assignment
expr^          : pointer dereferencing
expr.id        : field selection

```

List of infix and prefix operator binding precedence (strongest binding at the top of the list):

```

x.a            : infix dot
f(x) a[i]      : applicative (, [
p^            : postfix ^
+ -           : prefix arithmetics
* / mod ∩      : infix arithmetics; set intersection
+ -           : infix arithmetics; set union and difference
= ≠ < ≤ ≥ > in : infix relations
not           : prefix not
and           : infix and
or            : infix or

```

All infix operators are left-associative, and so, for example:

```
a - b + c
```

means:

```
(a - b) + c
```

Parentheses can be used to override the precedence rules.

The infix Boolean operators "and" and "or" are evaluated left to right, conditionally. The expression "p and q" is true if both p and q are true. If p is false, q is not evaluated. The expression "p or q" is true if at least one of p and q are true. If p is true, q is not evaluated.

3.4.5 Access to Objects and Their Attributes

The specification contains many accesses to specific directory attributes. The specification uses the following concise notation for these accesses to aid readability. If o is a variable that contains a [DSName](#) or a [DN](#), then:

```
o!attr
```

... is an access to the [attr](#) attribute of the object named by the content of o, performed in the context of the NC replicas held by the server. In this notation, the name [attr](#) is a constant (like [objectGUID](#)), not a variable.

If the form o![attr](#) occurs in an expression context, it denotes a value. There are three possibilities:

- If the attr attribute is not present on o, the value of the expression is the distinguished value null.
- If the attr attribute is present and declared multi-valued, the value of the expression is a set that contains all the values of attr. If only one value is present, the value of the expression is a set that contains one element, the value.
- If the attr attribute is present and declared single-valued, the value of the expression is the value of attr.

If the form o!attr occurs on the left side of an assignment statement, it is used as a variable. The attr attribute need not already be present on o for this assignment to be well defined. The assignment:

```
o!attr := null
```

... removes the attr attribute from object o.

The distinguished value null is an admissible value for any type that is stored as the value of an attribute. Suppose, for example, that attr is a single-valued integer attribute. If attr is not present on object o, the assignment:

```
i := o!attr
```

... assigns the value null to the integer variable i. There is no ambiguity between this use of null and the use of null as the value of a pointer because pointer values cannot be stored as the value of an attribute.

The value null can be used in the following ways:

- Tested for equality or inequality.
- Used where a sequence value is expected; it is equivalent to [], the empty sequence.
- Used where a set value is expected; it is equivalent to {}, the empty set.
- Used within a set constructor, where it adds no element to the resulting set.

The value null cannot be used in other expressions involving normal values. Therefore:

```
i: integer
s: set

i := o!attr
s := { o!attr }
if i = null then /* attr not present on object o */
    s := s + o!attr
endif
```

... is a valid pseudocode sequence. If the attr attribute is not present on object o, the branch of the if statement will be executed, and the set s is empty. But the statement:

```
i := o!attr * 2
```

... is a specification error if the attr attribute is not present on object *o*.

Queries in this specification are expressed in one of the following two forms:

```
rt := select all scope where predicate
rt := select one scope where predicate
```

In either form of query, *scope* specifies the set of values or objects to be examined, and *predicate* specifies the subset of the scope that is the query result.

Scopes take the form:

```
var from set-of-values-or-objects
```

... where *var* is an identifier to be used in the predicate, and *set-of-values-or-objects* is a set of values or [DSNames](#) that designate objects. These sets can be the result of evaluating any expression; for example, they can be the values of local set-valued variables. But usually they are sets of values or objects from the directory; for example, in the following form:

```
var from o!attr
```

... the scope is the set of all values of attribute attr on object *o*; by the definition of null, the scope is the empty set if *o!attr* = null.

There are three special forms for scopes that are sets of objects:

```
var from children o
var from subtree o
var from all
```

Here, *o* is a DSName or DN valued variable. The form **children o** denotes the set of children of the object *o* within the NC of *o*. This form does not include the object *o* itself. The form **subtree o** denotes the set of all descendants of *o* within the NC of *o*, plus the object *o* itself. The form **all** denotes the set of all objects in all NC replicas held by the server.

The predicate is an arbitrary predicate that uses the scoping identifier (*var* above) as a variable. The query is evaluated by binding each value or object (in arbitrary order) to *var*, and then evaluating the predicate. If the predicate is true, the value or object is said to *satisfy* the predicate.

If the query takes the form "select all", the result of the evaluation is the set of all values or objects in the scope that satisfy the predicate. If the scope is a set of values, the type of the result is a set of values; otherwise, the type of the result is a set of DSName.

If the query takes the form select one, the result of the evaluation is any single value or object that satisfies the predicate, or null if no value or object satisfies the predicate. If more than one result is possible, the result is nondeterministic. If the scope is a set of values, the type of the result is the type of the value; otherwise, the type of the result is DSName.

Here is a query example:

```
rt := select one v from nc!repsTo where
    v.naDsa = pReq^.V1.pszDsaDest or
    v.uuidDsa = pReq^.V1.uuidDsaObjDest
if rt = null then
    /* no matching values */
endif
```

In the "children / subtree / all" forms, as specified, the scope includes normal objects, not tombstones. Adding the qualifier "-ts-included" to these forms expands the scope to include both normal objects and tombstones. For example, the expression:

```
select all o from subtree-ts-included nc
```

... returns the set that contains the DSNames of all objects and tombstones in the subtree that is rooted at the DSName nc.

3.4.6 Asynchronous Processing

Several methods involve "asynchronous processing" in which a method initiates a separate logical thread of control with some initial state, and then the method execution continues independently. However, all the documented operations are synchronous operations as specified in [\[MS-RPCE\]](#). No documented operations make use of RPC-defined asynchronous processing.

The phrase "logical thread of control" suggests that asynchronous processing can be implemented in a variety of ways, including message processing (where each message represents a logical thread of control), "heavyweight" processes that have exclusive use of an address space, system-level multi-threading within a single address space, thread pooling, and so on.

A method that uses asynchronous processing always returns its response immediately after initiating the separate logical thread of control; there is never any interaction with the new logical thread of control. The results of the new logical thread of control are visible only through its effects on the database representing the persistent state of the DC. If the server crashes before the new logical thread of control has completed all its documented effects, the new logical thread of control never has any effects.

Asynchronous processing is always performed in the security context of the server itself, not the security context of the client. Therefore, all necessary access checks **MUST** be performed before the new logical thread of control is initiated.

This design pattern is indicated by the following text in the pseudocode:

```
Asynchronous Processing: Initiate a logical thread of control
to process the remainder of this request asynchronously
```

3.5 Conventions for Protocol Examples

3.5.1 Common Configuration

This section specifies the test setup that is used for most of the examples presented in section [4](#). The behavior of certain methods can be highlighted only by starting from a different state. The

example section for such a method specifies the difference between the initial state used for that example and the state given here.

The configuration is a forest with two domains CONTOSO.COM (Forest Root Domain) and ASIA.CONTOSO.COM (Domain NC):

Forest: CONTOSO.COM

- The forest functional level is DS_BEHAVIOR_WIN2003 functional level, therefore only Windows Server 2003 or higher versions of DCs are present in the forest. All DCs are running Windows Server 2003 Enterprise Edition.

Domains:

- CONTOSO.COM (Forest Root Domain NC)
- ASIA.CONTOSO.COM (Domain NC)

Sites:

- Default-First-Site-Name
- Default-Second-Site-Name

DCs:

- Domain: CONTOSO.COM
 - CN=DC1, OU=DOMAIN CONTROLLERS, DC=CONTOSO, DC=COM,
 - CN=DC2, OU=DOMAIN CONTROLLERS, DC=CONTOSO, DC=COM,
- Domain: ASIA.CONTOSO.COM
 - CN=DCA1, OU=DOMAIN CONTROLLERS, DC=ASIA, DC=CONTOSO, DC=COM.

Domain-joined computer:

- Domain: CONTOSO.COM
 - CN=M1, CN=COMPUTERS, DC=CONTOSO, DC=COM.

Users added:

- Domain: CONTOSO.COM
 - CN =Kim Akers, CN =Users, DC =CONTOSO, DC =COM,
- Domain: ASIA.CONTOSO.COM
 - CN =Yan Li, CN =Users, DC = ASIA, DC =CONTOSO, DC =COM,

Groups added:

- Domain: CONTOSO.COM
 - CN =GroupA, CN =Users, DC =CONTOSO, DC =COM,
 - [objectSid](#): S-1-5-21-254470460-2440132622-709970653-1114

- [member](#): null
- [groupType](#): {GROUP_TYPE_RESOURCE_GROUP, GROUP_TYPE_SECURITY_ENABLED}
- CN = Administrators, CN =Builtin, DC =CONTOSO, DC =COM
- [objectSid](#): S-1-5-32-544
- [member](#): Domain Admins, Enterprise Admins, Local Administrator of DC1
- [groupType](#): {GROUP_TYPE_BUILTIN_LOCAL_GROUP, GROUP_TYPE_RESOURCE_GROUP, GROUP_TYPE_SECURITY_ENABLED}

3.5.2 Data Display Conventions

The typical (server behavior only) example shows an initial state, a request, a response, and a final state.

The initial and final states highlight the changes for methods that perform updates. If the method is a query then only the initial state is shown.

States are rendered using the LDP tool. The LDP transcript shown has been edited slightly for clarity. Specifically:

- The "Id" and "&msg" are not shown for each search request. Nor is the "0" that means "typesOnly = false".
- The actual attribute list is shown, in *italics*, within square brackets. The LDP tool does not show it in the transcript it produces.
- The numeric constant that controls the search scope is replaced by its [RFC2251](#) name: *baseObject*, *singleLevel*, or *wholeSubtree*.

For example, the string:

```
ldap_search_s(Id, "DC=CONTOSO,DC=COM", 0, "(objectclass=*)", attrList, 0, &msg)
```

in the LDP transcript is changed to:

```
ldap_search_s("DC=CONTOSO,DC=COM", baseObject, "(objectclass=*)", [repsTo])
```

assuming that the search requested that only the [repsTo](#) attribute be returned.

Requests and responses are rendered by using the Windows debugger in the context of the server (for server behavior) or client (for client behavior), with editing of the transcript for clarity. The following two edits are performed consistently:

- The [DRS_HANDLE](#) parameter is not shown.
- Where the value of a parameter is a binary large object (BLOB), the value is not shown, but instead expressed as *binary blob*.

3.6 Server and Client Initialization

The server MUST start the RPC service to listen on the incoming RPC. For server configurations, see section [2.1](#).

The client creates an RPC association, or binding, to the server RPC endpoint before calling an RPC method. The client MAY create a separate association for each method invocation (subject to the use of "context handles"), or it MAY [<3>](#) reuse an association for multiple invocations.

3.6.1 AD/LDS Specifics

It is possible to run multiple AD/LDS DCs on the same computer. All of these AD/LDS DCs listen on the same RPC interface ID. So that clients can distinguish between different instances of AD/LDS that are running on the same computer, each RPC endpoint is annotated (as specified in [\[C706\]](#)) with a string containing the LDAP port number on which the DC listens. For example, if two AD/LDS DCs are running on a computer, with one listening on port 389 and the other listening on port 50000, the RPC endpoints of the AD/LDS DCs are annotated with "389" and "50000", respectively.

For a client to establish an RPC connection to an AD/LDS DC, the client needs to know the name of the computer and the number of the LDAP port on which the AD/LDS DC is listening. First, the client establishes a connection to the endpoint mapper service on the computer. Next, the client enumerates all endpoints that are registered for the desired interface ID. Finally, the client selects the endpoint whose annotation equals the LDAP port number of the desired AD/LDS DC.

AD/DS DCs do not annotate their RPC endpoints. RPC endpoint annotation is not required for AD/DS because it is not possible to run multiple AD/DS DCs on a computer.

4 RPC Methods and Their Behavior

The methods for the drsuapi RPC interface are described in section [4.1](#).

The methods for the dsaop RPC interface are described in section [4.2](#).

4.1 drsuapi RPC Interface

This section specifies the methods for the **drsuapi** RPC interface of this protocol and the processing rules for the methods.

Methods in RPC Opnum Order

Method	Description
IDL_DRSBind	Creates a context handle necessary to call any other method in this interface. Opnum: 0
IDL_DRSUnbind	Destroys a context handle previously created by the IDL_DRSBind method. Opnum: 1
IDL_DRSReplicaSync	Triggers replication from another DC. Opnum: 2
IDL_DRSGetNCChanges	Replicates updates from an NC replica on the server. Opnum: 3
IDL_DRSUpdateRefs	Adds or deletes a value from the repsTo attribute of a specified NC replica. Opnum: 4
IDL_DRSReplicaAdd	Adds a replication source reference for the specified NC. Opnum: 5
IDL_DRSReplicaDel	Deletes a replication source reference for the specified NC. Opnum: 6
IDL_DRSReplicaModify	Updates the value for repsFrom for the NC replica. Opnum: 7
IDL_DRSVerifyNames	Resolves a sequence of object identities. Opnum: 8
IDL_DRSGetMemberships	Retrieves group membership for an object. Opnum: 9
IDL_DRSInterDomainMove	A helper method used in a cross-NC move LDAP operation. Opnum: 10
IDL_DRSGetNT4ChangeLog	Returns a sequence of PDC change log entries or the Windows NT 4.0 replication state. Opnum: 11

Method	Description
IDL_DRSCrackNames	Looks up each of a set of objects in the directory and returns it to the caller in the requested format. Opnum: 12
IDL_DRSSetSPN	Updates the set of service principal names (SPNs) on an object. Opnum: 13
IDL_DRSSetDsServer	Removes the representation of a DC from the directory. Opnum: 14
IDL_DRSSetDsDomain	Removes the representation of a domain from the directory. Opnum: 15
IDL_DRSGetDomainControllerInfo	Retrieves information about DCs in a given domain. Opnum: 16
IDL_DRSSetEntry	Adds one or more objects. Opnum: 17
IDL_DRSExecuteKCC	Validates the replication interconnections of DCs and updates them if necessary. Opnum: 18
IDL_DRSGetReplInfo	Retrieves the replication state of the server. Opnum: 19
IDL_DRSSetSidHistory	Adds one or more SIDs to the sIDHistory attribute of a given object. Opnum: 20
IDL_DRSGetMemberships2	Retrieves group memberships for a sequence of objects. Opnum: 21
IDL_DRSReplicaVerifyObjects	Verifies the existence of objects in an NC replica. Opnum: 22
IDL_DRSGetObjectExistence	Helps the client check the consistency of object existence between its replica of an NC and the server's replica of the same NC. Opnum: 23
IDL_DRSQuerySitesByCost	Determines the communication cost from a "from" site to one or more "to" sites. Opnum: 24
IDL_DRSInitDemotion	Performs the first phase of the removal of a DC from an AD/LDS forest. Opnum: 25
IDL_DRSReplicaDemotion	Replicates off all changes to the specified NC and moves any FSMOs held to another server. Opnum: 26
IDL_DRSFinishDemotion	Finishes or cancels the removal of a DC from an AD/LDS forest. Opnum: 27

The following considerations apply to the order of method calls. See section [1.3.2](#) for details.

- **IDL_DRSBind** must be called before any other method in order to obtain a context handle.
- After the [IDL_DRSUnbind](#) method is called, the context handle that was passed to **IDL_DRSUnbind** cannot be used for other method calls.
- [IDL_DRSInitDemotion](#) is called before the other demotion methods.
- All other method calls are independent, apart from their dependencies on the state of the directory.

Because the order of method call is generally nonsequential (except as noted above), the method sections following this section are arranged alphabetically by method name.

All methods MUST NOT throw exceptions.

4.1.1 IDL_DRSAddEntry (Opnum 17)

The **IDL_DRSAddEntry** method adds one or more objects.

```
ULONG IDL_DRSAddEntry(  
    [in, ref] DRS_HANDLE hDrs,  
    [in] DWORD dwInVersion,  
    [in, ref, switch_is(dwInVersion)]  
        DRS_MSG_ADDENTRYREQ* pmsgIn,  
    [out, ref] DWORD* pdwOutVersion,  
    [out, ref, switch_is(*pdwOutVersion)]  
        DRS_MSG_ADDENTRYREPLY* pmsgOut  
);
```

hDrs: The RPC context handle that is returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: A pointer to the request message.

pdwOutVersion: A pointer to the version of the response message.

pmsgOut: A pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.1.1 Method-Specific Concrete Types

4.1.1.1.1 DRS_MSG_ADDENTRYREQ

The **DRS_MSG_ADDENTRYREQ** union defines the request messages that are sent to the [IDL_DRSAddEntry](#) method.

```
typedef  
[switch_type(DWORD)]  
union {  
    [case(1)]  
        DRS_MSG_ADDENTRYREQ_V1 V1;  
    [case(2)]
```

```

        DRS_MSG_ADDENTRYREQ_V2 V2;
    [case(3)]
        DRS_MSG_ADDENTRYREQ_V3 V3;
} DRS_MSG_ADDENTRYREQ;

```

V1: Version 1 request (obsolete).

V2: Version 2 request (Windows 2000 and later).

V3: Version 3 request (Windows Server 2003 and later).

4.1.1.1.2 DRS_MSG_ADDENTRYREQ_V1

The **DRS_MSG_ADDENTRYREQ_V1** structure defines the request message sent to the [IDL DRSAddEntry](#) method. This request version is obsolete. <4>

```

typedef struct {
    [ref] DSNAME* pObject;
    ATTRBLOCK AttrBlock;
} DRS_MSG_ADDENTRYREQ_V1;

```

pObject: The identity of the object to add.

AttrBlock: The attributes of the object to add.

4.1.1.1.3 DRS_MSG_ADDENTRYREQ_V2

The **DRS_MSG_ADDENTRYREQ_V2** structure defines the request message sent to the [IDL DRSAddEntry](#) method. This request version is supported by Windows 2000 and later.

```

typedef struct {
    ENTINFLIST EntInfList;
} DRS_MSG_ADDENTRYREQ_V2;

```

EntInfList: The objects to be added.

4.1.1.1.4 DRS_MSG_ADDENTRYREQ_V3

The **DRS_MSG_ADDENTRYREQ_V3** structure defines the request message sent to the [IDL DRSAddEntry](#) method. This request version is supported by Windows Server 2003 and later.

```

typedef struct {
    ENTINFLIST EntInfList;
    DRS_SecBufferDesc* pClientCreds;
} DRS_MSG_ADDENTRYREQ_V3;

```

EntInfList: The objects to be added.

pClientCreds: The user credentials to authorize the operation.

4.1.1.1.5 DRS_MSG_ADDENTRYREPLY

The **DRS_MSG_ADDENTRYREPLY** union defines the response messages received from the [IDL DRSAddEntry](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_ADDENTRYREPLY_V1 V1;
    [case(2)]
        DRS_MSG_ADDENTRYREPLY_V2 V2;
    [case(3)]
        DRS_MSG_ADDENTRYREPLY_V3 V3;
} DRS_MSG_ADDENTRYREPLY;
```

V1: Version 1 response (obsolete).

V2: Version 2 response (Windows 2000 and later).

V3: Version 3 response (Windows Server 2003 and later).

4.1.1.1.6 DRS_MSG_ADDENTRYREPLY_V1

The **DRS_MSG_ADDENTRYREPLY_V1** structure defines the response message received from the [IDL DRSAddEntry](#) method. This response version is obsolete. [<5>](#)

```
typedef struct {
    GUID Guid;
    NT4SID Sid;
    DWORD errCode;
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
} DRS_MSG_ADDENTRYREPLY_V1;
```

Guid: The [objectGUID](#) of the added object.

Sid: The [objectSid](#) of the added object.

errCode: 0 if successful, or a DIRERR error code (section [4.1.1.1.25](#)) if a fatal error occurred.

dsid: The implementation-specific diagnostic code.

extendedErr: 0, [STATUS code](#), or Windows error code.

extendedData: The implementation-specific diagnostic code.

problem: 0 or [PROBLEM](#) error code.

4.1.1.1.7 DRS_MSG_ADDENTRYREPLY_V2

The **DRS_MSG_ADDENTRYREPLY_V2** structure defines the response message received from the [IDL DRSAddEntry](#) method.

```
typedef struct {
    [unique] DSNAME* pErrorObject;
    DWORD errCode;
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
    [range(0,10000)] ULONG cObjectsAdded;
    [size_is(cObjectsAdded)] ADDENTRY_REPLY_INFO* infoList;
} DRS_MSG_ADDENTRYREPLY_V2;
```

pErrorObject: Null, or the identity of the object that was being added when an error occurred.

errCode: 0 if successful, otherwise a [DIRERR](#) error code.

dsid: The implementation-specific diagnostic code.

extendedErr: 0, [STATUS code](#), or Windows error code.

extendedData: The implementation-specific diagnostic code.

problem: 0 or [PROBLEM](#) error code.

cObjectsAdded: The count of items in the infoList array.

infoList: The identities of the added objects. The item order matches the item order of values in the EntInfList field in the request structure.

4.1.1.1.8 DRS_MSG_ADDENTRYREPLY_V3

The **DRS_MSG_ADDENTRYREPLY_V3** structure defines the response message received from the [IDL DRSAddEntry](#) method.

```
typedef struct {
    DSNAME* pdsErrObject;
    DWORD dwErrVer;
    [switch_is(dwErrVer)] DRS_ERROR_DATA* pErrData;
    [range(0,10000)] ULONG cObjectsAdded;
    [size_is(cObjectsAdded)] ADDENTRY_REPLY_INFO* infoList;
} DRS_MSG_ADDENTRYREPLY_V3;
```

pdsErrObject: Null, or the identity of the object that was being added when an error occurred.

dwErrVer: MUST be set to 1.

pErrData: Null, or an error that occurred while processing the request.

cObjectsAdded: The count of items in the infoList array.

infoList: The identities of the added objects. The item order matches the item order of values in the EntInfList field in the request structure.

4.1.1.1.9 ADDENTRY_REPLY_INFO

The **ADDENTRY_REPLY_INFO** structure defines the identity of an object added by the [IDL_DRSAddEntry](#) method.

```
typedef struct {
    GUID objGuid;
    NT4SID objSid;
} ADDENTRY_REPLY_INFO;
```

objGuid: The [objectGUID](#) of the added object.

objSid: The [objectSid](#) of the added object.

4.1.1.1.10 DIRERR_DRS_WIRE_V1

The **DIRERR_DRS_WIRE_V1** union defines the error that occurred during processing of a request sent to the [IDL_DRSAddEntry](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        ATRERR_DRS_WIRE_V1 AtrErr;
    [case(2)]
        NAMERR_DRS_WIRE_V1 NamErr;
    [case(3)]
        REFERR_DRS_WIRE_V1 RefErr;
    [case(4)]
        SECERR_DRS_WIRE_V1 SecErr;
    [case(5)]
        SVCERR_DRS_WIRE_V1 SvcErr;
    [case(6)]
        UPDERR_DRS_WIRE_V1 UpdErr;
    [case(7)]
        SYSERR_DRS_WIRE_V1 SysErr;
} DIRERR_DRS_WIRE_V1;
```

AtrErr: Attribute errors.

NamErr: Name resolution error.

RefErr: Referral.

SecErr: Security error.

SvcErr: Service error.

UpdErr: Update error.

SysErr: System error.

4.1.1.1.11 ATRERR_DRS_WIRE_V1

The **ATRERR_DRS_WIRE_V1** structure defines attribute errors.

```
typedef struct {
    DSNAME* pObject;
    ULONG count;
    PROBLEMLIST_DRS_WIRE_V1 FirstProblem;
} ATRERR_DRS_WIRE_V1;
```

pObject: The identity of the object being processed when the error occurred.

count: The count of items in the FirstProblem linked list.

FirstProblem: The first element in the linked list of attribute errors.

4.1.1.1.12 PROBLEMLIST_DRS_WIRE_V1

The **PROBLEMLIST_DRS_WIRE_V1** structure defines an attribute error link entry.

```
typedef struct _PROBLEMLIST_DRS_WIRE_V1 {
    struct _PROBLEMLIST_DRS_WIRE_V1* pNextProblem;
    INTFORMPROB_DRS_WIRE_V1 intprob;
} PROBLEMLIST_DRS_WIRE_V1;
```

pNextProblem: Null, or a pointer to the next item in the list.

intprob: Attribute error description.

4.1.1.1.13 INTFORMPROB_DRS_WIRE_V1

The **INTFORMPROB_DRS_WIRE_V1** structure defines an attribute error.

```
typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
    ATTRTYP type;
    BOOL valReturned;
    ATTRVAL Val;
} INTFORMPROB_DRS_WIRE_V1;
```

dsid: The implementation-specific diagnostic code.

extendedErr: 0, STATUS code, or Windows error code.

extendedData: The implementation-specific diagnostic code.

problem: 0 or [PROBLEM](#) error code.

type: The attribute that was being processed when the error occurred.

valReturned: If true, the offending value is returned in the Val member.

Val: The offending value.

4.1.1.1.14 NAMERR_DRS_WIRE_V1

The **NAMERR_DRS_WIRE_V1** structure defines a name resolution error.

```
typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
    DSNAME* pMatched;
} NAMERR_DRS_WIRE_V1;
```

dsid: The implementation-specific diagnostic code.

extendedErr: 0, [STATUS](#) code, or Windows error code.

extendedData: The implementation-specific diagnostic code.

problem: 0 or [PROBLEM](#) error code.

pMatched: The best match for the supplied object identity.

4.1.1.1.15 REFERR_DRS_WIRE_V1

The **REFERR_DRS_WIRE_V1** structure defines a referral to other DCs.

```
typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    CONTREF_DRS_WIRE_V1 Refer;
} REFERR_DRS_WIRE_V1;
```

dsid: The implementation-specific diagnostic code.

extendedErr: 0, [STATUS](#) code, or Windows error code.

extendedData: The implementation-specific diagnostic code.

Refer: The DCs to contact to chase the referral.

4.1.1.1.16 NAMERESOP_DRS_WIRE_V1

The **NAMERESOP_DRS_WIRE_V1** structure defines the state of name resolution.

```
typedef struct {
    UCHAR nameRes;
    UCHAR unusedPad;
    USHORT nextRDN;
} NAMERESOP_DRS_WIRE_V1;
```

nameRes: MUST be the uppercase ASCII character "S".

unusedPad: MUST be 0.

nextRDN: MUST be 0.

4.1.1.1.17 DSA_ADDRESS_LIST_DRS_WIRE_V1

The **DSA_ADDRESS_LIST_DRS_WIRE_V1** structure defines a linked list entry for a referral network name.

```
typedef struct _DSA_ADDRESS_LIST_DRS_WIRE_V1 {
    struct _DSA_ADDRESS_LIST_DRS_WIRE_V1* pNextAddress;
    UNICODE_STRING* pAddress;
} DSA_ADDRESS_LIST_DRS_WIRE_V1;
```

pNextAddress: Null, or the next element in the linked list.

pAddress: Network name of the DC to which the referral is directed.

4.1.1.1.18 CONTREF_DRS_WIRE_V1

The **CONTREF_DRS_WIRE_V1** structure defines a linked list entry for a continuation referral.

```
typedef struct CONTREF_DRS_WIRE_V1 {
    DSNAME* pTarget;
    NAMERESOP_DRS_WIRE_V1 OpState;
    USHORT aliasRDN;
    USHORT RDNsInternal;
    USHORT refType;
    USHORT count;
    DSA_ADDRESS_LIST_DRS_WIRE_V1* pDAL;
    struct CONTREF_DRS_WIRE_V1* pNextContRef;
    BOOL bNewChoice;
    UCHAR choice;
} CONTREF_DRS_WIRE_V1;
```

pTarget: The object to which the referral is directed.

OpState: The operation state.

aliasRDN: MUST be 0.

RDNsInternal: MUST be 0.

refType: The type of referral. This field MUST be one of the following values:

Value	Meaning
CH_REFTYPE_SUPERIOR 0x0000	A referral to a superior DC.
CH_REFTYPE_SUBORDINATE 0x0001	A referral to a subordinate DC (for example, to a child domain).
CH_REFTYPE_NSSR 0x0002	Not in use.
CH_REFTYPE_CROSS 0x0003	A referral to an external crossRef object. See [MS-ADTS] section 7.1.1.2.1.1.1.

count: The count of items in the pDAL linked list.

pDAL: A list of network names of the DCs to which the referral is directed.

pNextContRef: Null, or the next item in the linked list.

bNewChoice: True if and only if a new choice is specified.

choice: The choice to use in the continuation referral. This field MUST be one of the following values:

Value	Meaning
SE_CHOICE_BASE_ONLY 0x00	A base search should be performed.
SE_CHOICE_IMMED_CHLDRN 0x01	A one-level search should be performed.
SE_CHOICE_WHOLE_SUBTREE 0x02	A subtree search should be performed.

4.1.1.1.19 SECERR_DRS_WIRE_V1

The **SECERR_DRS_WIRE_V1** structure defines a security error.

```
typedef struct {  
    DWORD dsid;  
    DWORD extendedErr;  
    DWORD extendedData;  
    USHORT problem;  
} SECERR_DRS_WIRE_V1;
```

dsid: The implementation-specific diagnostic code.

extendedErr: 0, [STATUS](#) code, or Windows error code.

extendedData: The implementation-specific diagnostic code.

problem: 0 or [PROBLEM](#) error code.

4.1.1.1.20 SVCERR_DRS_WIRE_V1

The **SVCERR_DRS_WIRE_V1** structure defines a service error.

```
typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
} SVCERR_DRS_WIRE_V1;
```

dsid: The implementation-specific diagnostic code.

extendedErr: 0, [STATUS](#) code, or Windows error code.

extendedData: The implementation-specific diagnostic code.

problem: 0 or [PROBLEM](#) error code.

4.1.1.1.21 UPDERR_DRS_WIRE_V1

The **UPDERR_DRS_WIRE_V1** structure defines an update error.

```
typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
} UPDERR_DRS_WIRE_V1;
```

dsid: The implementation-specific diagnostic code.

extendedErr: 0, [STATUS](#) code, or Windows error code.

extendedData: The implementation-specific diagnostic code.

problem: 0 or [PROBLEM](#) error code.

4.1.1.1.22 SYSERR_DRS_WIRE_V1

The **SYSERR_DRS_WIRE_V1** structure defines a system error.

```
typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
} SYSERR_DRS_WIRE_V1;
```

dsid: The implementation-specific diagnostic code.

extendedErr: 0, [STATUS](#) code, or Windows error code.

extendedData: The implementation-specific diagnostic code.

problem: 0 or [PROBLEM](#) error code.

4.1.1.1.23 DRS_ERROR_DATA

The **DRS_ERROR_DATA** union defines the error responses that are received from the [IDL DRSAddEntry](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_ERROR_DATA_V1 V1;
} DRS_ERROR_DATA;
```

V1: Version 1 response.

4.1.1.1.24 DRS_ERROR_DATA_V1

The **DRS_ERROR_DATA_V1** structure defines the error response received from the [IDL DRSAddEntry](#) method.

```
typedef struct {
    DWORD dwRepError;
    DWORD errCode;
    [switch_is(errCode)] DIRERR_DRS_WIRE_V1* pErrInfo;
} DRS_ERROR_DATA_V1;
```

dwRepError: 0 or a Windows error code.

errCode: A DIRERR code (section [4.1.1.1.25](#)) that specifies the error category.

pErrInfo: Category-specific error information.

4.1.1.1.25 DIRERR Codes

The DIRERR codes classify an error that occurs during a search for, or the update of, a directory object.

Symbolic name	Value	Meaning
attributeError	0x00000001	Attribute errors.
nameError	0x00000002	Name resolution error.
referralError	0x00000003	Referral.

Symbolic name	Value	Meaning
securityError	0x00000004	Security error.
serviceError	0x00000005	Service error.
updError	0x00000006	Update error.
systemError	0x00000007	System error.

4.1.1.1.26 PROBLEM Error Codes

The PROBLEM error codes describe the problems that can be reported by an update operation.

Symbolic name	Value	Meaning
ENOMEM	0x0000000C	Out of memory.
EBUSY	0x00000010	Too busy to proceed.
EINVAL	0x00000016	Invalid parameter.
ENOSPC	0x0000001C	Out of storage.
NA_PROBLEM_NO_OBJECT	0x000007D1	Parent object not found in the NC replica.
PR_PROBLEM_NO_ATTRIBUTE_OR_VAL	0x000003E9	Attribute or value not found.
PR_PROBLEM_INVALID_ATT_SYNTAX	0x000003EA	Invalid attribute syntax.
PR_PROBLEM_UNDEFINED_ATT_TYPE	0x000003EB	Unknown attribute type.
PR_PROBLEM_WRONG_MATCH_OPER	0x000003EC	Incorrect matching operation (only applies to match operators in LDAP filters).
PR_PROBLEM_CONSTRAINT_ATT_TYPE	0x000003ED	Attribute value violated a schema constraint.
PR_PROBLEM_ATT_OR_VALUE_EXISTS	0x000003EE	Attribute or value already exists (multiple values specified for a single-valued attribute OR duplicate value specified for a multi-valued attribute).
SE_PROBLEM_INAPPROPRIATE_AUTH	0x00000FA1	Inappropriate authentication method.
SE_PROBLEM_INVALID_CREDENTIALS	0x00000FA2	Invalid user name or password.
SE_PROBLEM_INSUFF_ACCESS_RIGHTS	0x00000FA3	Access denied.
SE_PROBLEM_INVALID_SIGNATURE	0x00000FA4	Invalid signature.
SE_PROBLEM_PROTECTION_REQUIRED	0x00000FA5	Encrypted connection required.
SE_PROBLEM_NO_INFORMATION	0x00000FA6	Insufficient permissions to generate a referral.
SV_PROBLEM_BUSY	0x00001389	Directory service is busy.
SV_PROBLEM_UNAVAILABLE	0x0000138A	Directory service is unavailable.
SV_PROBLEM_WILL_NOT_PERFORM	0x0000138B	The requested operation will not be performed.

Symbolic name	Value	Meaning
SV_PROBLEM_CHAINING_REQUIRED	0x0000138C	Chaining is required to perform the operation.
SV_PROBLEM_UNABLE_TO_PROCEED	0x0000138D	Directory service is unable to proceed with the requested operation.
SV_PROBLEM_INVALID_REFERENCE	0x0000138E	Invalid crossRef object.
SV_PROBLEM_TIME_EXCEEDED	0x0000138F	Time limit exceeded while processing the operation.
SV_PROBLEM_ADMIN_LIMIT_EXCEEDED	0x00001390	Administrative limit exceeded while processing the operation.
SV_PROBLEM_LOOP_DETECTED	0x00001391	Not in use.
SV_PROBLEM_UNAVAIL_EXTENSION	0x00001392	The requested extended operation is not available.
SV_PROBLEM_OUT_OF_SCOPE	0x00001393	Not in use.
SV_PROBLEM_DIR_ERROR	0x00001394	Generic directory service error.
UP_PROBLEM_NAME_VIOLATION	0x00001771	Naming violation.
UP_PROBLEM_OBJ_CLASS_VIOLATION	0x00001772	Object class violation.
UP_PROBLEM_CANT_ON_NON_LEAF	0x00001773	The operation cannot be performed on an object with child objects.
UP_PROBLEM_CANT_ON_RDN	0x00001774	The operation cannot be performed on an RDN attribute.
UP_PROBLEM_ENTRY_EXISTS	0x00001775	Object already exists.
UP_PROBLEM_AFFECTS_MULT_DSAS	0x00001776	The operation affects multiple DCs.
UP_PROBLEM_CANT_MOD_OBJ_CLASS	0x00001777	The objectClass attribute cannot be modified in this way.

4.1.1.2 Method-Specific Abstract Types and Procedures

4.1.1.2.1 ConstructReplSpn

```
procedure ConstructReplSpn(
    dnsHostName: uncodestring,
    guid: GUID): uncodestring
```

This procedure returns a replication SPN with the given DNS host name (in `dnsHostName`) and GUID (in `guid`). The service class of the resulting SPN is [DRS_SPN_CLASS](#). For example, given `dnsHostName = "dc-01.fabrikam.com"` and `guid` being the GUID whose string representation is "{d66e9688-66a5-4a52-8af2-17b110febe0c}", the return value is:

```
E3514235-4B06-11D1-AB04-00C04FC2DCD2/d66e9688-66a5-4a52-8af2-17b110febe0c/dc-01.fabrikam.com
```

4.1.1.2.2 CreateCrossRef

```
procedure CreateCrossRef(  
    hDrs: DRS_HANDLE,  
    e: ENTINF,  
    pmsgOut: ADDRESS OF DRS_MSG_ADDENTRYREPLY,  
    ver: DWORD,  
    info: ADDRESS OF ADDENTRY_REPLY_INFO): boolean
```

Informative summary of behavior: This procedure creates a [crossRef](#) object. If the [crossRef](#) object exists already in a disabled state, it will mark the [crossRef](#) object as enabled.

```
ulSysFlags, err: DWORD  
ncNameV: DSName  
trustParentV, rootTrustV, dnsRootV: unicodestring  
cr: DSName  
prefixTable: PrefixTable  
/* Only attributes and classes in the base schema may be specified.*/  
prefixTable := NewPrefixTable()  
ulSysFlags := ENTINF_GetValue(e, systemFlags, prefixTable)  
ncNameV := ENTINF_GetValue(e, ncName, prefixTable)  
/* Check whether the crossRef object for the given ncName exists. */  
cr := select one v from subtree ConfigNC()  
    where v!ncName = ncNameV and crossRef in v!objectClass  
if (cr = null) or not (FLAG_CR_NTDS_DOMAIN in ulSysFlags) then  
    if FLAG_CR_NTDS_NC in ulSysFlags then  
        SetErrorData(SV_PROBLEM_WILL_NOT_PERFORM, serviceError,  
            ERROR_DS_MISSING_EXPECTED_ATT, pmsgOut, ver)  
        return false  
    endif  
    /* Add the crossRef object as a regular operation; this is subject  
    * to an access check and will succeed only if the server is the  
    * Partition Naming Master FSMO role owner. */  
    err := PerformAddOperation(e, cr, dc.prefixTable)  
    if err ≠ 0 then  
        /* Pick up the error information from the previous call. */  
        SetErrorData(0, 0, 0, pmsgOut, ver)  
        return false  
    endif  
    /* Set the systemFlags because PerformAddOperation does not set it.  
    */  
    cr!systemFlags := ulSysFlags  
    /* Return the objectGUID of the new crossRef object. */  
    info^.objGuid := cr.guid;  
else  
    /* crossRef already exists; enable it. */  
    /* The crossRef is expected to be disabled. */  
    if cr!enabled = null or cr!enabled = true then  
        SetErrorData(SV_PROBLEM_DIR_ERROR,  
            serviceError,  
            ERROR_DUP_DOMAINNAME,  
            pmsgOut, ver)  
        return false  
    endif  
    /* Only allow certain client IP to make the change. */  
    if not (ClientIpMatch(hDrs, cr!dnsRoot)) then  
        SetErrorData(SE_PROBLEM_INAPPROPRIATE_AUTH, securityError,
```

```

        ERROR_DS_INTERNAL_FAILURE, pmsgOut, ver)
    return false
endif

/* dnsRoot must be set in the given ENTINF. */
dnsRootV := ENTINF_GetValue(e, dnsRoot, prefixTable)
if dnsRootV = null then
    SetErrorData(PR_PROBLEM_NO_ATTRIBUTE_OR_VAL, attributeError,
        ERROR_DS_MISSING_REQUIRED_ATT, pmsgOut, ver)
    return false
endif
cr!dnsRoot := dnsRootV
/* Two more attributes can be set; the rest are ignored. */
trustParentV := ENTINF_GetValue(e, trustParent, prefixTable)
if trustParentV ≠ null then
    cr!trustParent := trustParentV
endif
rootTrustV := ENTINF_GetValue(e, rootTrust, prefixTable)
if rootTrustV ≠ null then
    cr!rootTrust := rootTrustV
endif
/* Update the systemFlags and enable the crossRef. */
cr!systemFlags := {FLAG_CR_NTDS_NC, FLAG_CR_NTDS_DOMAIN}
cr!enabled := null
/* return the guid of the crossRef object */
info^.objGuid := cr.guid
endif
return true

```

4.1.1.2.3 CreateNtdsDsa

```

procedure CreateNtdsDsa(
    hDrs: DRS_HANDLE,
    e: ENTINF,
    entList: ADDRESS OF ENTINFLIST,
    pmsgOut: ADDRESS OF DRS_MSG_ADDENTRYREPLY,
    ver: DWORD,
    info: ADDRESS OF ADDENTRY_REPLY_INFO): boolean

```

Informative summary of behavior: This procedure creates an [nTDSDSA](#) object.

```

domainName, domainCR, domain, cr, v,
    partitionsObj, sl, dsaObj: DSName
accessAllowed: boolean
dcfl, err: DWORD
spn: unicodestring
prefixTable: PrefixTable

/* Only attributes and classed in the base schema may be specified.*/
prefixTable := NewPrefixTable()

domainName := GetDomainNameFromEntinf(e)

domainCR := select one v from ConfigNC() where v!nCName = domainName
    and crossRef in v!objectClass

```

```

    and FLAG_CR_NTDS_DOMAIN in v!systemFlags

domain := select one v from all where v = domainName

if domain ≠ null then
    /* Perform access check. */
    accessAllowed :=
        AccessCheckCAR(domain, DS-Replication-Manage-Topology)
else
    /* If the domain crossRef is created in the same call, we will
    * allow it. The call will fail if the caller does not have right
    * to create the crossRef object. */
    accessAllowed := IsDomainToBeCreated(entList, domain)
endif

if not accessAllowed then
    SetErrorData(SV_PROBLEM_DIR_ERROR, serviceError,
        ERROR_ACCESS_DENIED, pmsgOut, ver)
    return false
endif

/* Check for the functional level compliance. The functional level
* of a DC cannot be less than the functional level of the forest.
* If the DC is not the first DC in its domain, its functional level
* cannot be less than the functional level of its domain. */
dcfl := ENTINF_GetValue(e, msDS-Behavior-Version, prefixTable)
if dcfl = null then
    dcfl := 0
endif

if domain = DefaultNC() and
    dcfl < DefaultNC()!msDS-Behavior-Version then
    SetErrorData(SV_PROBLEM_WILL_NOT_PERFORM, serviceError,
        ERROR_DS_INCOMPATIBLE_VERSION, pmsgOut, ver)
    return false
endif

partitionsObj := DescendantObject(ConfigNC(), "CN=Partitions,")

if dcfl < partitionsObj!msDS-Behavior-Version then
    SetErrorData(SV_PROBLEM_WILL_NOT_PERFORM, serviceError,
        ERROR_DS_INCOMPATIBLE_VERSION, pmsgOut, ver)
    return false
endif

/* serverReference attribute is not updated here; instead, it is used
* to find the computer object of the DC so that the replication SPN
* can be added to the DC's computer object. */
sl := ENTINF_GetValue(e, serverReference, prefixTable)
ENTINF_SetValue(e, serverReference, null, prefixTable)

/* Create the object in the system context; this is necessary to
* avoid the system-only class constraint defined in the schema.*/
err := PerformAddOperationAsSystem(e, dsaObj, prefixTable)
if err ≠ 0 then
    /* Pick up the error information PerformAddOperationAsSystem set.*/
    SetErrorData(0, 0, 0, pmsgOut, ver)
    return false

```

```

endif

/* Find the computer object and update its SPN. */
if sl ≠ null then
    dcObj := select one v from subtree DefaultNC() where v = sl
    spn := ConstructReplSpn(domainCR!dnsHostName, dcObj.guid)
    dcObj!serverPrincipalName := dcObj!serverPrincipalName + {spn}
endif

/* Return the objectGUID of the new nTDSDSA object. */
info^.objGuid := dsaObj.guid

return true

```

4.1.1.2.4 UseCredsForAccessCheck

```

procedure UseCredsForAccessCheck(creds: DRS_SecBufferDesc): DWORD

```

This procedure gets authorization information for a client (using the [ClientAuthorizationInfo](#) abstract type, which is a security token) by authenticating the given credentials. Any access checks performed during the remainder of the RPC call are performed against this information.

4.1.1.2.5 IsDomainToBeCreated

```

procedure IsDomainToBeCreated(
    entList: ADDRESS OF ENTINFLIST,
    ncName: DSName): boolean

```

This procedure searches all the [ENTINF](#) values in entList for any request to create a [crossRef](#) object cr such that cr![nCName](#) = ncName. It returns true if such a cr is found; otherwise, it returns false.

4.1.1.2.6 GetDomainNameFromEntinf

```

procedure GetDomainNameFromEntinf(e: ENTINF): DSName

```

This procedure retrieves the value ncs for [hasMasterNCs](#) from e. If ncs contains [ConfigNC\(\)](#), [SchemaNC\(\)](#), and another value dnc, it returns dnc. Otherwise, it returns null.

4.1.1.2.7 SetErrorData

```

procedure SetErrorData(
    problem: USHORT,
    errCode: ULONG,
    extendedError: ULONG,
    pmsgOut: ADDRESS OF DRS_MSG_ADDENTRYREPLY,
    version: ULONG)

```

This procedure sets the error message fields of **pmsgOut**: the **problem**, **errCode**, and **extendedErr** fields of pmsgOut^.V2 if version = 2 or the **pErrData** field of pmsgOut^.V3 if version

= 3. If **problem**, **errCode**, and **extendedError** are all 0, the error information is the result of the last call to [PerformAddOperation](#) or [PerformAddOperationToSystem](#).

4.1.1.2.8 ClientIpMatch

```
procedure ClientIpMatch(  
    hDrs: DRS_HANDLE,  
    dnsRoot: set of unicodestring): boolean
```

This function returns true if the IP address of the client with [DRS_HANDLE](#) hDrs matches one of the IP addresses of the DNS host names in the set [dnsRoot](#).

4.1.1.2.9 PerformModifyOperation

```
procedure PerformModifyOperation(  
    hDrs: DRS_HANDLE,  
    e: ENTINF,  
    info: ADDRESS OF ADDENTRY_REPLY_INFO): boolean
```

This function performs a modify operation on the object e.pName[^]. It enforces all security, schema, and other constraints and follows all processing rules as used by the LDAP modify operation (see [\[MS-ADTS\]](#)). The [objectGUID](#) and [objectSid](#) of the object being modified are returned in the info output structure. If the operation succeeds, [PerformModifyOperation](#) returns true. If the operation fails for some reason, [PerformModifyOperation](#) sets an appropriate error code (as defined by the LDAP modify operation) in the info structure, and returns false.

4.1.1.3 Server Behavior of the IDL_DRSAddEntry Method

Informative summary of behavior: A disabled [crossRef](#) object cr is one with cr![Enabled](#) = false. Enabling a disabled [crossRef](#) object cr means setting cr![nCName](#) and cr![dnsRoot](#), and removing cr![Enabled](#).

This method enables, creates, or modifies one or more objects, as requested by the client, in a single transaction. It enables [crossRef](#) objects, creates [crossRef](#) objects and [nTDSDSA](#) objects, and modifies arbitrary objects. The client uses an [ENTINF](#) structure to specify the state of each enabled, created, or modified object:

- Enabling a [crossRef](#) object: The [dnsRoot](#) attribute of a disabled [crossRef](#) object contains a set of one or more DNS host names, expressed as Unicode strings. The request to enable a [crossRef](#) object succeeds only if the IP address of the client that is making the request matches the IP address of one of the DNS host names in the [dnsRoot](#) attribute. When a disabled [crossRef](#) object is enabled through this method, the server is not required to be the Domain Naming Master FSMO role owner.

The client must specify the [nCName](#) and [dnsRoot](#) attributes. The [trustParent](#) and [rootTrust](#) attributes are optional.

- Creating a [crossRef](#) object: If the request creates a [crossRef](#) object, it succeeds only if the server owns the forest's Domain Naming Master FSMO role. The access check is the same as when a [crossRef](#) object is created through LDAP.

The client must specify the same attributes that are required during an LDAP Add of a [crossRef](#) object, namely the new object's DN, plus all must-have attributes of the [crossRef](#) class. See [\[MS-ADTS\]](#) section 7.1.1.2.1.1 for the specification of [crossRef](#) objects.

- Creating an [nTDSDSA](#) object: Creating an [nTDSDSA](#) object is not possible with LDAP. To create an [nTDSDSA](#) object, the [hasMasterNCs](#) attribute in the request must identify the forest's schema NC and config NC, and the DC's default NC; that is, the domain of the DC corresponding to the new [nTDSDSA](#) object. If the default NC exists on the server as the [nTDSDSA](#) object is being created by [IDL_DRSAddEntry](#), the client must have the control access right DS-Replication-Manage-Topology on the default NC. Otherwise, the client must have the right to enable or create the [crossRef](#) object that corresponds to the default NC, and must enable or create this [crossRef](#) object in the same [IDL_DRSAddEntry](#) request.

The client must specify the new object's DN, plus the [hasMasterNCs](#) attribute. To create an [nTDSDSA](#) object for a functional DC, the request will contain [invocationId](#), [dmdLocation](#), [options](#), [msDS-Behavior-Version](#), and [systemFlags](#). See [MS-ADTS] section 7.1.1.2.2.1.2.1.1 for the specification of [nTDSDSA](#) objects.

If the [serverReference](#) attribute is given a value in the request, the [computer](#) object to which the [serverReference](#) attribute points is updated with a new replication SPN.

- Modifying an object: To modify an existing object (other than enabling a [crossRef](#) object), the client-supplied **ENTINF** structure includes ENTINF_REMOTE_MODIFY in the ulFlags field and specifies the modified attributes and their values. The client must have the same rights as those needed to perform the modification via LDAP. The DC enforces the same schema and other constraints on the modification as if performed via LDAP. Performing the modification by using [IDL_DRSAddEntry](#) rather than LDAP allows changes to multiple objects to be made in a single transaction. This operation is only supported by AD/LDS and AD/DS in Windows Server 2008.

```
ULONG
IDL_DRSAddEntry(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_ADDENTRYREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_ADDENTRYREPLY *pmsgOut)

ext: DRS_EXTENSIONS_INT
pEntInfList: ADDRESS OF ENTINFLIST
pClientCreds: ADDRESS OF DRS_SecBufferDesc
objCls : ATTRTYP
ncNameV: DSName
infoList: ADDENTRY_REPLY_INFO
cObjects: ULONG
res: boolean
prefixTable: PrefixTable
/* Only attributes and classes in the base schema may be specified.*/
prefixTable := NewPrefixTable()
/* Validate parameters. */
if not (dwInVersion in {2,3}) then
    SetErrorData(SV_PROBLEM_BUSY, 0, ERROR_INVALID_PARAMETER,
        pmsgOut, 2)
    return 0
endif
/* Originating updates are blocked on an RODC */
if AmIRODC() then
    SetErrorData(NA_PROBLEM_NO_OBJECT, 0, ERROR_DS_DRA_INTERNAL_ERROR,
        pmsgOut, pdwOutVersion^)
endif
```



```

/* If the client supports the version 3 response, use version 3. */
ext := ClientExtensions(hDrs)
if DRS_EXT_ADDENTRYREPLY_V3 in ext.dwFlags then
    pdwOutVersion^ := 3
else
    pdwOutVersion^ := 2
endif
cObjects := 0
if dwInVersion = 2 then
    pEntInfList := pmsgIn^.V2.EntInfList
    pClientCreds := null
else
    pEntInfList := pmsgIn^.V3.EntInfList
    pClientCreds := pmsgIn^.V3.pClientCreds
endif
/* If explicit credentials are given, use them for access checks. */
if pClientCreds ≠ null then
    err := UseCredsForAccessCheck(pClientCreds^)
    if err ≠ 0 then
        return err
    endif
endif
/* Walk through each item in the EntInfList and perform the requested
 * operation. */
e := pEntInfList
while e ≠ null
    if ENTINF_REMOTE_MODIFY in e^.ulFlags then
        if DSAObj()!msDS-Behavior-Version ≥ DS_BEHAVIOR_LH then
            res := PerformModifyOperation(
                hDrs, e^.Entinf, ADR(infoList[cObjects]))
            if not res then
                return 0
            endif
        else
            /* Not supported (Win2k3 or older DC). */
            SetErrorData(SV_PROBLEM_BUSY,
                0,
                ERROR_INVALID_PARAMETER,
                pmsgOut,
                pdwOutVersion^)
            return 0
        endif
    else
        objCls := ENTINF_GetValue(e^.Entinf, objectClass, prefixTable)
        if objCls = crossRef then
            /* Create or enable a crossRef object. */
            res := CreateCrossRef(hDrs, e^.Entinf, pmsgOut, pdwOutVersion^,
                ADR(infoList[cObjects]))
            if not res then
                return 0
            endif
        else if objCls = nTDSDSA then
            /* Create an nTDSDSA object. */
            res := CreateNtdsDsa(hDrs, e^.Entinf, pEntInfList, pmsgOut,
                pdwOutVersion^, ADR(infoList[cObjects]))
            if not res then
                return 0
            endif
        endif
    end
endwhile

```

```

else
    /* Not supported. */
    SetErrorData(SV_PROBLEM_BUSY, 0, ERROR_INVALID_PARAMETER,
        pmsgOut, pdwOutVersion^)
    return 0
endif
endif
e := e^.pNextEntInf
cObjects := cObjects + 1
endwhile
if pdwOutVersion^ = 2 then
    pmsgOut^.V2.cObjectsAdded := cObjects
    pmsgOut^.V2.infoList := infoList
else
    pmsgOut^.V3.cObjectsAdded := cObjects
    pmsgOut^.V3.infoList := infoList
endif
return 0

```

4.1.2 IDL_DRSAddSidHistory (Opnum 20)

The **IDL_DRSAddSidHistory** method adds one or more SIDs to the [SIDHistory](#) attribute of a given object.

```

ULONG IDL_DRSAddSidHistory(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_ADDSIDREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_ADDSIDREPLY* pmsgOut
);

```

hDrs: RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: Version of the request message. Must be set to 1, because no other version is supported.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message. The value will always be 1, because no other version is supported.

pmsgOut: Pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.2.1 Method-Specific Concrete Types

4.1.2.1.1 DRS_MSG_ADDSIDREQ

The **DRS_MSG_ADDSIDREQ** union defines the request messages that are sent to the [IDL DRSAddSidHistory](#) method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_ADDSIDREQ_V1 V1;
} DRS_MSG_ADDSIDREQ;
```

V1: Version 1 request.

4.1.2.1.2 DRS_MSG_ADDSIDREQ_V1

The **DRS_MSG_ADDSIDREQ_V1** structure defines the request message sent to the [IDL DRSAddSidHistory](#) method.

```
typedef struct {
    DWORD Flags;
    [string] WCHAR* SrcDomain;
    [string] WCHAR* SrcPrincipal;
    [string, ptr] WCHAR* SrcDomainController;
    [range(0,256)] DWORD SrcCredsUserLength;
    [size_is(SrcCredsUserLength)] WCHAR* SrcCredsUser;
    [range(0,256)] DWORD SrcCredsDomainLength;
    [size_is(SrcCredsDomainLength)]
        WCHAR* SrcCredsDomain;
    [range(0,256)] DWORD SrcCredsPasswordLength;
    [size_is(SrcCredsPasswordLength)]
        WCHAR* SrcCredsPassword;
    [string] WCHAR* DstDomain;
    [string] WCHAR* DstPrincipal;
} DRS_MSG_ADDSIDREQ_V1;
```

Flags: A set of zero or more [DRS_ADDSID_FLAGS](#) bit flags.

SrcDomain: Name of the domain to query for the SID of SrcPrincipal. The domain name can be a DNS name or a NetBIOS name.

SrcPrincipal: Name of a security principal (user, computer, or group) in the source domain. This is the source principal, whose SIDs will be added to the destination principal. If Flags does not contain DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ, this name is a domain-relative Security Accounts Manager (SAM) name. Otherwise, it is a DN.

SrcDomainController: Name of the primary domain controller (PDC) (or PDC role owner) in the source domain. The DC name can be a DNS name or a NetBIOS name. If null, the implementation of **IDL_DRSAddSidHistory** will locate such a DC.

SrcCredsUserLength: Count of characters in the SrcCredsUser array.

SrcCredsUser: User name for the credentials to be used in the source domain.

SrcCredsDomainLength: Count of characters in the SrcCredsDomain array.

SrcCredsDomain: Domain name for the credentials to be used in the source domain.

SrcCredsPasswordLength: Count of characters in the SrcCredsPassword array.

SrcCredsPassword: Password for the credentials to be used in the source domain.

DstDomain: Name of the destination domain in which DstPrincipal resides. The domain name can be a DNS name or a NetBIOS name.

DstPrincipal: Name of a security principal (user, computer, or group) in the destination domain. This is the destination principal, to which the source principal's SIDs will be added. If Flags does not contain DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ, this name is a domain-relative SAM name. Otherwise, it is a DN.

4.1.2.1.3 DRS_MSG_ADDSIDREPLY

The **DRS_MSG_ADDSIDREPLY** union defines the response messages received from the [IDL DRSAddSidHistory](#) method. Only one version, identified by `pdwOutVersion^ = 1`, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_ADDSIDREPLY_V1 V1;
} DRS_MSG_ADDSIDREPLY;
```

V1: Version 1 of the reply packet structure.

4.1.2.1.4 DRS_MSG_ADDSIDREPLY_V1

The **DRS_MSG_ADDSIDREPLY_V1** structure defines the response message received from the [IDL DRSAddSidHistory](#) method.

```
typedef struct {
    DWORD dwWin32Error;
} DRS_MSG_ADDSIDREPLY_V1;
```

dwWin32Error: Windows error code.

4.1.2.1.5 DRS_ADDSID_FLAGS

The **DRS_ADDSID_FLAGS** type consists of bit flags that indicate how the SID is to be added to the security principal.

The valid bit flags are shown in the following diagram.

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	C	D	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	E																												
		L																													

X: Unused. MUST be zero and ignored.

CS (DS_ADDSID_FLAG_PRIVATE_CHK_SECURE): If set, the server should verify whether the channel is secure and should return the result of the verification in the response.

DEL (DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ): If set, the server should append the [objectSid](#) and [sIDHistory](#) attributes of SrcPrincipal to the [sIDHistory](#) attribute of DstPrincipal, and should delete SrcPrincipal from the source domain.

This type is declared as follows:

```
typedef DWORD DRS_ADDSID_FLAGS;
```

4.1.2.2 Method-Specific Abstract Types and Procedures

4.1.2.2.1 ConnectionCtx

The ConnectionCtx abstract type represents a connection to a specific server with a given set of credentials. It does not imply any particular protocol or transport. It provides a means for pseudocode to compactly represent the notion of the target server and corresponding credentials for an operation.

Procedures that take a ConnectionCtx as an input perform their operations against the server represented by the ConnectionCtx, using the credentials associated with the ConnectionCtx.

4.1.2.2.2 ConnectToDC

```
procedure ConnectToDC(dcname: unicodestring): ConnectionCtx
```

Creates a [ConnectionCtx](#) for the DC named by dcname, associating the credentials of the client's security context with the ConnectionCtx. dcname may be the DNS name or the NetBIOS name of the DC. If the ConnectionCtx cannot be created, the procedure returns null.

4.1.2.2.3 ConnectToDCWithCreds

```
procedure ConnectToDCWithCreds(
  dcname: unicodestring,
  username: unicodestring,
  pwd: unicodestring,
  domain: unicodestring): ConnectionCtx
```

Creates a [ConnectionCtx](#) for the DC named by *dcname*, associating the credentials of user *username*, password *pwd*, and user-domain *domain* with the ConnectionCtx. *dcname* may be the DNS name or the NetBIOS name of the DC. If the ConnectionCtx cannot be created, it returns null.

4.1.2.2.4 GenerateFailureAudit

```
procedure GenerateFailureAudit()
```

Generates a failure audit event on the server on which it is called if auditing is enabled. The generated audit event indicates that an operation failed. Does nothing if auditing is not enabled. The content of the audit event is an implementation-specific behavior.

4.1.2.2.5 GenerateSuccessAudit

```
procedure GenerateSuccessAudit()
```

Generates a success audit event on the server on which it is called if auditing is enabled. The generated audit event indicates that an operation succeeded. Does nothing if auditing is not enabled. The content of the audit event is an implementation-specific behavior.

4.1.2.2.6 GenerateSuccessAuditRemotely

```
procedure GenerateSuccessAuditRemotely(ctx: ConnectionCtx): boolean
```

If auditing is enabled on the server associated with *ctx*, the *GenerateSuccessAuditRemotely* procedure generates a success audit event on that server and returns true. The generated audit event indicates that an operation succeeded. Returns false if auditing is not enabled on that server. The content of the audit event is an implementation-specific behavior. [<6>](#)

4.1.2.2.7 GetKeyLength

```
procedure GetKeyLength(hDrs: DRS_HANDLE): integer
```

Returns the key length, in bits, of the encryption used on the *hDrs* connection. Returns 0 if no encryption is in use on the connection.

4.1.2.2.8 GetPDC

```
procedure GetPDC(domainName: unicodestring): unicodestring
```

Returns the DNS name of the DC that holds the PDC FSMO role for the domain whose name is *domainName*, or null if such a DC cannot be found. *domainName* can be either the DNS name or the NetBIOS name of the domain.

4.1.2.2.9 HasAdminRights

```
procedure HasAdminRights(ctx: ConnectionCtx) : boolean
```

Returns true if the credentials associated with ctx have administrative rights on the DC associated with ctx. Possessing administrative rights is defined as having the ability to write to (that is, change the membership of) the Domain Admins group in the domain that is the default domain NC on the DC associated with ctx.

4.1.2.2.10 IsAuditingEnabledForNc

```
procedure IsAuditingEnabledForNc(nc: DSName): boolean
```

Returns true if auditing on the server on which it is called is enabled for the NC replica of the NC whose name is nc, and returns false otherwise.

4.1.2.2.11 IsLocalRpcCall

```
procedure IsLocalRpcCall(hDrs: DRS_HANDLE): boolean
```

Returns true if the RPC call that is being processed on hDrs originated from the same computer as the computer that is processing the call.

4.1.2.2.12 IsNT4SP4OrBetter

```
procedure IsNT4SP4OrBetter(ctx: ConnectionCtx): boolean
```

If the DC named in ctx is running Windows NT 4.0 and is not running at least Windows NT 4.0 SP4, this procedure returns false. Otherwise, it returns true. [<7>](#)

4.1.2.2.13 IsWellKnownDomainRelativeSid

```
procedure IsWellKnownDomainRelativeSid(sid: SID): boolean
```

Returns true if sid consists of the domain SID of the server's default domain and of a RID whose value is less than 1000, and returns false otherwise.

4.1.2.2.14 LastRID

```
procedure LastRID(sid: SID): Rid
```

Extracts and returns the RID from the SID sid. See [\[MS-DTYP\]](#) section **2.4.2**.

4.1.2.2.15 RemoteQuery

```
procedure RemoteQuery(  
  ctx: ConnectionCtx,  
  query: unicodestring): select-return-value
```

Performs the select statement represented by the string query against the server associated with ctx, using the credentials associated with ctx. Returns the results of the select operation. The return value of this function is the same type as the return value of the select statement performed.

4.1.2.3 Server Behavior of the IDL_DRSAddSidHistory Method

Informative summary of behavior: The IDL_DRSAddSidHistory method adds the SIDs associated with one principal (the source principal) to the [sIDHistory](#) attribute of another principal (the destination principal). The source principal's [objectSid](#) and any SIDs in the source principal's [sIDHistory](#) are added to the destination principal's [sIDHistory](#). This method is called on a DC whose default NC contains the destination principal. If necessary, the destination DC will contact a DC whose default NC contains the source principal as part of executing this method.

This method has three different variants on this behavior, and the caller indicates which variant is desired by specifying a combination of flags in pmsgIn^.V1.flags.

If the DS_ADDSID_FLAG_PRIVATE_CHK_SECURE flag is specified, the first variant is selected. In this variant, the method verifies only that the RPC call is secure. It does not perform any further processing or manipulate the [sIDHistory](#) attribute of any object, regardless of other flags that may be present.

If DS_ADDSID_FLAG_PRIVATE_CHK_SECURE is not specified but DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ is specified, the second variant is selected. In this variant, the source and destination principals are in the same domain. The values of the [objectSid](#) and [sIDHistory](#) attributes of the source principal are added to the destination principal's [sIDHistory](#) attribute, and then the source principal is deleted. Loosely speaking, the destination principal adopts the source principal as an "alias" and the source principal disappears.

The third variant is selected by specifying neither DS_ADDSID_FLAG_PRIVATE_CHK_SECURE nor DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ. In this variant, the source and destination principals are in different forests. The values of the source principal's [objectSid](#) and [sIDHistory](#) attributes are copied into the destination principal's [sIDHistory](#) attribute, as in the second variant, but without deleting the source principal. Loosely speaking, the destination principal adopts the source principal as an "alias" while coexisting with the source principal.

```
ULONG
IDL_DRSAddSidHistory(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_ADDSIDREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)] DRS_MSG_ADDSIDREPLY *pmsgOut)

flags: DRS_ADDSID_FLAGS
srcPrinc: DSName
dstPrinc: DSName
srcPrincInDst: DSName
srcNc: DSName
dstNc: DSName
crSrc: DSName
crDst: DSName
partCtr: DSName
srcDomainController: unicodestring
srcCtx: ConnectionCtx
srcPrincSid: SID
srcPrincSidHistory: set of SID

if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
```



```

if AmIRODC() then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_OBJ_NOT_FOUND
    return ERROR_DS_OBJ_NOT_FOUND
endif

pdwOutVersion^ := 1
pmsgOut^.V1.dwWin32Error := 0

flags := pmsgIn^.V1.flags
if DS_ADDSID_FLAG_PRIVATE_CHK_SECURE in flags then
    /* First mode of operation: verify connection security.
     * If connecting from off-machine, connection must have 128-bit
     * encryption or better. */
    if (not IsLocalRpcCall(hDrs)) and
        (GetKeyLength(hDrs) < 128) then
        pmsgOut^.V1.dwWin32Error := ERROR_DS_MUST_RUN_ON_DST_DC
        return ERROR_DS_MUST_RUN_ON_DST_DC
    else
        return 0
    endif
endif

if DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ in flags then
    /* Second mode of operation: add objectSid/sidHistory from source
     * principal to destination principal, then delete source
     * principal. */

    /* Basic parameter validation */
    if (pmsgIn^.V1.SrcDomain # null) or
        (pmsgIn^.V1.DstDomain # null) or
        (pmsgIn^.V1.SrcCredsUserLength # 0) or
        (pmsgIn^.V1.SrcCredsDomainLength # 0) or
        (pmsgIn^.V1.SrcCredsPasswordLength # 0) or
        (pmsgIn^.V1.SrcDomainController = "") or
        (pmsgIn^.V1.SrcPrincipal = null) or
        (pmsgIn^.V1.SrcPrincipal = "") or
        (pmsgIn^.V1.DstPrincipal = null) or
        (pmsgIn^.V1.DstPrincipal = "") then
        pmsgOut^.V1.dwWin32Error := ERROR_INVALID_PARAMETER
        return ERROR_INVALID_PARAMETER
    endif

    /* In this case, pmsgIn^.V1.SrcPrincipal and .DstPrincipal are
     * DNSs. */
    srcPrinc := GetDSName(pmsgIn^.V1.SrcPrincipal)
    dstPrinc := GetDSName(pmsgIn^.V1.DstPrincipal)
    srcNc := GetObjectNC(srcPrinc)
    dstNc := GetObjectNC(dstPrinc)

    /* Source and destination principals must be in same domain. */
    if srcNc # dstNc then
        pmsgOut^.V1.dwWin32Error := ERROR_INVALID_PARAMETER
        return ERROR_INVALID_PARAMETER
    endif

    /* Destination NC must be this server's default domain NC. */
    if dstNc # DefaultNC() then
        pmsgOut^.V1.dwWin32Error := ERROR_DS_MASTERDSA_REQUIRED
    endif
endif

```

```

    return ERROR_DS_MASTERDSA_REQUIRED
endif

/* Verify this server has auditing enabled for destination
 * domain.*/
if not IsAuditingEnabledForNc(dstNc) then
    pmsgOut^.V1.dwWin32Error :=
        ERROR_DS_DESTINATION_AUDITING_NOT_ENABLED
    return ERROR_DS_DESTINATION_AUDITING_NOT_ENABLED
endif

/* Must have the control access right. */
if not AccessCheckCAR(dstNc, Migrate-SID-History) then
    GenerateFailureAudit()
    pmsgOut^.V1.dwWin32Error := ERROR_DS_INSUFF_ACCESS_RIGHTS
    return ERROR_DS_INSUFF_ACCESS_RIGHTS
endif

/* Destination domain must be in native mode. */
partCtr := DescendantObject(ConfigNC(), "CN=Partitions,")
if partCtr ≠ null
    crDst := select one dd from subtree partCtr where
        (crossRef in dd!objectClass and
         dd!nCName = dstNc)
endif
if partCtr = null or crDst = null then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_INTERNAL_FAILURE
    return ERROR_DS_INTERNAL_FAILURE
else
    if crDst!nTMixedDomain = 1 then
        pmsgOut^.V1.dwWin32Error := ERROR_DS_DST_DOMAIN_NOT_NATIVE
        return ERROR_DS_DST_DOMAIN_NOT_NATIVE
    endif
endif

/* Validation of object state. */
if (not ObjExists(srcPrinc)) or
    (not (user in srcPrinc!objectClass or
         group in srcPrinc!objectClass)) or
    (not ObjExists(dstPrinc)) or
    (not (user in dstPrinc!objectClass or
         group in dstPrinc!objectClass)) or
    (srcPrinc = dstPrinc) or
    (IsWellKnownDomainRelativeSid(srcPrinc!objectSid)) or
    (IsWellKnownDomainRelativeSid(dstPrinc!objectSid)) then
    pmsgOut^.V1.dwWin32Error := ERROR_INVALID_PARAMETER
    return ERROR_INVALID_PARAMETER
endif

/* Check that we have rights to delete the source principal. */
if (not AccessCheckObject(srcPrinc, RIGHT_DS_DELETE)) and
    (not AccessCheckObject(srcPrinc.parent, RIGHT_DS_DELETE_CHILD))
    then
        pmsgOut^.V1.dwWin32Error := ERROR_ACCESS_DENIED
        return ERROR_ACCESS_DENIED
    endif

/* Add source principal's objectSid and sidHistory to

```

```

    * destination principal's sidHistory. */
    dstPrinc!sidHistory := dstPrinc!sidHistory + {srcPrinc!objectSid}
    dstPrinc!sidHistory := dstPrinc!sidHistory + srcPrinc!sidHistory

    RemoveObj(srcPrinc)
    GenerateSuccessAudit()
    return 0
endif

/* Third mode of operation: add objectSid/sidHistory from source
 * principal to destination principal. Source principal is
 * untouched. */

/* Basic parameter validation. */
if (pmsgIn^.V1.SrcDomain = null) or
    (pmsgIn^.V1.SrcDomain = "") or
    (pmsgIn^.V1.DstDomain = null) or
    (pmsgIn^.V1.DstDomain = "") or
    (pmsgIn^.V1.SrcCredsUserLength > 0 and
        pmsgIn^.V1.SrcCredsUser = null) or
    (pmsgIn^.V1.SrcCredsDomainLength > 0 and
        pmsgIn^.V1.SrcCredsDomain = null) or
    (pmsgIn^.V1.SrcCredsPasswordLength > 0 and
        pmsgIn^.V1.SrcCredsPassword = null) or
    (pmsgIn^.V1.SrcDomainController = "") or
    (pmsgIn^.V1.SrcPrincipal = null) or
    (pmsgIn^.V1.SrcPrincipal = "") or
    (pmsgIn^.V1.DstPrincipal = null) or
    (pmsgIn^.V1.DstPrincipal = "") then
    pmsgOut^.V1.dwWin32Error := ERROR_INVALID_PARAMETER
    return ERROR_INVALID_PARAMETER
endif

/* Confirm destination domain is in forest of server. */
crDst := select one dd from subtree ConfigNC() where
    (crossRef in dd!objectClass and
        (dd!dNSHostName = pmsgIn^.V1.DstDomain or
            dd!nETBIOName = pmsgIn^.V1.DstDomain))
if crDst = null then
    pmsgOut^.V1.dwWin32Error :=
        ERROR_DS_DESTINATION_DOMAIN_NOT_IN_FOREST
    return ERROR_DS_DESTINATION_DOMAIN_NOT_IN_FOREST
endif

/* Confirm source domain is not in forest of server. */
crSrc := select one ss from subtree ConfigNC() where
    (crossRef in ss!objectClass and
        (ss!dNSHostName = pmsgIn^.V1.SrcDomain or
            ss!nETBIOName = pmsgIn^.V1.SrcDomain))
if crSrc ≠ null then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_SOURCE_DOMAIN_IN_FOREST
    return ERROR_DS_SOURCE_DOMAIN_IN_FOREST
endif

/* Destination NC must be this server's default domain NC. */
if crDst!nCName ≠ DefaultNC() then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_MASTERDSA_REQUIRED
    return ERROR_DS_MASTERDSA_REQUIRED

```

```

endif

/* Destination domain must be in native mode. */
if crDst!nTMixedDomain = 1 then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_DST_DOMAIN_NOT_NATIVE
    return ERROR_DS_DST_DOMAIN_NOT_NATIVE
endif

srcNC := GetDSNameFromDN(pmsgIn^.V1.SrcDomain)
dstNC := GetDSNameFromDN(pmsgIn^.V1.DstDomain)

/* Verify this server has auditing enabled for destination domain. */
if not IsAuditingEnabledForNc(dstNC) then
    pmsgOut^.V1.dwWin32Error :=
        ERROR_DS_DESTINATION_AUDITING_NOT_ENABLED
    return ERROR_DS_DESTINATION_AUDITING_NOT_ENABLED
endif

/* Must have the control access right. */
if not AccessCheckCAR(dstNc, Migrate-SID-History) then
    GenerateFailureAudit()
    pmsgOut^.V1.dwWin32Error := ERROR_DS_INSUFF_ACCESS_RIGHTS
    return ERROR_DS_INSUFF_ACCESS_RIGHTS
endif

/* Retrieve destination principal.
 * In this case, pmsgIn^.V1.DstPrincipal is a SAM name. */
dstPrinc := select one o from subtree DefaultNC() where
    (o!sAMAccountName = pmsgIn^.V1.DstPrincipal)
if dstPrinc = null then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_OBJ_NOT_FOUND
    return ERROR_DS_OBJ_NOT_FOUND
endif

/* Locate a source DC if one wasn't supplied. Source DC must be
 * the PDC FSMO role owner. */
srcDomainController := pMsgin^.V1.SrcDomainController
if srcDomainController = null then
    srcDomainController := GetPDC(pmsgIn^.V1.SrcDomain)
else
    if srcDomainController ≠ GetPDC(pmsgIn^.V1.SrcDomain) then
        pmsgOut^.V1.dwWin32Error := ERROR_INVALID_DOMAIN_ROLE
        return ERROR_INVALID_DOMAIN_ROLE
    endif
endif
endif
if srcDomainController = null then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_CANT_FIND_DC_FOR_SRC_DOMAIN
    return ERROR_DS_CANT_FIND_DC_FOR_SRC_DOMAIN
endif

/* Connect to source DC, using supplied credentials if applicable. */
if (pmsgIn^.V1.SrcCredsUserLength = 0) and
    (pmsgIn^.V1.SrcCredsPasswordLength = 0) and
    (pmsgIn^.V1.SrcCredsDomainLength = 0) then
    srcCtx := ConnectToDC(srcDomainController)
else
    srcCxt := ConnectToDCWithCreds(srcDomainController,
        pmsgIn^.V1.SrcCredsUser, pmsgIn^.V1.SrcCredsPassword,

```

```

        pmsgIn^.V1.SrcCredsDomain)
endif

if (srcCtx = null) then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_CANT_FIND_DC_FOR_SRC_DOMAIN
    return ERROR_DS_CANT_FIND_DC_FOR_SRC_DOMAIN
endif

/* Confirm client has administrative rights on source DC. */
if not HasAdminRights(srcCtx) then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_INSUFF_ACCESS_RIGHTS
    return ERROR_DS_INSUFF_ACCESS_RIGHTS
endif

/* Confirm source domain has auditing enabled and generate an audit
* event on it. */
if not GenerateSuccessAuditRemotely(srcCtx)
    pmsgOut^.V1.dwWin32Error := ERROR_DS_SOURCE_AUDITING_NOT_ENABLED
    return ERROR_DS_SOURCE_AUDITING_NOT_ENABLED
endif

/* Verify that if source domain is running Windows NT 4.0, it is
* running at least Service Pack 4 of that operating system. */
if not IsNT4SP4OrBetter(srcCtx)
    pmsgOut^.V1.dwWin32Error := ERROR_DS_SRC_DC_MUST_BE_SP4_OR_GREATER
    return ERROR_DS_SRC_DC_MUST_BE_SP4_OR_GREATER
endif

/* Retrieve source principal from source DC.
* In this case, pmsgIn^.V1.SrcPrincipal is a SAM name. */
srcPrinc := RemoteQuery(srcCtx, "select one o
    from subtree DefaultNCOfDC(srcDomainController)
    where (o!sAMAccountName = pmsgIn^.V1.SrcPrincipal)")
if srcPrinc = null then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_OBJ_NOT_FOUND
    return ERROR_DS_OBJ_NOT_FOUND
endif

/* Source principal must be user (which includes computer) or
* group.*/
if not (group in srcPrinc!objectClass or
    user in srcPrinc!objectClass) then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_SRC_OBJ_NOT_GROUP_OR_USER
    return ERROR_DS_SRC_OBJ_NOT_GROUP_OR_USER
endif

srcPrincSid := srcPrinc!objectSid
srcPrincSidHistory := srcPrinc!sIDHistory

/* Source and destination principals must both be computer, or both
* be user, or both be group. The order is important: although
* computer objects are user objects, we disallow the case where
* one principal is a computer and the other principal is a user
* but not a computer. */
if ((computer in srcPrinc!objectClass and
    not computer in dstPrinc!objectClass) or
    (computer in dstPrinc!objectClass and
    not computer in srcPrinc!objectClass)) or

```

```

        ((user in srcPrinc!objectClass and
         not user in dstPrinc!objectClass) or
         (user in dstPrinc!objectClass and
         not user in srcPrinc!objectClass)) or
        ((group in srcPrinc!objectClass and
         not group in dstPrinc!objectClass) or
         (group in dstPrinc!objectClass and
         not group in srcPrinc!objectClass)) then
    pmsgOut^.V1.dwWin32Error :=
        ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
    return ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
endif

/* Class-specific object state tests.
 * Note that computer is a subclass of user, so the following test
 * applies to both user and computer objects. */
if user in srcPrinc!objectClass then
    if srcPrinc!userAccountControl ∩ {ADS_UF_NORMAL_ACCOUNT,
                                     ADS_UF_WORKSTATION_TRUST_ACCOUNT,
                                     ADS_UF_SERVER_TRUST_ACCOUNT} ≠
        dstPrinc!userAccountControl ∩ {ADS_UF_NORMAL_ACCOUNT,
                                     ADS_UF_WORKSTATION_TRUST_ACCOUNT,
                                     ADS_UF_SERVER_TRUST_ACCOUNT} then
        pmsgOut^.V1.dwWin32Error :=
            ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
        return ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
    endif
endif

if group in srcPrinc!objectClass and
   srcPrinc!groupType ≠ dstPrinc!groupType then
    pmsgOut^.V1.dwWin32Error :=
        ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
    return ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
endif

/* Check if source principal is built-in principal. */
if IsBuiltinPrincipal(srcPrinc!objectSid) then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_UNWILLING_TO_PERFORM
    return ERROR_DS_UNWILLING_TO_PERFORM
endif

/* If source principal has well-known domain-relative SID
 * make sure final RIDs of source and destination principals
 * are the same. */
if IsWellKnownDomainRelativeSid(srcPrinc!objectSid) then
    if LastRID(srcPrinc!objectSid) ≠ LastRID(dstPrinc!objectSid)
        pmsgOut^.V1.dwWin32Error := ERROR_DS_UNWILLING_TO_PERFORM
        return ERROR_DS_UNWILLING_TO_PERFORM
    endif
endif

/* Make sure a principal with the SID of the source principal
 * does not already exist in destination domain. */
srcPrincInDst := select one o from subtree DefaultNC() where
    (o ≠ dstPrinc) and
    ((o!objectSid = srcPrincSid) or
     (o!objectSid in srcPrincSidHistory) or
     (srcPrincSid in o!SIDHistory)) or

```

```

        ((srcPrincSidHistory ∩ o!sIDHistory) ≠ {}))
if srcPrincInDst ≠ null then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_SRC_SID_ALREADY_IN_FOREST
    return ERROR_DS_SRC_SID_ALREADY_IN_FOREST
endif

/* Add source principal's objectSid and sIDHistory to
 * destination principal's sidHistory. */
dstPrinc!sIDHistory := dstPrinc!sIDHistory + {srcPrincSid}
dstPrinc!sIDHistory := dstPrinc!sIDHistory + srcPrincSidHistory
GenerateSuccessAudit()
return 0

```

4.1.3 IDL_DRSBind (Opnum 0)

The **IDL_DRSBind** method creates a context handle that is necessary to call any other method in this interface.

```

ULONG IDL_DRSBind(
    [in] handle_t rpc_handle,
    [in, unique] UUID* puuidClientDsa,
    [in, unique] DRS_EXTENSIONS* pextClient,
    [out] DRS_EXTENSIONS** ppextServer,
    [out, ref] DRS_HANDLE* phDrs
);

```

rpc_handle: An RPC binding handle, as specified in [\[C706\]](#).

puuidClientDsa: A pointer to an implementation-specific identity of the caller.

pextClient: A pointer to client capabilities, for use in version negotiation.

ppextServer: A pointer to a pointer to server capabilities, for use in version negotiation.

phDrs: A pointer to an RPC context handle (as specified in [\[C706\]](#)), which may be used in calls to other methods in this interface.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.3.1 Client Behavior When Sending the IDL_DRSBind Request

The client uses puuidClientDsa to pass a unique identifier. The client freely determines puuidClientDsa; the server places no constraints on it. This identifier is intended for diagnostic purposes only. [<8>](#)

The client uses pextClient to pass a properly initialized [DRS_EXTENSIONS_INT](#) structure to the server. The client reads the value of [msDS-ReplicationEpoch](#) from its [nTDSDSA](#) object, and assigns this value to the dwReplEpoch field of the [DRS_EXTENSIONS_INT](#) structure.

The remaining information in the [DRS_EXTENSIONS_INT](#) structure must be consistent with the client's capabilities. This information affects the versions of response structures that the server returns in method calls using the [DRS_HANDLE](#) returned by IDL_DRSBind. In descriptions of method calls that use a [DRS_HANDLE](#), this handle is sometimes called the client's RPC context. [<9>](#)

If a method of this protocol takes a parameter named `dwInVersion`, the client uses that parameter to specify the version of the referent of the next parameter to that method, often named `pmsgIn`. The referent of this parameter is called the method's request. The `dwInVersion` parameter is called the request version. For example, if the client passes `dwInVersion = 7` to `IDL_DRSGetNCChanges`, the client also passes a `DRS_MSG_GETCHGREQ_V7` request.

If a method of this protocol takes an integer parameter named `pdwOutVersion`, the server uses that parameter to return the version number of the referent of the next parameter to that method, often named `pmsgOut`. The referent of this parameter is called the method's response. The referent of `pdwOutVersion` is called the response version. For example, when the server returns `pdwOutVersion^ = 6` from `IDL_DRSGetNCChanges`, the server also returns a `DRS_MSG_GETCHGREPLY_V6` response.

Most methods in this protocol are capable of generating only a certain response version from a certain request version. The following special cases apply:

- `IDL_DRSGetNCChanges` is capable of returning a version 6 response from both a version 7 and a version 8 request. However, the `DRS_EXT_GETCHGREPLY_V6` bit must be set in the client's RPC context for the server to generate a version 6 response. Otherwise, the server returns `ERROR_REVISION_MISMATCH`. Note that whenever `IDL_DRSGetNCChanges` is capable of returning a version 6 response, it is also capable of returning a version 7 response, which is a compressed form of a version 6 response. Compression of `IDL_DRSGetNCChanges` responses is not controlled by the state of the client's RPC context; it is controlled on a per-request basis by the client; see `DRS_USE_COMPRESSION` in section [5.37](#).
- `IDL_DRSSAddEntry` can generate either a version 2 or version 3 response from either a version 2 or version 3 request. The server generates a version 3 response when `DRS_EXT_ADDENTRYREPLY_V3` is set in the client's RPC context; otherwise, the server generates a version 2 response.
- `IDL_DRSDomainControllerInfo` has only one request version; it contains an `InfoLevel` field. The `InfoLevel`, not the `dwInputVersion`, determines the response version. Similarly, `IDL_DRSGetReplInfo` has two request versions, which both contain an `InfoType` field. The `InfoType`, not the `dwInputVersion`, determines the response version.

The following tables describe how the server determines the response version based on the request version, the [DRS_EXTENSIONS_INT](#) structure specified when creating the [DRS_HANDLE](#), and in some cases, the contents of the request message. The tables show all the valid combinations. All cases not shown in the tables below are a client error; the server returns `ERROR_INVALID_PARAMETER`, except for the error return already described in case 1 above.

`IDL_DRSReplicaSync`

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	-

`IDL_DRSGetNCChanges`

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
4	-	1
5	-	1
7	<code>DRS_EXT_GETCHGREPLY_V6</code>	6

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
8	DRS_EXT_GETCHGREPLY_V6	6

IDL_DRSUpdateRefs

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	-

IDL_DRSReplicaAdd

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	-
2	-	-

IDL_DRSReplicaDel

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	-

IDL_DRSReplicaModify

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	-

IDL_DRSVerifyNames

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSGetMemberships

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSInterDomainMove

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
2	-	2

IDL_DRSGetNT4ChangeLog

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSCrackNames

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSTransferSPN

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSTransferDomainControllerInfo

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSTransferDomainControllerInfo

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSTransferDomainControllerInfo

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	request.InfoLevel ¹

IDL_DRSTransferDomainControllerInfo

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
2	-	2
3	-	2
2	DRS_EXT_ADDENTRYREPLY_V3	3
3	DRS_EXT_ADDENTRYREPLY_V3	3

IDL_DRSTransferDomainControllerInfo

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSTransferDomainControllerInfo

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	request.InfoType ²
2	-	request.InfoType ²

IDL_DRSSAddSidHistory

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSGetMemberships2

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSReplicaVerifyObjects

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	-

IDL_DRSGetObjectExistence

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DRSQuerySitesByCost

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DSAPrepareScript

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

IDL_DSASExecuteScript

Request version	DRS_EXTENSIONS bit set in client's RPC context	Response version
1	-	1

¹ Possible values are 0x1, 0x2, and 0xffffffff (see section [4.1.5](#)).

² Possible values are detailed in section [4.1.13](#).

4.1.3.2 Server Behavior of the IDL_DRSBind Method

The server retains the [DRS_EXTENSIONS_INT](#) structure passed as pextClient^.

The server sets ppextServer to a [DRS_EXTENSIONS_INT](#) structure whose dwReplEpoch field is initialized as described in the previous section ([Client Behavior When Sending the IDL_DRSBind Request \(section 4.1.3.1\)](#)), and whose other fields describe the server. The server then returns a [DRS_HANDLE](#) as the referent of phDr.

The following tables specify the capability assertions made by a server that sets bits in the [DRS_EXTENSIONS_INT](#) structure returned from [IDL_DRSBind](#). Each row of a table gives a request version (including both dwInVersion and the InfoLevel of [IDL_DRSDomainControllerInfo](#) and the InfoType of [IDL_DRSGetReplInfo](#)) and the [DRS_EXTENSIONS_INT](#) bit or bits that the server sets to indicate support for that request. For instance, every server supports a version 1 request to [IDL_DRSReplicaSync](#), but a server does not support a version 5 request to [IDL_DRSGetNCChanges](#) unless it has set both the DRS_EXT_GETCHGREQ_V5 and DRS_EXT_RESTORE_USN_OPTIMIZATION bits.

A server supports version 4 and version 7 requests to **IDL_DRSGetNCChanges** only via the SMTP replication transport (see [\[MS-SRPL\]](#)). These cases are noted in the relevant table. A server supports all other requests only via the RPC transport.

IDL_DRSReplicaSync

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

IDL_DRSGetNCChanges

Request version	DRS_EXTENSIONS_INT bit(s)
4	SMTP replication transport
5	DRS_EXT_GETCHGREQ_V5 DRS_EXT_RESTORE_USN_OPTIMIZATION
7	SMTP replication transport
8	DRS_EXT_GETCHGREQ_V8 DRS_EXT_RESTORE_USN_OPTIMIZATION

[IDL_DRSUpdateRefs](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[IDL_DRSReplicaAdd](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-
2	DRS_EXT_ASYNCREPL

[IDL_DRSReplicaDel](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[IDL_DRSReplicaModify](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[IDL_DRSVerifyNames](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[IDL_DRSGetMemberships](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[IDL_DRSInterDomainMove](#)

Request version	DRS_EXTENSIONS_INT bit(s)
2	DRS_EXT_MOVEREQ_V2

[IDL_DRSGetNT4ChangeLog](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[IDL_DRSCrackNames](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[IDL_DRSWriteSPN](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[IDL_DRSRemoveDsServer](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_REMOVEAPI

[IDL_DRSRemoveDsDomain](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_REMOVEAPI

IDL_DRSDomainControllerInfo

Request version	DRS_EXTENSIONS_INT bit(s)
1 InfoLevel = 0x1	DRS_EXT_DCINFO_V1
1 InfoLevel = 0x2	DRS_EXT_DCINFO_V2
1 InfoLevel = 0x3	DRS_EXT_LH_BETA2
1 InfoLevel = 0xffffffff	DRS_EXT_DCINFO_VFFFFFFFF

[IDL_DRSAddEntry](#)

Request version	DRS_EXTENSIONS_INT bit(s)
2	DRS_EXT_ADDENTRY_V2
3	DRS_EXT_NONDOMAIN_NCS

[IDL_DRSExecuteKCC](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_KCC_EXECUTE

IDL_DRSGetReplInfo

Request version	DRS_EXTENSIONS_INT bit(s)
1	-
2	DRS_EXT_GETCHGREQ_V8
2 InfoType = [3..5]	DRS_EXT_POST_BETA3
2 InfoType = 6	DRS_EXT_GETCHGREQ_V8

Request version	DRS_EXTENSIONS_INT bit(s)
2 InfoType = [7..10]	DRS_EXT_GETCHGREPLY_V6

[IDL_DRSAddSidHistory](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_ADD_SID_HISTORY

[IDL_DRSGetMemberships2](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_GETMEMBERSHIPS2

[IDL_DRSReplicaVerifyObjects](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_VERIFY_OBJECT

[IDL_DRSGetObjectExistence](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_VERIFY_OBJECT

[IDL_DRSQuerySitesByCost](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_QUERYSITESBYCOST_V1

[IDL_DRSInitDemotion](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_ADAM

[IDL_DRSReplicaDemotion](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_ADAM

[IDL_DRSFinishDemotion](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_ADAM

[IDL_DSAPrepareScript](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[IDL_DSAScriptExecute](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

4.1.3.3 Client Behavior When Receiving the IDL_DRSBind Response

The client receives a [DRS_EXTENSIONS_INT](#) structure from the server as the referent of ppextServer.

A server supports only a subset of the possible request versions, including both dwInVersion and the InfoLevel of [IDL_DRSDomainControllerInfo](#) and the InfoType of [IDL_DRSGetReplInfo](#). The server informs the client of its capabilities via the [DRS_EXTENSIONS_INT](#) structure returned from [IDL_DRSBind](#), as described in [Server Behavior of the IDL_DRSBind Method \(section 4.1.3.2\)](#).

The client receives a [DRS_HANDLE](#) as the referent of phDrs.

The client retains the context handle phDrs^ for use in method calls on the [drsuapi](#) interface. The handle remains valid until either the server unilaterally breaks the RPC connection (for example, by crashing) or until [IDL_DRSUnbind](#) has been performed.

4.1.3.4 Examples of the IDL_DRSBind Method

The LDAP Server on DC2.CONTOSO.COM is binding to the directory server DC1.CONTOSO.COM.

4.1.3.4.1 Initial State

Querying the [nTDSDSA](#) objects for the root domain NC DC=CONTOSO, DC=COM for DC1 and DC2 respectively:

- ldap_search_s("CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com", baseObject, "(objectClass=*)", [objectClass, cn, distinguishedName, objectGUID, msDS-Behavior-Version])
- Result <0>: (null)
- Matched DN's:
- Getting 1 entries:
- >> Dn: CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
- 3> objectClass: top; applicationSettings; nTDSDSA;
- 1> cn: NTDS Settings;
- 1> distinguishedName: CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com;


```

1> objectGUID: c20bc312-4d35-4cc0-9903-b1073368af4a;

1> msDS-Behavior-Version: 2 = (DS_BEHAVIOR_WIN2003);

▪ ldap_search_s("CN=NTDS Settings,CN=DC2,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com", baseObject, "(objectClass=*)", [objectClass, cn, distinguishedName, objectGUID, msDS-Behavior-Version])

▪ Result <0>: (null)

▪ Matched DNs:

▪ Getting 1 entries:

▪ >> Dn: CN=NTDS Settings,CN=DC2,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com

3> objectClass: top; applicationSettings; nTDSDSA;

1> cn: NTDS Settings;

1> distinguishedName: CN=NTDS Settings, CN=DC2, CN=Servers, CN=Default-First-Site-Name, CN=Sites, CN=Configuration, DC=contoso, DC=com;

1> objectGUID: 6aad8f5a-07cc-403a-9696-9102fe1c320b;

1> msDS-Behavior-Version: 2 = (DS_BEHAVIOR_WIN2003)

```

4.1.3.4.2 Client Request

DC2 invokes the IDL_DRSBind method against DC1, with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- puuidClientDsa = GUID {6aad8f5a-07cc-403a-9696-9102fe1c320b}
- pextClient:
 - cb: 0x1c
 - dwFlags:
 - DRS_EXT_BASE
 - DRS_EXT_ASYNCREPL
 - DRS_EXT_REMOVEAPI
 - DRS_EXT_MOVEREQ_V2
 - DRS_EXT_GETCHG_DEFLATE
 - DRS_EXT_DCINFO_V1
 - DRS_EXT_RESTORE_USN_OPTIMIZATION
 - DRS_EXT_KCC_EXECUTE
 - DRS_EXT_ADDENTRY_V2

- DRS_EXT_LINKED_VALUE_REPLICATION
- DRS_EXT_DCINFO_V2
- DRS_EXT_INSTANCE_TYPE_NOT_REQ_ON_MOD
- DRS_EXT_CRYPTO_BIND
- DRS_EXT_GET_REPL_INFO
- DRS_EXT_STRONG_ENCRYPTION
- DRS_EXT_DCINFO_VFFFFFFFF
- DRS_EXT_TRANSITIVE_MEMBERSHIP
- DRS_EXT_ADD_SID_HISTORY
- DRS_EXT_POST_BETA3
- DRS_EXT_GET_MEMBERSHIPS2
- DRS_EXT_GETCHGREQ_V6
- DRS_EXT_NONDOMAIN_NCS
- DRS_EXT_GETCHGREQ_V8
- DRS_EXT_GETCHGREPLY_V5
- DRS_EXT_GETCHGREPLY_V6
- DRS_EXT_WHISTLER_BETA3
- DRS_EXT_W2K3_DEFLATE
- SiteObjGuid: GUID {620954c7-7044-400f-9c0b-5c9154198aa6}
- Pid: 632
- dwReplEpoch:0

4.1.3.4.3 Server Response

Return code of 0 ([DRS_HANDLE](#) to DC1 omitted) with the following values:

- ppextServer:
 - cb: 0x1c
 - dwFlags:
 - DRS_EXT_BASE
 - DRS_EXT_ASYNCREPL
 - DRS_EXT_REMOVEAPI
 - DRS_EXT_MOVEREQ_V2

- DRS_EXT_GETCHG_DEFLATE
- DRS_EXT_DCINFO_V1
- DRS_EXT_RESTORE_USN_OPTIMIZATION
- DRS_EXT_KCC_EXECUTE
- DRS_EXT_ADDENTRY_V2
- DRS_EXT_LINKED_VALUE_REPLICATION
- DRS_EXT_DCINFO_V2
- DRS_EXT_INSTANCE_TYPE_NOT_REQ_ON_MOD
- DRS_EXT_GET_REPL_INFO
- DRS_EXT_STRONG_ENCRYPTION
- DRS_EXT_DCINFO_VFFFFFFFF
- DRS_EXT_TRANSITIVE_MEMBERSHIP
- DRS_EXT_ADD_SID_HISTORY
- DRS_EXT_POST_BETA3
- DRS_EXT_GETCHGREQ_V5
- DRS_EXT_GET_MEMBERSHIPS2
- DRS_EXT_GETCHGREQ_V6
- DRS_EXT_NONDOMAIN_NCS
- DRS_EXT_GETCHGREQ_V8
- DRS_EXT_GETCHGREPLY_V5
- DRS_EXT_GETCHGREPLY_V6
- DRS_EXT_WHISTLER_BETA3
- DRS_EXT_W2K3_DEFLATE
- SiteObjGuid: GUID {620954c7-7044-400f-9c0b-5c9154198aa6}
- Pid: 632
- dwReplEpoch: 0

4.1.3.4.4 Final State

No change in state.

4.1.4 IDL_DRSCrackNames (Opnum 12)

The **IDL_DRSCrackNames** method looks up each of a set of objects in the directory and returns it to the caller in the requested format.

```
ULONG IDL_DRSCrackNames(  
    [in, ref] DRS_HANDLE hDrs,  
    [in] DWORD dwInVersion,  
    [in, ref, switch_is(dwInVersion)]  
        DRS_MSG_CRACKREQ* pmsgIn,  
    [out, ref] DWORD* pdwOutVersion,  
    [out, ref, switch_is(*pdwOutVersion)]  
        DRS_MSG_CRACKREPLY* pmsgOut  
);
```

hDrs: RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

Return Values: 0 if successful, otherwise a Windowserror code.

4.1.4.1 Method-Specific Concrete Types

4.1.4.1.1 DRS_MSG_CRACKREQ

The **DRS_MSG_CRACKREQ** union defines the request messages sent to the [IDL_DRSCrackNames](#) method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef  
[switch_type(DWORD)]  
union {  
    [case(1)]  
        DRS_MSG_CRACKREQ_V1 V1;  
} DRS_MSG_CRACKREQ;
```

V1: Version 1 request.

4.1.4.1.2 DRS_MSG_CRACKREQ_V1

The **DRS_MSG_CRACKREQ_V1** structure defines the request message sent to the [IDL_DRSCrackNames](#) method.

```
typedef struct {  
    ULONG CodePage;  
    ULONG LocaleId;  
    DWORD dwFlags;
```

```

DWORD formatOffered;
DWORD formatDesired;
[range(1,10000)] DWORD cNames;
[string, size_is(cNames)] WCHAR** rpNames;
} DRS_MSG_CRACKREQ_V1;

```

CodePage: The character set used by the client.

LocaleId: The locale used by the client.

dwFlags: Zero or more of the following bit flags:

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	G C R	T R	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	F P O

X: Unused. MUST be zero and ignored.

GC (DS_NAME_FLAG_GC_VERIFY): If set, the call fails if the server is not a GC server.

TR (DS_NAME_FLAG_TRUST_REFERRAL): If set and the lookup fails on the server, referrals are returned to trusted forests where the lookup might succeed.

FPO (DS_NAME_FLAG_PRIVATE_RESOLVE_FPOS): If set and the named object is a foreign security principal, indicate this by using the status of the lookup operation.

formatOffered: The format of the names in rpNames. This may be one of the values from [DS_NAME_FORMAT \(section 4.1.4.1.3\)](#) or one of the following:

Value	Meaning
DS_LIST_SITES 0xFFFFFFFF	Get all sites in the forest .
DS_LIST_SERVERS_IN_SITE 0xFFFFFFFFE	Get all servers in a given site.
DS_LIST_DOMAINS_IN_SITE 0xFFFFFFFFD	Get all domains in a given site.
DS_LIST_SERVERS_FOR_DOMAIN_IN_SITE 0xFFFFFFFFC	Get all DCs of a specified domain in a given site.
DS_LIST_INFO_FOR_SERVER 0xFFFFFFFFB	Get DNS host name and server reference for a given DC.
DS_LIST_ROLES 0xFFFFFFFFA	Get FSMO role owners.
DS_NT4_ACCOUNT_NAME_SANS_DOMAIN	Get value of sAMAccountName attribute.

Value	Meaning
0xFFFFFFFF9	
DS_MAP_SCHEMA_GUID 0xFFFFFFFF8	Get LDAP display name from schema GUID .
DS_LIST_DOMAINS 0xFFFFFFFF7	Get all domains in the forest.
DS_LIST_NCS 0xFFFFFFFF6	Get all NCs in the forest.
DS_ALT_SECURITY_IDENTITIES_NAME 0xFFFFFFFF5	Alternate security identifier .
DS_STRING_SID_NAME 0xFFFFFFFF4	String form of SID.
DS_LIST_SERVERS_WITH_DCS_IN_SITE 0xFFFFFFFF3	Get all DCs in a given site.
DS_LIST_GLOBAL_CATALOG_SERVERS 0xFFFFFFFF1	Get all GCs .
DS_NT4_ACCOUNT_NAME_SANS_DOMAIN_EX 0xFFFFFFFF0	Get value of sAMAccountName attribute; return status DS_NAME_ERROR_NOT_FOUND if account is invalid.
DS_USER_PRINCIPAL_NAME_AND_ALTSECID 0xFFFFFEF	The user principal name and alternate security identifier.

formatDesired: Format of the names in the rItems field of the [DS_NAME_RESULTW](#) structure, which is returned inside the [DRS_MSG_CRACKREPLY](#) message. This may be one of the values from **DS_NAME_FORMAT** or one of the following:

Value	Meaning
DS_STRING_SID_NAME 0xFFFFFFFF4	String form of a SID.
DS_USER_PRINCIPAL_NAME_FOR_LOGON 0xFFFFFFF2	User principal name.

cNames: Count of items in the rpNames array.

rpNames: Input names to translate.

4.1.4.1.3 DS_NAME_FORMAT

The **DS_NAME_FORMAT** enumeration describes the format of a name sent to or received from the [IDL DRSCrackNames](#) method.

```
typedef enum
{
    DS_UNKNOWN_NAME = 0,
    DS_FQDN_1779_NAME = 1,
```

```

    DS_NT4_ACCOUNT_NAME = 2,
    DS_DISPLAY_NAME = 3,
    DS_UNIQUE_ID_NAME = 6,
    DS_CANONICAL_NAME = 7,
    DS_USER_PRINCIPAL_NAME = 8,
    DS_CANONICAL_NAME_EX = 9,
    DS_SERVICE_PRINCIPAL_NAME = 10,
    DS_SID_OR_SID_HISTORY_NAME = 11,
    DS_DNS_DOMAIN_NAME = 12
} DS_NAME_FORMAT;

```

DS_UNKNOWN_NAME: The server looks up the name by using the algorithm specified in the [LookupUnknownName](#) procedure.

DS_FQDN_1779_NAME: A distinguished name.

DS_NT4_ACCOUNT_NAME: Windows NT 4.0 (and prior) name format. The account name is in the format domain\user and the domain-only name is in the format domain\.

DS_DISPLAY_NAME: A user-friendly display name.

DS_UNIQUE_ID_NAME: Dashed string representation of an [objectGUID](#). The format of the string representation is specified in [RFC4122](#) section 3.

DS_CANONICAL_NAME: A canonical name.

DS_USER_PRINCIPAL_NAME: User principal name.

DS_CANONICAL_NAME_EX: Same as DS_CANONICAL_NAME except that rightmost forward slash (/) is replaced with a newline character (\n).

DS_SERVICE_PRINCIPAL_NAME: Service principal name (SPN).

DS_SID_OR_SID_HISTORY_NAME: String representation of a SID (as specified in [\[MS-DTYP\]](#) section 2.4.2).

DS_DNS_DOMAIN_NAME: Not supported.

4.1.4.1.4 DS_NAME_RESULT_ITEMW

The **DS_NAME_RESULT_ITEMW** structure defines the translated name returned by the [IDL DRSCrackNames](#) method.

```

typedef struct {
    DWORD status;
    [string, unique] WCHAR* pDomain;
    [string, unique] WCHAR* pName;
} DS_NAME_RESULT_ITEMW,
*PDS_NAME_RESULT_ITEMW;

```

status: Status of the crack name operation for the corresponding element of the rpNames field in the request. The status is one of the values from the enumeration [DS_NAME_ERROR](#).

pDomain: DNS domain name of the domain in which the named object resides.

pName: Object name in the requested format.

4.1.4.1.5 DS_NAME_RESULTW

The **DS_NAME_RESULTW** structure defines the translated names returned by the [IDL DRSCrackNames](#) method.

```
typedef struct {
    DWORD cItems;
    [size_is(cItems)] PDS_NAME_RESULT_ITEMW rItems;
} DS_NAME_RESULTW,
 *PDS_NAME_RESULTW;
```

cItems: The count of items in the rItems array.

rItems: Translated names that correspond one-to-one with the elements in the rpNames field of the request.

4.1.4.1.6 DRS_MSG_CRACKREPLY

The **DRS_MSG_CRACKREPLY** union defines the response messages received from the [IDL DRSCrackNames](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_CRACKREPLY_V1 V1;
} DRS_MSG_CRACKREPLY;
```

V1: Version 1 reply.

4.1.4.1.7 DRS_MSG_CRACKREPLY_V1

The **DRS_MSG_CRACKREPLY_V1** structure defines the response message received from the [IDL DRSCrackNames](#) method.

```
typedef struct {
    DS_NAME_RESULTW* pResult;
} DRS_MSG_CRACKREPLY_V1;
```

pResult: Translated form of the names.

4.1.4.1.8 DS_NAME_ERROR

This section enumerates the possible statuses of a translation operation.

Symbolic name	Value
DS_NAME_NO_ERROR	0
DS_NAME_ERROR_RESOLVING	1
DS_NAME_ERROR_NOT_FOUND	2
DS_NAME_ERROR_NOT_UNIQUE	3
DS_NAME_ERROR_NO_MAPPING	4
DS_NAME_ERROR_DOMAIN_ONLY	5
DS_NAME_ERROR_TRUST_REFERRAL	7
DS_NAME_ERROR_IS_SID_HISTORY_UNKNOWN	0xFFFFFFFF2
DS_NAME_ERROR_IS_SID_HISTORY_ALIAS	0xFFFFFFFF3
DS_NAME_ERROR_IS_SID_HISTORY_GROUP	0xFFFFFFFF4
DS_NAME_ERROR_IS_SID_HISTORY_USER	0xFFFFFFFF5
DS_NAME_ERROR_IS_SID_UNKNOWN	0xFFFFFFFF6
DS_NAME_ERROR_IS_SID_ALIAS	0xFFFFFFFF7
DS_NAME_ERROR_IS_SID_GROUP	0xFFFFFFFF8
DS_NAME_ERROR_IS_SID_USER	0xFFFFFFFF9
DS_NAME_ERROR_SCHEMA_GUID_CONTROL_RIGHT	0xFFFFFFFFFA
DS_NAME_ERROR_SCHEMA_GUID_CLASS	0xFFFFFFFFFB
DS_NAME_ERROR_SCHEMA_GUID_ATTR_SET	0xFFFFFFFFFC
DS_NAME_ERROR_SCHEMA_GUID_ATTR	0xFFFFFFFFFD
DS_NAME_ERROR_SCHEMA_GUID_NOT_FOUND	0xFFFFFFFFFE
DS_NAME_ERROR_IS_FPO	0xFFFFFFFFF

4.1.4.2 Method-Specific Abstract Types and Procedures

4.1.4.2.1 CanonicalNameFromCanonicalNameEx

```
procedure CanonicalNameFromCanonicalNameEx(
    name: unicodestring): unicodestring
```

This procedure converts name from extended canonical name format to canonical name format by replacing the last newline character in name with a forward slash character. If name is not in the correct format, "domain/container/container/.../container\nleaf" (where \n designates a newline character), this procedure returns null.

4.1.4.2.2 DomainDNSNameFromDomain

```
procedure DomainDNSNameFromDomain(domainNC: DSName): unicodestring
```

If the domain NC, whose root has the [DSName](#) domainNC, is hosted in the forest, this procedure returns the DNS domain name of that domain NC. Otherwise, null is returned.

4.1.4.2.3 DomainFromDomainDNSName

```
procedure DomainFromDomainDNSName(domainName: unicodestring): DSName
```

If the DC hosts an NC replica of the domain NC whose DNS domain name is domainName, this procedure returns the [DSName](#) of the root of that domain NC. Otherwise, it returns null.

4.1.4.2.4 DomainNameFromCanonicalName

```
procedure DomainNameFromCanonicalName(  
    canonicalName: unicodestring): unicodestring
```

Given a name in canonical format, this procedure extracts and returns the domain DNS name. If the input is not in canonical name format, then null is returned. For example, when the input is "example.fabrikam.com/container/username", the returned domain DNS name is "example.fabrikam.com".

4.1.4.2.5 DomainNameFromSid

```
procedure DomainNameFromSid(domainSid: SID): unicodestring
```

Looks up the domain SID domainSid among trusted domains and domains in trusted forests. If domainSid is the domain SID of a trusted domain, then the name of this domain is returned. If the input is null, then null is returned.

4.1.4.2.6 DomainNameFromUPN

```
procedure DomainNameFromUPN(upn: unicodestring): unicodestring
```

Parses and returns the domain name from a UPN-formatted string upn. The domain name is the component after the '@'. For example, when the input is "username@example.fabrikam.com", then "example.fabrikam.com" is returned. If upn is not in UPN format, then null is returned.

4.1.4.2.7 DomainNetBIOSNameFromDomain

```
procedure DomainNetBIOSNameFromDomain(domainNC: DSName): unicodestring
```

If the domain NC, whose root has the [DSName](#) domainNC, is hosted in the forest, this procedure returns the NetBIOS domain name of that domain NC. Otherwise, null is returned.

4.1.4.2.8 DomainSidFromSid

```
procedure DomainSidFromSid(sid: SID): SID
```

Removes the last sub-authority from the input security identifier **sid** and returns the resulting security identifier, which is the domain SID. If the input is null, the procedure returns null. See [\[MS-DTYP\]](#) section **2.4.2** for more information on security identifiers (SIDs).

4.1.4.2.9 LookupName

```
procedure LookupName(  
  flags: DWORD,  
  formatOffered: DWORD,  
  name: unicodestring): DS_NAME_RESULT_ITEM
```

Informative summary of behavior: The LookupName procedure performs the lookup of a single name in a given input format and produces the output name in the required format.

```
rt: sequence of DSName  
obj: DSName  
fSidHistory: boolean  
result: DS_NAME_RESULT_ITEM  
names: sequence of unicodestring  
  
if formatOffered = DS_UNKNOWN_NAME then  
  return LookupUnknownName(flags, name)  
endif  
  
if formatOffered = DS_FQDN_1779_NAME then  
  rt := LookupAttr(flags, distinguishedName, name)  
else if formatOffered = DS_NT4_ACCOUNT_NAME then  
  rt := LookupAttr(flags, sAMAccountName,  
    UserNameFromNT4AccountName(name))  
else if formatOffered = DS_USER_PRINCIPAL_NAME then  
  rt := LookupAttr(flags, userPrincipalName, name)  
else if formatOffered = DS_CANONICAL_NAME then  
  rt := LookupCanonicalName(name)  
else if formatOffered = DS_UNIQUE_ID_NAME then  
  rt := LookupAttr(flags, objectGUID, name)  
else if formatOffered = DS_DISPLAY_NAME then  
  rt := LookupAttr(flags, displayName, name)  
else if formatOffered = DS_SERVICE_PRINCIPAL_NAME then  
  rt := LookupSPN(flags, name)  
else if formatOffered in {DS_SID_OR_SID_HISTORY_NAME,  
  DS_STRING_SID_NAME} then  
  rt := LookupSID(flags, SidFromStringSid(name))  
else if formatOffered = DS_CANONICAL_NAME_EX then  
  rt := LookupCanonicalName(CanonicalNameFromCanonicalNameEx(name))  
else if formatOffered in {DS_NT4_ACCOUNT_NAME_SANS_DOMAIN,  
  DS_NT4_ACCOUNT_NAME_SANS_DOMAIN_EX} then  
  rt := LookupAttr(flags, sAMAccountName, name)  
else if formatOffered = DS_ALT_SECURITY_IDENTITIES_NAME then  
  rt := LookupAttr(flags, altSecurityIdentities, name)  
else if formatOffered = DS_USER_PRINCIPAL_NAME_AND_ALTSECID then  
  rt := LookupUPNAndAltSecID(flags, name)
```

```

else
    rt := null
endif

result.pName^ := null
result.pDomain^ := null
if rt = null then
    /* No match. */
    result.status := DS_NAME_ERROR_NOT_FOUND
    return result
endif

if rt.length > 1 then
    /* Found more than one matching object. */
    result.status := DS_NAME_ERROR_NOT_UNIQUE
    return result
endif

obj := rt[0]

if formatOffered = DS_NT4_ACCOUNT_NAME_SANS_DOMAIN_EX then
    /* Check that the account is valid. */
    if obj!userAccountControl ∩ {ADS_UF_ACCOUNTDISABLE,
        ADS_UF_TEMP_DUPLICATE_ACCOUNT} ≠ {} then
        result.status := DS_NAME_ERROR_NOT_FOUND
        return result
    endif
endif

/* Found exactly one object. Construct the output name in the
 * desired format. */
names := ConstructOutput(obj, msgIn.formatDesired)
if names = null then
    /* Could not construct the required name format. */
    result.status := DS_NAME_ERROR_NOT_FOUND
    return result
endif
if names.length > 1 then
    /* Too many output names. */
    result.status := DS_NAME_ERROR_NOT_UNIQUE
    return result
endif

result.pName^ := names[0]
result.pDomain^ := DomainDNSNameFromDomain(GetObjectNC(obj))
result.status = DS_NAME_NO_ERROR

if formatOffered = DS_STRING_SID_NAME then
    /* We need to specify the type of the object in result.status. */
    /* Check if the value came from sIDHistory or objectSid. */
    fSidHistory := SidFromStringSid(name) in obj!sidHistory

    if obj!sAMAccountType in {SAM_USER_OBJECT, SAM_MACHINE_ACCOUNT,
        SAM_TRUST_ACCOUNT} then
        if fSidHistory then
            result.status := DS_NAME_ERROR_IS_SID_HISTORY_USER
        else
            result.status := DS_NAME_ERROR_IS_SID_USER
        endif
    endif
endif

```

```

endif
else if obj!sAMAccountType in {SAM_NON_SECURITY_GROUP_OBJECT,
    SAM_GROUP_OBJECT} then
    if fSidHistory then
        result.status := DS_NAME_ERROR_IS_SID_HISTORY_GROUP
    else
        result.status := DS_NAME_ERROR_IS_SID_GROUP
    endif
else if obj!sAMAccountType in {SAM_NON_SECURITY_ALIAS_OBJECT,
    SAM_ALIAS_OBJECT} then
    if fSidHistory then
        result.status := DS_NAME_ERROR_IS_SID_HISTORY_ALIAS
    else
        result.status := DS_NAME_ERROR_IS_SID_ALIAS
    endif
else
    if fSidHistory then
        result.status := DS_NAME_ERROR_IS_SID_HISTORY_UNKNOWN
    else
        result.status := DS_NAME_ERROR_IS_SID_UNKNOWN
    endif
endif
endif
endif

if foreignSecurityPrincipal in obj!objectClass and
    DS_NAME_FLAG_PRIVATE_RESOLVE_FPOS in flags then
    result.status = DS_NAME_ERROR_IS_FPO
endif

return result

```

4.1.4.2.10 LookupAttr

```

procedure LookupAttr(
    flags: DWORD,
    att: ATTRTYP,
    attrValue: unicodestring): set of DSName

```

Informative summary of behavior: The LookupAttr procedure is a helper function that looks up an object in an NC replica based on an attributeName=attributeValue criterion. It returns the set of objects that match the criterion.

```

rt: set of DSName

if DS_NAME_FLAG_GC_VERIFY in flags then
    rt := select all O from all
        where attrValue in GetAttrVals(O, att, false)
else
    rt := select all O from subtree DefaultNC()
        where attrValue in GetAttrVals(O, att, false)
endif
return rt

```

4.1.4.2.11 LookupCanonicalName

```
procedure LookupCanonicalName(name: unicodestring): DSName
```

Informative summary of behavior: The LookupCanonicalName procedure is a helper function that looks up an object based on its canonical name by walking down the NC replica from the NC root and looking up objects by name.

```
curObj: DSName
label: unicodestring

ParseCanonicalName(name, label, name)
curObj := DomainFromDomainDNSName(label)
while name ≠ null and curObj ≠ null
  ParseCanonicalName(name, label, name)
  curObj := select one O from children curObj where O!name=label
  if curObj = null then
    return null
  endif
endwhile
return curObj
```

4.1.4.2.12 GetCanonicalName

```
procedure GetCanonicalName(
  obj: DSName, extended: boolean): unicodestring
```

Informative summary of behavior: The **GetCanonicalName** function constructs the canonical name of an object by walking up its ancestors to the NC root.

```
result: unicodestring

if obj = GetObjectNC(obj) then
  return DomainDNSNameFromDomain(obj)
endif

/* Recurse into parent, obtain non-extended canonical name. */
result := GetCanonicalName(obj!parent, false)
if extended = true then
  result := result + "\n"
else
  result := result + "/"
endif

result := result + obj!name
return result
```

4.1.4.2.13 LookupSPN

```
procedure LookupSPN(flags: DWORD, name: unicodestring): set of DSName
```

Informative summary of behavior: LookupSPN is a helper function that implements the service principal name (SPN) lookup algorithm.

```
rt: set of DSName
obj: DSName
dcGuid: GUID
spnMappings: set of unicodestring
mappedSpn: unicodestring
/* First, try to look up the SPN directly. */
rt := LookupAttr(flags, servicePrincipalName, name)
if rt ≠ null then
    return rt
endif
/* Obtain SPN mappings value. */
obj := DescendantObject(ConfigNC(),
    "CN=Directory Service,CN=Windows,CN=Services,")
spnMappings := obj!sPNMappings
if spnMappings ≠ null
    mappedSpn := MapSPN(name, spnMappings)
    if mappedSpn ≠ null then
        /* try to lookup a mapped SPN */
        rt := LookupAttr(flags, servicePrincipalName, mappedSpn)
        if rt ≠ null then
            return rt
        endif
    endif
endif
/* Try to find replication SPN, which might not be present in our
 * NC replicas yet. */
if GetServiceClassFromSPN(spn) = DRS_SPN_CLASS and
    GetServiceNameFromSPN(spn) =
        DomainNameFromDN(DefaultNC()!distinguishedName) then
    /* Yes, it looks like a replication SPN. Try to find DC by guid. */
    dcGuid := GuidFromString(GetInstanceNameFromSPN(spn))
    if dcGuid ≠ null then
        /* Find DSA object with this objectGUID value. */
        obj := select one o from subtree ConfigNC()
            where o!objectGUID = dcGuid
        if obj ≠ null then
            /* Get the server object. */
            obj := obj!parent
            if obj ≠ null then
                /* server!serverReference points to the DC's computer
                 * object.*/
                rt := {obj!serverReference}
            endif
        endif
    endif
endif
return rt
```

4.1.4.2.14 LookupSID

```
procedure LookupSID(flags: DWORD, sid: SID): set of DSName
```

Informative summary of behavior: The LookupSID procedure is a helper function that implements the SID lookup algorithm.

```
rt1, rt2: set of DSName
rt1 := LookupAttr(flags, objectSid, sid)
rt2 := LookupAttr(flags, sIDHistory, sid)

return rt1 + rt2
```

4.1.4.2.15 LookupUnknownName

```
procedure LookupUnknownName(
    flags: DWORD,
    name: unicodestring): DS_NAME_RESULT_ITEM
```

Informative summary of behavior: The server uses LookupUnknownName to look up names of format DS_UNKNOWN_NAME. LookupUnknownName looks up the name by trying formats in the specific order listed in the foreach statement shown below until a lookup succeeds.

```
result: DS_NAME_RESULT_ITEM
format: DWORD

/* Attempt to resolve in the following formats in this specific
 * order. */
foreach format in {DS_FQDN_1779_NAME, DS_USER_PRINCIPAL_NAME,
                  DS_NT4_ACCOUNT_NAME, DS_CANONICAL_NAME,
                  DS_UNIQUE_ID_NAME, DS_DISPLAY_NAME,
                  DS_SERVICE_PRINCIPAL_NAME,
                  DS_SID_OR_SID_HISTORY_NAME,
                  DS_CANONICAL_NAME_EX}
    result := LookupName(flags, format, name)
    if result.status ≠ DS_NAME_ERROR_NOT_FOUND then
        return result
    endif
endfor
return result
```

4.1.4.2.16 LookupUPNAndAltSecID

```
procedure LookupUPNAndAltSecID(
    flags: DWORD,
    name: unicodestring): set of DSName
```

Informative summary of behavior: Returns [DSNames](#) of objects, with the given value as a value of [userPrincipalName](#), [altSecurityIdentities](#), or [sAMAccountName](#).

```
rt: set of DSName
/* First, try lookup by userPrincipalName. */
rt := LookupAttr(flags, userPrincipalName, name)
if rt ≠ null then
    return rt
endif
```



```

/* Next, try lookup by altSecurityIdentities. */
rt := LookupAttr(flags, altSecurityIdentities, name)
if rt ≠ null then
    return rt
endif
/* Finally, attempt to parse the name as simpleName@domain and
 * search for
 * sAMAccountName=simpleName. */
name := UserNameFromUPN(name)
if name ≠ null then
    rt := LookupAttr(flags, sAMAccountName, name)
endif
return rt

```

4.1.4.2.17 LookupDomainSyntactically

```

procedure LookupDomainSyntactically(
    flags: DWORD,
    formatOffered: DWORD,
    name: unicodestring): DS_NAME_RESULT_ITEM

```

Informative summary of behavior: LookupDomainSyntactically is a helper function that attempts to obtain the domain name from a name that is otherwise unresolvable. If the domain name can be parsed out, and if it represents a trusted domain, then this domain name is returned in the DS_NAME_RESULT_ITEM.pDomain member variable, with an appropriate status value.

```

domainSID: SID
domainName: unicodestring
domainDN: unicodestring
result: DS_NAME_RESULT_ITEM

domainName := null
if formatOffered = DS_FQDN_1779_NAME then
    domainDN := RetrieveDCSuffixFromDn(name)
    domainName := DomainDNSNameFromDomain(domainDN)
else if formatOffered = DS_NT4_ACCOUNT_NAME then
    domainName := DomainNameFromNT4AccountName(name)
else if formatOffered in {DS_CANONICAL_NAME,
    DS_CANONICAL_NAME_EX} then
    domainName := DomainNameFromCanonicalName(name)
else if formatOffered = DS_USER_PRINCIPAL_NAME then
    domainName := DomainNameFromUPN(name)
else if formatOffered = DS_SERVICE_PRINCIPAL_NAME then
    domainName := GetServiceNameFromSPN(name)
else if formatOffered = DS_SID_OR_SID_HISTORY_NAME then
    domainSID := DomainSidFromSid(SidFromStringSid(name))
    domainName := DomainNameFromSid(domainSID)
endif

if domainName ≠ null and IsDomainNameInTrustedForest(domainName) then
    /* We found domain name syntactically, and it is a domain we
     * trust. */
    result.pDomain^ := domainName
    if DS_NAME_FLAG_TRUST_REFERRAL in flags then
        result.status := DS_NAME_ERROR_TRUST_REFERRAL
    end if
end if

```

```

    else
        result.status := DS_NAME_ERROR_DOMAIN_ONLY
    endif
else
    /* We could not even parse the domain out. */
    result.pDomain^ := null
    result.status := DS_NAME_ERROR_NOT_FOUND
endif

return result

```

4.1.4.2.18 MapSPN

```

procedure MapSPN(spn: unicodestring,
                 spnMappings: set of unicodestring):
    unicodestring

```

The MapSPN procedure performs an SPN mapping operation on spn according to the map specified in spnMappings, and returns the mapped version of spn. The mapping operation is used to change the service class of the SPN. An SPN service class is the first part of an SPN; for example, "ldap" is the service class of the SPN "ldap/fabrikam.com".

Each value of spnMappings consists of an alias, followed by an equals sign (=), followed by a comma-separated list of one or more SPN service classes. Thus, each value must be in the following format:

alias=serviceClass1,serviceClass2,serviceClass3,...,serviceClassN

If the service class portion of spn corresponds to one of the serviceClassX values in value v of spnMappings, then the return value of this procedure is the SPN value this is constructed from spn by substituting the alias value from v as the service class of spn. If no mapping is found (that is, if there is no such v), or if spn is not an SPN, then null is returned.

For example, suppose that spnMappings is the following set:

{"ldap=ldap,otherldap", "host=alterter,appmgmt,cisvc"}

If spn is "alterter/fabrikam.com", then the procedure returns "host/fabrikam.com".

4.1.4.2.19 ParseCanonicalName

```

procedure ParseCanonicalName(
    name: unicodestring,
    var firstPart: unicodestring,
    var remainder: unicodestring)

```

The ParseCanonicalName procedure parses the first label from the canonical name string name and returns the first label in firstPart and the remainder of the string in remainder. For example, name = "container1/container2/leaf" is parsed as firstPart:= "container1" and remainder:= "container2/leaf". As another example, name = "example.fabrikam.com/container/username" is parsed as firstPart:= "example.fabrikam.com" and remainder:= "container/username". If name does not contain a slash character, then it is parsed as firstPart:= name and remainder:= null.

4.1.4.2.20 RetrieveDCSuffixFromDn

```
procedure RetrieveDCSuffixFromDn(dn: unicodestring): unicodestring
```

The RetrieveDCSuffixFromDn procedure parses the distinguished name dn syntactically and returns the suffix that consists entirely of the RDN components whose attribute is "DC". For example, given "CN=Administrator,CN=Users,DC=fabrikam,DC=com", this procedure would return "DC=fabrikam,DC=com".

4.1.4.2.21 UserNameFromUPN

```
procedure UserNameFromUPN(upn: unicodestring): unicodestring
```

Parses and returns the user name from a UPN-formatted string upn. The user name is the component before the '@'. For example, when the input is "username@example.fabrikam.com", then "username" is returned. If the input is not in UPN format, then null is returned.

4.1.4.2.22 ConstructOutput

```
procedure ConstructOutput(  
    obj: DSName,  
    formatDesired: DWORD): set of unicodestring
```

Informative summary of behavior: ConstructOutput is a helper function that constructs the name of the object in the required output format. Note that the returned set of values may be empty, or may contain more than one value. These situations are handled by the caller function, [LookupName \(section 4.1.4.2.9\)](#).

```
if formatDesired = DS_FQDN_1779_NAME then  
    return {obj!distinguishedName}  
else if formatDesired = DS_NT4_ACCOUNT_NAME then  
    return {DomainNetBIOSNameFromDomain(GetObjectNC(obj)) + "\" +  
        obj!sAMAccountName}  
else if formatDesired = DS_USER_PRINCIPAL_NAME then  
    return {obj!userPrincipalName}  
else if formatDesired = DS_CANONICAL_NAME then  
    return {GetCanonicalName(obj, false)}  
else if formatDesired = DS_UNIQUE_ID_NAME then  
    return {GuidToString(obj!objectGUID)}  
else if formatDesired = DS_DISPLAY_NAME then  
    return {obj!displayName}  
else if formatDesired = DS_SERVICE_PRINCIPAL_NAME then  
    return obj!servicePrincipalName  
else if formatDesired = DS_CANONICAL_NAME_EX then  
    return {GetCanonicalName(obj, true)}  
else if formatDesired = DS_STRING_SID_NAME then  
    return {StringSidFromSid(obj!objectSid)}  
else if formatDesired = DS_USER_PRINCIPAL_NAME_FOR_LOGON then  
    /* If UPN is set, then return it. */  
    if obj!userPrincipalName ≠ null then  
        return {obj!userPrincipalName}  
    endif  
    return {obj!sAMAccountName + "@" +
```

```

        DomainDNSNameFromDomain(GetObjectNC(obj)) }
endif
/* Otherwise, unknown format. */
return null

```

4.1.4.3 Server Behavior of the IDL_DRSCrackNames Method

Informative summary of behavior: The [IDL_DRSCrackNames](#) method is a generic method that is used to look up information in the directory. The most common usage is looking up directory object names that are provided in one format (for example, SPNs) and returning them in a different format (for example, DNs). One special mode occurs when the input format is not specified, in which case the server tries to "guess" the format of the name by following some heuristics. The method can also be used to look up generic information in the directory, such as the list of sites, or the list of servers in a specific site.

```

ULONG
IDL_DRSCrackNames(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_CRACKREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_CRACKREPLY *pmsgOut)

msgIn: DRS_MSG_CRACKREQ_V1
msgOut: DS_NAME_RESULTW
i: DWORD
rt: set of DSName
serverObj, siteObj, attr, class, er: DSName

pdwOutVersion^ := 1
msgOut^.V1.pResult^.cItems := 0
msgOut^.V1.pResult^.rItems := null
if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
if DS_NAME_FLAG_GC_VERIFY in msgIn.dwFlags and
    not IsGC() then
    return ERROR_DS_GCVERIFY_ERROR
endif

if msgIn.formatOffered in {
    all constants in DS_NAME_FORMAT enumeration,
    DS_NT4_ACCOUNT_NAME_SANS_DOMAIN,
    DS_NT4_ACCOUNT_NAME_SANS_DOMAIN_EX,
    DS_ALT_SECURITY_IDENTITIES_NAME,
    DS_STRING_SID_NAME,
    DS_USER_PRINCIPAL_NAME_AND_ALTSECID} then
/* Regular name lookup. */
for i := 0 to msgIn.cNames - 1
    /* Perform the lookup based on the input format. */
    msgOut.rItems[i] := LookupName(
        msgIn.dwFlags, msgIn.formatOffered, msgIn.rItems[i])
    if msgOut.rItems[i] = null then
        /* Did not find anything. Try to at least guess the domain

```

```

        * name from the input name. */
        msgOut.rItems[i] := LookupDomainSyntactically(msgIn.dwFlags,
            msgIn.formatOffered, msgIn.rItems[i])
    endif
endfor
msgOut.cItems = msgIn.cNames
else if msgIn.formatOffered = DS_LIST_ROLES then
    /* Return the list of FSMO role owners. */
    i := 0
    foreach role in {FSMO_SCHEMA, FSMO_DOMAIN_NAMING, FSMO_PDC,
        FSMO_RID, FSMO_INFRASTRUCTURE}
        msgOut.rItems[i].pName := GetFSMORoleOwner(role).dn
        msgOut.rItems[i].status := DS_NAME_NO_ERROR
        i := i + 1
    endfor
    msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_SITES then
    /* Return the list of known sites. */
    rt := select all o from children
        DescendantObject(ConfigNC(), "CN=Sites,")
        where o!objectCategory = GetDefaultObjectCategory(site)
    i := 0
    foreach siteObj in rt
        msgOut.rItems[i].pName := siteObj.dn
        msgOut.rItems[i].status := DS_NAME_NO_ERROR
        i := i + 1
    endfor
    msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_SERVERS_IN_SITE then
    /* Return all DCs in a site named msgIn.rItems[0]. */
    rt := select all o from subtree msgIn.rItems[0]
        where o!objectCategory = GetDefaultObjectCategory(server)
    i := 0
    foreach serverObj in rt
        msgOut.rItems[i].pName := serverObj.dn
        msgOut.rItems[i].status := DS_NAME_NO_ERROR
        i := i + 1
    endfor
    msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_DOMAINS then
    /* Return all known AD domains. */
    rt := select all o from
        subtree DescendantObject(ConfigNC(), "CN=Partitions")
        where o!objectCategory = GetDefaultObjectCategory(crossRef)
        and FLAG_CR_NTDS_DOMAIN in o!systemFlags
    i := 0
    foreach crObj in rt
        msgOut.rItems[i].pName := crObj!ncName.dn
        msgOut.rItems[i].status := DS_NAME_NO_ERROR
        i := i + 1
    endfor
    msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_NCS then
    /* Return all known NCs. */
    rt := select all o from
        subtree DescendantObject(ConfigNC(), "CN=Partitions")
        where o!objectCategory = GetDefaultObjectCategory(crossRef)
    i := 0

```

```

foreach crObj in rt
    msgOut.rItems[i].pName := crObj!ncName.dn
    msgOut.rItems[i].status := DS_NAME_NO_ERROR
    i := i + 1
endfor
msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_DOMAINS_IN_SITE then
    /* Return the list of domains that are hosted by DCs in a site
    * named msgIn.rItems[0]. */
    /* First find all DCs in a site named msgIn.rItems[0]. */
    rt := select all o from subtree msgIn.rItems[0]
        where o!objectCategory = GetDefaultObjectCategory(nTDSDSA)
    /* Gather the list of all domains from DSA object. */
    hostedDomains := null
    foreach dsaObj in rt
        /* Union operation eliminates duplicates. */
        hostedDomains := hostedDomains + dsaObj!hasMasterNCs
    endfor
    i := 0
    foreach domain in hostedDomains
        if domain ≠ SchemaNC() and domain ≠ ConfigNC() then
            msgOut.rItems[i].pName := domain.dn
            msgOut.rItems[i].status := DS_NAME_NO_ERROR
            i := i + 1
        endif
    endfor
    msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_SERVERS_FOR_DOMAIN_IN_SITE then
    /* Return all DSAs hosting domain msgIn.rItems[0] in a site named
    * msgIn.rItems[0]. */
    rt := select all o from subtree msgIn.rItems[0]
        where o!objectCategory = GetDefaultObjectCategory(nTDSDSA)
        and msgIn.rItems[0] in o!msDS-hasMasterNCs
    /* Return the list of server objects (parents of DSAs). */
    i := 0
    foreach dsaObj in rt
        serverObj := select one o from subtree ConfigNC() where
            o!objectGUID = dsaObj!parent
        msgOut.rItems[i].pName := serverObj.dn
        msgOut.rItems[i].status := DS_NAME_NO_ERROR
        i := i + 1
    endfor
    msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_SERVERS_WITH_DCS_IN_SITE then
    /* Return all servers that have DSA objects in a site named
    * msgIn.rItems[0]. */
    rt := select all o from subtree msgIn.rItems[0]
        where o!objectCategory = GetDefaultObjectCategory(nTDSDSA)
        and o!hasMasterNCs ≠ null
    /* Return the list of server objects (parents of DSAs). */
    i := 0
    foreach dsaObj in rt
        serverObj := select one o from subtree ConfigNC() where
            o!objectGUID = dsaObj!parent
        msgOut.rItems[i].pName := serverObj.dn
        msgOut.rItems[i].status := DS_NAME_NO_ERROR
        i := i + 1
    endfor

```

```

msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_INFO_FOR_SERVER then
/* Returns the DSA object, the dnsHostName and the serverReference
 * for the server specified by msgIn.rItems[0]. */
serverObj := GetDSNameFromDN(msgIn.rItems[0])
dsaObj := select one o from subtree msgIn.rItems[0]
        where o!objectCategory = GetDefaultObjectCategory(nTDSDSA)
if dsaObj ≠ null then
/* Ok, looks like a valid server object. */
msgOut.rItems[0].pName := dsaObj.dn
msgOut.rItems[0].status := DS_NAME_NO_ERROR
msgOut.rItems[1].pName := serverObj.dn!dnsHostName
msgOut.rItems[1].status := DS_NAME_NO_ERROR
msgOut.rItems[2].pName := dsaObj!serverReference
msgOut.rItems[2].status := DS_NAME_NO_ERROR
msgOut.cItems := 3
endif
else if msgIn.formatOffered = DS_LIST_GLOBAL_CATALOG_SERVERS then
/* Returns the list of GC servers, including the info which site
 * each GC belongs to. */
rt := select all o from subtree ConfigNC()
        where O!objectCategory = GetDefaultObjectCategory(nTDSDSA)
        and NTDSDSA_OPT_IS_GC in o!options and o!invocationId ≠ null
i := 0
foreach dsaObj in rt
/* server object is the parent of the DSA object. */
serverObj := select one o from subtree ConfigNC() where
        o!objectGUID = dsaObj!parent
/* Site object is the parent of the server object. */
siteObj := select one o from subtree ConfigNC() where
        o!objectGUID = serverObj!parent
msgOut.rItems[i].pDomain := serverObj!dnsHostName
msgOut.rItems[i].pName := siteObj.dn
msgOut.rItems[i].status := DS_NAME_NO_ERROR
i := i+1
endfor
msgOut.cItems := i
else if msgIn.formatOffered = DS_MAP_SCHEMA_GUID then
for i := 0 to msgIn.cNames - 1
/* Map a guid contained in msgIn.rItems[i] to attribute or class
 * or propertySet. */
/* Assume no match by default. */
msgOut.rItems[i].status := DS_NAME_ERROR_NOT_FOUND

/* First, try to find a matching attribute. */
attr := select one o from subtree SchemaNC()
        where attributeSchema in o!objectClass and
        o!schemaIdGuid = msgIn.rItems[i]
if attr ≠ null
/* Found a matching attribute object. */
msgOut.rItems[i].pName := attr!LDAPDisplayName
msgOut.rItems[i].status := DS_NAME_ERROR_SCHEMA_GUID_ATTR
else
/* Next, try to find a matching class. */
class := select one o from subtree SchemaNC()
        where classSchema in o!objectClass
        o!schemaIdGuid = msgIn.rItems[i]
if class ≠ null

```

```

/* Found a matching class object. */
msgOut.rItems[i].pName := class!lDAPDisplayName
msgOut.rItems[i].status := DS_NAME_ERROR_SCHEMA_GUID_CLASS
else
/* Finally, try to find a matching extendedRight object. */
er := select one o from
    subtree DescendantObject(ConfigNC(),
        "CN=Extended-Rights,")
    where extendedRight in o!objectClass and
        o!rightsGuid = msgIn.rItems[i]
if er ≠ null
/* Found a matching extendedRight object */
if RIGHT_DS_READ_PROPERTY in er!validAccesses or
    RIGHT_DS_WRITE_PROPERTY in er!validAccesses then
    msgOut.rItems[i].pName := er!displayName
    msgOut.rItems[i].status :=
        DS_NAME_ERROR_SCHEMA_GUID_ATTR_SET
else if RIGHT_DS_CONTROL_ACCESS in er!validAccesses or
    RIGHT_DS_WRITE_PROPERTY_EXTENDED in er!validAccesses
    then
    msgOut.rItems[i].pName := er!displayName
    msgOut.rItems[i].status :=
        DS_NAME_ERROR_SCHEMA_GUID_CONTROL_RIGHT
    endif
endif
endif
endif
endif
msgOut.cItems := msgIn.cNames
endif

pmsgOut^.V1.pResult := ADR(msgOut)
return ERROR_SUCCESS

```

4.1.4.4 Examples of the IDL_DRSCrackNames Method

When user "Kim Akers" logs on to the computer MS1.Contoso.com using her Windows NT 4.0 account name "CONTOSO\kimakers", the domain controller needs to obtain a fully qualified DN (FQDN) that corresponds to the Windows NT 4.0 account name. The domain controller DC1 calls [IDL_DRSCrackNames](#) to translate the Windows NT 4.0 account name to an FQDN.

4.1.4.4.1 Initial State

Querying the [user](#) object with name KimAkers in the domain NC DC=CONTOSO, DC=COM on DC1:

- `ldap_search_s("CN=Kim Akers,CN=Users,DC=contoso,DC=com", baseObject, "(objectClass=user)", [objectClass, distinguishedName, sAMAccountName])`
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=Kim Akers,CN=Users,DC=contoso,DC=com

- 4> objectClass: top; person; organizationalPerson; user;
- 1> distinguishedName: CN=Kim Akers,CN=Users,DC=contoso,DC=com;
- 1> sAMAccountName: KimAkers

Querying the [crossRef](#) object on the NC root object for domain NC CONTOSO.COM on DC1:

Expanding base 'CN=CONTOSO,CN=Partitions,CN=Configuration,DC=contoso,DC=com'...

Result <0>: (null)

Matched DNs:

Getting 1 entries:

>> Dn: CN=CONTOSO,CN=Partitions,CN=Configuration,DC=contoso,DC=com

- 2> objectClass: top; crossRef;
- 1> nCName: DC=contoso,DC=com;
- 1> dnsRoot: contoso.com;
- 1> nETBIOSName: CONTOSO;

4.1.4.4.2 Client Request

DC1 invokes the [IDL DRSCrackNames](#) method against itself with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 1
- pmsgIn = DRS_MSG_CRACKREQ_V1
 - CodePage = 0x4e4
 - LocaleId = US-EN
 - dwFlags = DS_NAME_NO_FLAGS
 - formatOffered = DS_NT4_ACCOUNT_NAME
 - formatDesired = DS_FQDN_1779_NAME
 - cNames: 1
 - rpNames: "CONTOSO\kimakers"

4.1.4.4.3 Server Response

Returns code of 0 and the following values:

- pdwMessageOut = 1
- pmsgOut = DRS_MSG_CRACKREPLY_V1
 - pResult: DS_NAME_RESULTW

- cNames: 1
- rItems: DS_NAME_RESULT_ITEMW
- pDomain: "contoso.com"
- pName: "CN=Kim Akers,CN=Users,DC=contoso,DC=com"
- status: DS_NAME_NO_ERROR

4.1.4.4.4 Final State

The final state is the same as the initial state; there is no change.

4.1.5 IDL_DRSDomainControllerInfo (Opnum 16)

The **IDL_DRSDomainControllerInfo** method retrieves information about DCs in a given domain.

```
ULONG IDL_DRSDomainControllerInfo(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_DCINFOREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_DCINFOREPLY* pmsgOut
);
```

hDrs: RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.5.1 Method-Specific Concrete Types

4.1.5.1.1 DRS_MSG_DCINFOREQ

The **DRS_MSG_DCINFOREQ** union defines the request messages sent to the [IDL_DRSDomainControllerInfo](#) method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_DCINFOREQ_V1 V1;
} DRS_MSG_DCINFOREQ,
*PDRS_MSG_DCINFOREQ;
```

V1: Version 1 request.

4.1.5.1.2 DRS_MSG_DCINFOREQ_V1

The **DRS_MSG_DCINFOREQ_V1** structure defines the request message sent to the [IDL DRSDomainControllerInfo](#) method.

```
typedef struct {
    [string] WCHAR* Domain;
    DWORD InfoLevel;
} DRS_MSG_DCINFOREQ_V1;
```

Domain: DNS domain name for which the client requests DC information.

InfoLevel: The response version requested by the client: 1, 2, 3, or 0xFFFFFFFF. The responses at InfoLevel 1, 2, and 3 all contain information about DCs in the given domain. The information at InfoLevel 1 is a subset of the information at InfoLevel 2, which is a subset of the information at InfoLevel 3. InfoLevel 3 includes information about the RODCs in the given domain. InfoLevel 0xFFFFFFFF server returns information about the active LDAP connections.

4.1.5.1.3 DRS_MSG_DCINFOREPLY

The **DRS_MSG_DCINFOREPLY** union defines the response messages received from the [IDL DRSDomainControllerInfo](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_DCINFOREPLY_V1 V1;
    [case(2)]
        DRS_MSG_DCINFOREPLY_V2 V2;
    [case(3)]
        DRS_MSG_DCINFOREPLY_V3 V3;
    [case(0xFFFFFFFF)]
        DRS_MSG_DCINFOREPLY_VFFFFFFFF VFFFFFFFF;
} DRS_MSG_DCINFOREPLY;
```

V1: Version 1 response.

V2: Version 2 response.

V3: Version 3 response.

VFFFFFFFF: Version 0xFFFFFFFF response.

4.1.5.1.4 DRS_MSG_DCINFOREPLY_V1

The **DRS_MSG_DCINFOREPLY_V1** structure defines the response message received from the [IDL DRSDomainControllerInfo](#) method, when the client has requested InfoLevel = 1.

```
typedef struct {
    [range(0,10000)] DWORD cItems;
    [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_1W* rItems;
} DRS_MSG_DCINFOREPLY_V1;
```

cItems: Count of items in the rItems array.

rItems: DC information.

4.1.5.1.5 DRS_MSG_DCINFOREPLY_V2

The **DRS_MSG_DCINFOREPLY_V2** structure defines the response message received from the [IDL DRSDomainControllerInfo](#) method, when the client has requested InfoLevel = 2.

```
typedef struct {
    [range(0,10000)] DWORD cItems;
    [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_2W* rItems;
} DRS_MSG_DCINFOREPLY_V2;
```

cItems: Count of items in the rItems array.

rItems: DC information.

4.1.5.1.6 DRS_MSG_DCINFOREPLY_V3

The **DRS_MSG_DCINFOREPLY_V3** structure defines the response message received from the [IDL DRSDomainControllerInfo](#) method when the client has requested InfoLevel = 3.

```
typedef struct {
    [range(0,10000)] DWORD cItems;
    [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_3W* rItems;
} DRS_MSG_DCINFOREPLY_V3;
```

cItems: Count of items in the rItems array.

rItems: DC information.

4.1.5.1.7 DRS_MSG_DCINFOREPLY_VFFFFFFFF

The **DRS_MSG_DCINFOREPLY_VFFFFFFFF** structure defines the response message received from the [IDL DRSDomainControllerInfo](#) method, when the client has requested InfoLevel = 0xFFFFFFFF.

```
typedef struct {
    [range(0,10000)] DWORD cItems;
```

```

    [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_FFFFFFFF* rItems;
} DRS_MSG_DCINFOREPLY_VFFFFFFF;

```

cItems: The count of items in the rItems array.

rItems: DC information.

4.1.5.1.8 DS_DOMAIN_CONTROLLER_INFO_1W

The **DS_DOMAIN_CONTROLLER_INFO_1W** structure defines DC information that is returned as a part of the response to an InfoLevel = 1 request. The struct contains information about a single DC in the domain.

```

typedef struct {
    [string, unique] WCHAR* NetbiosName;
    [string, unique] WCHAR* DnsHostName;
    [string, unique] WCHAR* SiteName;
    [string, unique] WCHAR* ComputerObjectName;
    [string, unique] WCHAR* ServerObjectName;
    BOOL fIsPdc;
    BOOL fDsEnabled;
} DS_DOMAIN_CONTROLLER_INFO_1W;

```

NetbiosName: NetBIOS name of the DC.

DnsHostName: DNS host name of the DC.

SiteName: RDN of the [site](#) object.

ComputerObjectName: DN of the [computer](#) object that corresponds to the DC.

ServerObjectName: DN of the [server](#) object that corresponds to the DC.

fIsPdc: True if and only if the DC is the PDC FSMO role owner.

fDsEnabled: MUST be true.

4.1.5.1.9 DS_DOMAIN_CONTROLLER_INFO_2W

The **DS_DOMAIN_CONTROLLER_INFO_2W** structure defines DC information that is returned as a part of the response to an InfoLevel = 2 request. The struct contains information about a single DC in the domain.

```

typedef struct {
    [string, unique] WCHAR* NetbiosName;
    [string, unique] WCHAR* DnsHostName;
    [string, unique] WCHAR* SiteName;
    [string, unique] WCHAR* SiteObjectName;
    [string, unique] WCHAR* ComputerObjectName;
    [string, unique] WCHAR* ServerObjectName;
    [string, unique] WCHAR* NtdsDsaObjectName;
    BOOL fIsPdc;
}

```

```

    BOOL fDsEnabled;
    BOOL fIsGc;
    GUID SiteObjectGuid;
    GUID ComputerObjectGuid;
    GUID ServerObjectGuid;
    GUID NtdsDsaObjectGuid;
} DS_DOMAIN_CONTROLLER_INFO_2W;

```

NetbiosName: NetBIOS name of the DC.

DnsHostName: DNS host name of the DC.

SiteName: RDN of the [site](#) object.

SiteObjectName: DN of the [site](#) object.

ComputerObjectName: DN of the [computer](#) object that corresponds to the DC.

ServerObjectName: DN of the [server](#) object that corresponds to the DC.

NtdsDsaObjectName: DN of the [nTDSDSA](#) object that corresponds to the DC.

fIsPdc: True if and only if the DC is the PDC FSMO role owner.

fDsEnabled: MUST be true.

fIsGc: True if and only if the DC is also a GC.

SiteObjectGuid: The [objectGUID](#) attribute of the [site](#) object.

ComputerObjectGuid: The [objectGUID](#) attribute of the [computer](#) object that corresponds to the DC.

ServerObjectGuid: The [objectGUID](#) attribute of the [server](#) object that corresponds to the DC.

NtdsDsaObjectGuid: The [objectGUID](#) attribute of the [nTDSDSA](#) object that corresponds to the DC.

4.1.5.1.10 DS_DOMAIN_CONTROLLER_INFO_3W

The **DS_DOMAIN_CONTROLLER_INFO_3W** structure defines DC information that is returned as a part of the response to an InfoLevel = 3 request. The struct contains information about a single DC in the domain.

```

typedef struct {
    [string, unique] WCHAR* NetbiosName;
    [string, unique] WCHAR* DnsHostName;
    [string, unique] WCHAR* SiteName;
    [string, unique] WCHAR* SiteObjectName;
    [string, unique] WCHAR* ComputerObjectName;
    [string, unique] WCHAR* ServerObjectName;
    [string, unique] WCHAR* NtdsDsaObjectName;
    BOOL fIsPdc;
    BOOL fDsEnabled;
    BOOL fIsGc;
}

```

```

    BOOL fIsRdc;
    GUID SiteObjectGuid;
    GUID ComputerObjectGuid;
    GUID ServerObjectGuid;
    GUID NtdsDsaObjectGuid;
} DS_DOMAIN_CONTROLLER_INFO_3W;

```

NetbiosName: NetBIOS name of the DC.

DnsHostName: DNS host name of the DC.

SiteName: RDN of the [site](#) object.

SiteObjectName: DN of the [site](#) object.

ComputerObjectName: DN of the [computer](#) object that corresponds to the DC.

ServerObjectName: DN of the [server](#) object that corresponds to the DC.

NtdsDsaObjectName: DN of the [nTDSDSA](#) object that corresponds to the DC.

fIsPdc: True if and only if the DC is the PDC FSMO role owner.

fDsEnabled: MUST be true.

fIsGc: True if and only if the DC is also a GC.

fIsRdc: True if and only if the DC is an RODC.

SiteObjectGuid: [objectGUID](#) of the [site](#) object.

ComputerObjectGuid: [objectGUID](#) of the [computer](#) object that corresponds to the DC.

ServerObjectGuid: [objectGUID](#) of the [server](#) object that corresponds to the DC.

NtdsDsaObjectGuid: [objectGUID](#) of the [nTDSDSA](#) object that corresponds to the DC.

4.1.5.1.11 DS_DOMAIN_CONTROLLER_INFO_FFFFFFFFW

The **DS_DOMAIN_CONTROLLER_INFO_FFFFFFFFW** structure defines DC information that is returned as a part of the response to an InfoLevel = 0xFFFFFFFF request. The struct contains information about a single LDAP connection to the current server.

```

typedef struct {
    DWORD IPAddress;
    DWORD NotificationCount;
    DWORD secTimeConnected;
    DWORD Flags;
    DWORD TotalRequests;
    DWORD Reserved1;
    [string, unique] WCHAR* UserName;
} DS_DOMAIN_CONTROLLER_INFO_FFFFFFFFW;

```

IPAddress: The IPv4 address of the client that established the LDAP connection to the server. If the client connected with IPv6, this field contains zero.

NotificationCount: Number of LDAP notifications enabled on the server.

secTimeConnected: Total time in number of seconds that the connection is established.

Flags: Zero or more of the bit flags from [LDAP_CONN_PROPERTIES](#) indicating the properties of this connection.

TotalRequests: Total number of LDAP requests made on this LDAP connection.

Reserved1: MUST be 0.

UserName: Name of the security principal that established the LDAP connection.

4.1.5.2 Server Behavior of the IDL_DRSDomainControllerInfo Method

Informative summary of behavior: The [IDL_DRSDomainControllerInfo](#) method supports four information levels. For levels 1, 2, and 3, the server returns information for the domain controllers (DCs) in the domain of the server. For level 0xffffffff, the server returns information about the LDAP connections on the server that are currently open.

Regular read access checks apply to the information that is returned to the caller. Therefore, if the caller does not have read permission on data that needs to be returned, this data is not included in the response. See [\[MS-ADTS\]](#) section 3.1.1.4.3 for more information about access check behavior in read operations.

```
ULONG
IDL_DRSDomainControllerInfo(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_DCINFOREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)] DRS_MSG_DCINFOREPLY *pmsgOut)

msgIn: DRS_MSG_DCINFOREQ_V1
infoLevel, i: integer
domainName: unicodestring
dcSet: set of DSName
crObj, dc, dsaObj, svrObj, siteObj, obj, v: DSName
lc: DS_DOMAIN_CONTROLLER_INFO_FFFFFFFFW
rI1: ADDRESS OF DS_DOMAIN_CONTROLLER_INFO_1W
rI2: ADDRESS OF DS_DOMAIN_CONTROLLER_INFO_2W
rI3: ADDRESS OF DS_DOMAIN_CONTROLLER_INFO_3W
found: boolean
crackMsgIn: DRS_MSG_CRACKREQ
crackMsgOut: DRS_MSG_CRACKREPLY
outV: DWORD
userAccountControl: set of integer

msgIn := pmsgIn^.V1
infoLevel := msgIn.InfoLevel
domainName := msgIn.Domain

if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
```



```

endif

if not (infoLevel in {1,2,3,0xFFFFFFFF}) then
    return ERROR_INVALID_PARAMETER
endif

pdwOutVersion^ := infoLevel

if infoLevel = 0xFFFFFFFF then
    /* Enumerate the LDAP connections. */
    if not IsMemberOfBuiltinAdminGroup() then
        return ERROR_ACCESS_DENIED
    endif

    pmsgOut^.VFFFFFFFF.cItems := number(dc.ldapConnections)

    i := 0
    foreach lc in dc.ldapConnections
        pmsgOut^.VFFFFFFFF.rItems[i].IPAddress := lc.iPAddress
        pmsgOut^.VFFFFFFFF.rItems[i].NotificationCount :=
            lc.notificationCount
        pmsgOut^.VFFFFFFFF.rItems[i].secTimeConnected :=
            lc.secTimeConnected
        pmsgOut^.VFFFFFFFF.rItems[i].Flags := lc.flags
        pmsgOut^.VFFFFFFFF.rItems[i].TotalRequests := lc.totalRequests
        pmsgOut^.VFFFFFFFF.rItems[i].UserName := lc.userName
        pmsgOut^.VFFFFFFFF.rItems[i].Reserved1 := 0

        i := i + 1
    endfor

    return 0
endif

/* Verify that the given domain name matches the default domain NC.
 * First check if it is the nETBiosName or dNSHostName of the default
 * domain NC by searching for the crossRef object, then call
 * IDL_DRSCrackNames to check if the given domain name is a name
 * for the default domain NC. */

crObj := select one v from children
    DescendantObject(ConfigNC(), "CN=Partitions,")
    where
        (v!dNSHostName = domainName or v!nETBiosName = domainName)
        and
        v!nCName = DefaultNC()

found := (crObj ≠ null)

if not found then
    /* Not found; use IDL_DRSCrackNames to resolve the name. */
    crackMsgIn.V1.formatOffered := DS_UNKNOWN_NAME
    crackMsgIn.V1.formatDesired := DS_FQDN_1779_NAME
    crackMsgIn.V1.cNames := 3
    crackMsgIn.V1.rpNames[0] := domainName
    crackMsgIn.V1.rpNames[1] := domainName + "\"
    crackMsgIn.V1.rpNames[2] := domainName + "/"

```

```

/* Call IDL_DRSCrackNames as a local procedure. */
IDL_DRSCrackNames(null, 1, crackMsgIn, ADR(outV), ADR(crackMsgOut))

i := 0
while i < 3 and not found
    if crackMsgOut.V1.pResult.rItems[i].status = DS_NAME_NO_ERROR
        then
            if crackMsgOut.V1.pResult.rItems[i].pName = DefaultNC().dn
                then
                    found := true
            else
                return ERROR_INVALID_PARAMETER
            endif
        endif
        i := i + 1
    endwhile
endif

if not found then
    return ERROR_DS_OBJ_NOT_FOUND
endif

/* Enumerate the DCs in the domain. */
if infoLevel = 3 then
    /* client requests to return RODCs too */
    userAccountControl :=
        {ADS_UF_SERVER_TRUST_ACCOUNT, ADS_UF_PARTIAL_SECRETS_ACCOUNT}
else
    userAccountControl := {ADS_UF_SERVER_TRUST_ACCOUNT}
endif

dcSet := select all v from subtree DefaultNC() where
    v!objectCategory = GetDefaultObjectCategory(computer)
    and (userAccountControl ∩ v!userAccountControl ≠ null)

if infoLevel = 1 then
    pmsgOut^.V1.cItems := number(dcSet)

    i := 0
    foreach dc in dcSet
        rI1 := ADR(pmsgOut^.V1.rItems[i])

        rI1^.DnsHostName := dc!dnsHostName
        rI1^.ComputerObjectName := dc.dn
        /* sAMAccountName excluding the "$" at the end. */
        rI1^.NetbiosName := SubString(dc!sAMAccountName, 0,
            dc!samAccountName.length-1)
        rI1^.fDsEnabled := true

        /* select a server object from the serverReferenceBL, it is
           preferred that the server object has a child object with
           CN "NTDS Settings" */
        srvObj :=
            select one v from all where v.dn in dc!serverReferenceBL
            and DescendentObject(v, "CN=NTDS Settings") ≠ null
        if srvObj = null then
            srvObj :=
                select one v from all where v.dn in dc!serverReferenceBL

```

```

endif
if svrObj ≠ null then
    rI1^.ServerObjectName := svrObj.dn
    siteObj :=
        select one o from all where o!objectGUID = svrObj!parent
    rI1^.SiteObjectName := siteObj.dn
    dsaObj := DescendantObject(v, "CN=NTDS Settings,")
    rI1^.fIsPdc := (dsaObj = GetFSMORoleOwner(FSMO_PDC))
endif
i := i + 1
endfor
else
    if infoLevel = 2 then
        pmsgOut^.V2.cItems := number(dcSet)

        i := 0
        foreach dc in dcSet
            rI2 := ADR(pmsgOut^.V2.rItems[i])

            rI2^.DnsHostName := dc!dNSHostName
            rI2^.ComputerObjectName := dc.dn
            /* sAMAccountName excluding the "$" at the end. */
            rI2^.NetbiosName := SubString(dc!samAccountName, 0,
                dc!samAccountName.length-1)
            rI2^.ComputerObjectGUID := dc.guid
            rI2^.fDsEnabled := true

            /* select a server object from the serverReferenceBL, it is
               preferred that the server object has a child object with
               CN "NTDS Settigs" */
            srvObj :=
                select one v from all where v.dn in dc!serverReferenceBL
                    and DescendentObject(v, "CN=NTDS Settings") ≠ null
            if srvObj = null then
                srvObj :=
                    select one v from all where v.dn in dc!serverReferenceBL
            endif
            if svrObj ≠ null then
                rI2^.ServerObjectName := svrObj.dn
                rI2^.ServerObjectGuid := svrObj.guid
                siteObj :=
                    select one o from all where o!objectGUID = svrObj!parent
                rI2^.SiteObjectName := siteObj.dn
                rI2^.SiteObjectGUID := siteObj.guid
                dsaObj := DescendantObject(v, "CN=NTDS Settings,")
                rI2^.NtdsDsaObjectGUID := dsaObj.guid
                rI2^.fIsGc := (NTDSDSA_OPT_IS_GC in dsaObj!options)
                rI2^.fIsPdc := (dsaObj = GetFSMORoleOwner(FSMO_PDC))
            endif
            i := i + 1
        endfor
    else
        /* infoLevel = 3 */
        pmsgOut^.V2.cItems := number(dcSet)

        i := 0
        foreach dc in dcSet

```

```

rI3 := ADR(pmsgOut^.V2.rItems[i])

rI3^.DnsHostName := dc!dNSHostName
rI3^.ComputerObjectName := dc.dn
/* sAMAccountName excluding the "$" at the end. */
rI3^.NetbiosName := SubString(dc!samAccountName, 0,
    dc!samAccountName.length-1)
rI3^.ComputerObjectGUID := dc.guid
rI3^.fDsEnabled := true

/* select a server object from the serverReferenceBL, it is
   preferred that the server object has a child object with
   CN "NTDS Settigs" */
srvObj :=
    select one v from all where v.dn in dc!serverReferenceBL
        and DescendentObject(v, "CN=NTDS Settings") ≠ null
if srvObj = null then
    srvObj :=
        select one v from all where v.dn in dc!serverReferenceBL
endif
if svrObj ≠ null then
    rI3^.ServerObjectName := svrObj.dn
    rI3^.ServerObjectGuid := svrObj.guid
    siteObj :=
        select one o from all where o!objectGUID = svrObj!parent
    rI3^.SiteObjectName := siteObj.dn
    rI3^.SiteObjectGUID := siteObj.guid
    dsaObj := DescendantObject(v, "CN=NTDS Settings,")
    rI3^.NtdsDsaObjectGUID := dsaObj.guid
    rI3^.fIsGC := (NTDSDSA_OPT_IS_GC in dsaObj!options)
    rI3^.fIsPDC := (dsaObj = GetFSMORoleOwner(FSMO_PDC))
    Ri3^.fIsRdc := ((ADS_UF_PARTIAL_SECRETS_ACCOUNT ∩
        dc!userAccountControl) ≠ null)
endif
i := i + 1
endfor
endif
endif
return 0

```

4.1.5.3 Examples of the IDL_DRSDomainControllerInfo Method

An application running on DC2 invokes the [DRSDomainControllerInfo](#) method on DC2 to retrieve the NetBIOS and DNS host names for all DCs in the domain NC CONTOSO.COM

4.1.5.3.1 Initial State

Querying the [crossRef](#) object on the NC root object for domain NC CONTOSO.COM on DC2:

- Expanding base 'CN=CONTOSO,CN=Partitions,CN=Configuration,DC=contoso,DC=com'...
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:

- >> Dn: CN=CONTOSO,CN=Partitions,CN=Configuration,DC=contoso,DC=com
 - 2> objectClass: top; crossRef;
 - 1> nCName: DC=contoso,DC=com;
 - 1> dnsRoot: contoso.com;
 - 1> nETBIOSName: CONTOSO;

Querying the DC1 [computer](#) object in domain NC DC=CONTOSO, DC=COM

- Expanding base 'CN=DC1,OU=Domain Controllers,DC=contoso,DC=com'...
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
 - >> Dn: CN=DC1,OU=Domain Controllers,DC=contoso,DC=com
 - 5> objectClass: top; person; organizationalPerson; user; computer;
 - 1> cn: DC1;
 - 1> distinguishedName: CN=DC1, OU=Domain Controllers, DC=contoso, DC=com;
 - 1> instanceType: 0x4 = (IT_WRITE);
 - 1> whenCreated: 07/10/2006 18:04:35 Pacific Standard Daylight Time;
 - 1> whenChanged: 07/15/2006 19:39:05 Pacific Standard Daylight Time;
 - 1> uSNCreated: 12291;
 - 1> uSNChanged: 24577;
 - 1> name: DC1;
 - 1> objectGUID: ac1993e1-0377-4161-893e-ccd2a98e1bba;
 - 1> userAccountControl: (UF_SERVER_TRUST_ACCOUNT | UF_TRUSTED_FOR_DELEGATION);
 - 1> badPwdCount: 0;
 - 1> codePage: 0;
 - 1> countryCode: 0;
 - 1> badPasswordTime: 01/01/1601 00:00:00 UNC ;
 - 1> lastLogoff: 01/01/1601 00:00:00 UNC ;
 - 1> lastLogon: 07/17/2006 19:47:40 Pacific Standard Daylight Time;
 - 1> localPolicyFlags: 0;
 - 1> pwdLastSet: 07/10/2006 18:04:35 Pacific Standard Daylight Time;

- 1> primaryGroupID: 516;
- 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1001;
- 1> accountExpires: 09/14/30828 02:48:05 UNC ;
- 1> logonCount: 17;
- 1> sAMAccountName: DC1\$;
- 1> sAMAccountType: 805306369;
- 1> operatingSystem: Windows Server 2003;
- 1> operatingSystemVersion: 5.2 (3790);
- 1> operatingSystemServicePack: Service Pack 1;
- 1> serverReferenceBL: CN=DC1,CN=Servers, CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com;
- 1> dNSHostName: DC1.contoso.com;
- 1> rIDSetReferences: CN=RID Set,CN=DC1,OU=Domain Controllers, DC=contoso, DC=com;
- 15> servicePrincipalName: ldap/DC1.contoso.com/NDNC5.contoso.com;
ldap/DC1.contoso.com/NDNC2.contoso.com; ldap/DC1.contoso.com/NDNC1.contoso.com;
GC/DC1.contoso.com/contoso.com; HOST/DC1.contoso.com/CONTOSO; HOST/DC1;
HOST/DC1.contoso.com; HOST/DC1.contoso.com/contoso.com; E3514235-4B06-11D1-AB04-
00C04FC2DCD2/c20bc312-4d35-4cc0-9903-b1073368af4a/contoso.com; ldap/c20bc312-
4d35-4cc0-9903-b1073368af4a._msdcs.contoso.com; ldap/DC1.contoso.com/CONTOSO;
ldap/DC1; ldap/DC1.contoso.com; ldap/DC1.contoso.com/contoso.com; NtFrs-88f5d2bd-
b646-11d2-a6d3-00c04fc9b232/DC1.contoso.com;
- 1> objectCategory: CN=Computer, CN=Schema, CN=Configuration, DC=contoso, DC=com;
- 1> isCriticalSystemObject: TRUE;
- 1> frsComputerReferenceBL: CN=DC1, CN=Domain System Volume (SYSVOL share),CN=File
Replication Service,CN=System,DC=contoso,DC=com;
- 1> lastLogonTimestamp: 07/11/2006 04:02:42 Pacific Std Daylight Time;

Querying the DC1 [computer](#) object in domain NC DC=CONTOSO, DC=COM

- Expanding base 'CN=DC2,OU=Domain Controllers,DC=contoso,DC=com'...
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=DC2,OU=Domain Controllers,DC=contoso,DC=com
 - 5> objectClass: top; person; organizationalPerson; user; computer;
 - 1> cn: DC2;

- 1> distinguishedName: CN=DC2, OU=Domain Controllers, DC=contoso, DC=com;
- 1> instanceType: 0x4 = (IT_WRITE);
- 1> whenCreated: 07/10/2006 18:12:01 Pacific Standard Daylight Time;
- 1> whenChanged: 07/16/2006 13:46:14 Pacific Standard Daylight Time;
- 1> displayName: DC2\$;
- 1> uSNCreated: 13711;
- 1> uSNChanged: 28819;
- 1> name: DC2;
- 1> objectGUID: 09697f46-2458-4b26-a4e9-aa36059421c4;
- 1> userAccountControl: (UF_SERVER_TRUST_ACCOUNT | UF_TRUSTED_FOR_DELEGATION);
- 1> badPwdCount: 0;
- 1> codePage: 0;
- 1> countryCode: 0;
- 1> badPasswordTime: 01/01/1601 00:00:00 UNC ;
- 1> lastLogoff: 01/01/1601 00:00:00 UNC ;
- 1> lastLogon: 07/17/2006 20:38:08 Pacific Standard Daylight Time;
- 1> localPolicyFlags: 0;
- 1> pwdLastSet: 07/10/2006 18:12:02 Pacific Standard Daylight Time;
- 1> primaryGroupID: 516;
- 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1102;
- 1> accountExpires: 09/14/30828 02:48:05 UNC ;
- 1> logonCount: 8;
- 1> sAMAccountName: DC2\$;
- 1> sAMAccountType: 805306369;
- 1> operatingSystem: Windows Server 2003;
- 1> operatingSystemVersion: 5.2 (3790);
- 1> operatingSystemServicePack: Service Pack 1;
- 1> serverReferenceBL: CN=DC2,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com;
- 1> dNSHostName: DC2.contoso.com;
- 1> rIDSetReferences: CN=RID Set,CN=DC2,OU=Domain Controllers, DC=contoso, DC=com;

- 14> servicePrincipalName: ldap/DC2.contoso.com/NDNC5.contoso.com;
ldap/DC2.contoso.com/NDNC2.contoso.com; ldap/6aad8f5a-07cc-403a-9696-9102fe1c320b._msdcs.contoso.com; ldap/DC2.contoso.com/CONTOSO; ldap/DC2;
ldap/DC2.contoso.com; ldap/DC2.contoso.com/contoso.com; NtFrs-88f5d2bd-b646-11d2-a6d3-00c04fc9b232/DC2.contoso.com; HOST/DC2.contoso.com/CONTOSO;
HOST/DC2.contoso.com/contoso.com; C/DC2.contoso.com/contoso.com; E3514235-4B06-11D1-AB04-00C04FC2DCD2/6aad8f5a-07cc-403a-9696-9102fe1c320b/contoso.com;
- HOST/DC2; HOST/DC2.contoso.com;
- 1> objectCategory: CN=Computer, CN=Schema, CN=Configuration, DC=contoso, DC=com;
- 1> isCriticalSystemObject: TRUE;
- 1> frsComputerReferenceBL: CN=DC2,CN=Domain System Volume (SYSVOL share),CN=File Replication Service,CN=System,DC=contoso,DC=com;
- 4> dScorePropagationData: 07/10/2006 18:14:51 Pacific Standard Daylight Time;
07/10/2006 18:14:51 Pacific Standard Time Pacific Daylight Time; 07/10/2006 18:14:51
Pacific Standard Time Pacific Daylight Time; 01/08/1601 07:15:13 Pacific Standard Time
Pacific Daylight Time;
- 1> lastLogonTimestamp: 07/10/2006 19:52:48 Pacific Std Daylight Time;

4.1.5.3.2 Client Request

DC2 invokes the [IDL_DRSDomainControllerInfo](#) method against itself with the following parameters ([DRS_HANDLE](#) to DC2 is omitted):

- dwInVersion = 1
- pmsgIn = DRS_MSG_DCINFOREQ_V1
 - Domain = "contoso.com"
 - InfoLevel = 1

4.1.5.3.3 Server Response

Return code of 0 and the following values:

- pdwOutVersion^ = 1
- pmsgOut = DRS_MSG_DCINFOREPLY_V1
 - cItems: 2
 - rItems[0]: DS_DOMAIN_CONTROLLER_INFO_1W
 - NetbiosName: "DC1"
 - DnsHostName: "DC1.contoso.com"
 - SiteName: "Default-First-Site-Name"
 - ComputerObjectName: "CN=DC1, OU=Domain Controllers,DC=contoso,DC=com"

- ServerObjectName: "CN=DC1,CN=Servers, CN=Default-First-Site-Name,CN=Sites, CN=Configuration, DC=contoso,DC=com"
- fIsPdc: 1
- fIsDisabled: 1
- rItems[1]: DS_DOMAIN_CONTROLLER_INFO_1W
 - NetbiosName: "DC2"
 - DnsHostName: "DC2.contoso.com"
 - SiteName: "Default-First-Site-Name"
 - ComputerObjectName: "CN=DC2, OU=Domain Controllers,DC=contoso,DC=com"
 - ServerObjectName: "CN=DC2,CN=Servers, CN=Default-First-Site-Name,CN=Sites, CN=Configuration, DC=contoso,DC=com"
 - fIsPdc: 0
 - fIsDisabled: 1

4.1.5.3.4 Final State

The final state is the same as the initial state; there is no change.

4.1.6 IDL_DRSExecuteKCC (Opnum 18)

The **IDL_DRSExecuteKCC** method validates the replication interconnections of DCs and updates them if necessary.

```
ULONG IDL_DRSExecuteKCC(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
    DRS_MSG_KCC_EXECUTE* pmsgIn
);
```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.6.1 Method-Specific Concrete Types

4.1.6.1.1 DRS_MSG_KCC_EXECUTE

The **DRS_MSG_KCC_EXECUTE** union defines the request messages sent to the [IDL DRSExecuteKCC](#) method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_KCC_EXECUTE_V1 V1;
} DRS_MSG_KCC_EXECUTE;
```

V1: Version 1 request.

4.1.6.1.2 DRS_MSG_KCC_EXECUTE_V1

The **DRS_MSG_KCC_EXECUTE_V1** structure defines the request message sent to the [IDL DRSExecuteKCC](#) method.

```
typedef struct {
    DWORD dwTaskID;
    DWORD dwFlags;
} DRS_MSG_KCC_EXECUTE_V1;
```

dwTaskID: MUST be 0.

dwFlags: Zero or more of the following bit flags:

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
A	D	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	P																														

X: Unused. MUST be zero and ignored.

AS (DS_KCC_FLAG_ASYNC_OP): Request the KCC to run, then return immediately.

DP (DS_KCC_FLAG_DAMPED): Request the KCC to run unless there is already such a request pending.

4.1.6.2 Method-Specific Abstract Types and Procedures

4.1.6.2.1 ExecuteKCCTasks

```
procedure ExecuteKCCTasks(): ULONG
```

This procedure executes the tasks necessary for maintaining the replication topology between DCs.

If an error occurs, a Windows error code is returned. If successful, the method returns 0.

4.1.6.3 Server Behavior of the IDL_DRSExecuteKCC Method

Informative summary of behavior: The [IDL_DRSExecuteKCC](#) method triggers the execution of tasks that generate and maintain the replication topology between DCs. [<10>](#)

```
ULONG
IDL_DRSExecuteKCC(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_KCC_EXECUTE *pmsgIn)

msgIn: DRS_MSG_KCC_EXECUTE_V1

/* Validate the request version */
if dwInVersion != 1 then
    return ERROR_INVALID_PARAMETER
endif

msgIn := pmsgIn^.V1

if msgIn.dwTaskID != 0 then
    return ERROR_INVALID_PARAMETER
endif

if not AccessCheckCAR(ConfigNC(), DS-Replication-Manage-Topology)
    then
        return ERROR_DS_DRA_ACCESS_DENIED
    endif

return ExecuteKCCTasks()
```

4.1.7 IDL_DRSFinishDemotion (Opnum 27)

The **IDL_DRSFinishDemotion** method either performs one or more steps toward the complete removal of a DC from an AD/LDS forest, or it undoes the effects of the first phase of removal (performed by [IDL_DRSInitDemotion](#)). This method is supported by AD/LDS only.

```
ULONG IDL_DRSFinishDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_FINISH_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_FINISH_DEMOTIONREPLY* pmsgOut
);
```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.7.1 Method-Specific Concrete Types

4.1.7.1.1 DRS_MSG_FINISH_DEMOTIONREQ

The **DRS_MSG_FINISH_DEMOTIONREQ** union defines the request messages sent to the [IDL DRSFinishDemotion](#) method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_FINISH_DEMOTIONREQ_V1 V1;
} DRS_MSG_FINISH_DEMOTIONREQ;
```

V1: Version 1 request. Currently, only one version is defined.

4.1.7.1.2 DRS_MSG_FINISH_DEMOTIONREQ_V1

The **DRS_MSG_FINISH_DEMOTIONREQ_V1** structure defines the request message sent to the [IDL DRSFinishDemotion](#) method.

```
typedef struct {
    DWORD dwOperations;
    UUID uuidHelperDest;
    LPWSTR szScriptBase;
} DRS_MSG_FINISH_DEMOTIONREQ_V1;
```

dwOperations: Zero or more of the following bit flags:

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
R	C	D	U	U	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	F
			1	2																											

X: Unused. MUST be zero and ignored.

R (DS_DEMOTE_ROLLBACK_DEMOTE): Undo the effects of [IDL DRSInitDemotion](#). If present, any other flags present are ignored.

C (DS_DEMOTE_COMMIT_DEMOTE): Mark the DC's database "demotion-complete" (this effect is outside the state model).

D (DS_DEMOTE_DELETE_CSMETA): Delete the [nTDSDSA](#) object for this DC; see [RemoveADLDSServer \(section 4.1.7.2.1\)](#).

U1 (DS_DEMOTE_UNREGISTER_SCPS): Delete any [serviceConnectionPoint](#) objects for this DC from AD/DS; see [RemoveADLDSSCP \(section 4.1.7.2.2\)](#).

U2 (DS_DEMOTE_UNREGISTER_SPNS): Delete any AD/LDS SPNs from the service account object in AD/DS; see [RemoveADLDSSPNs \(section 4.1.7.2.3\)](#).

F (DS_DEMOTE_OPT_FAIL_ON_UNKNOWN_OP): Fail the request if the dwOperations field contains an unknown flag.

uuidHelperDest: MUST be NULL GUID.

szScriptBase: The path name of the folder in which to store SPN unregistration scripts. Required when DS_DEMOTE_UNREGISTER_SPNS is specified in dwOperations.

4.1.7.1.3 DRS_MSG_FINISH_DEMOTIONREPLY

The **DRS_MSG_FINISH_DEMOTIONREPLY** union defines the response messages received from the [IDL DRSFinishDemotion](#) method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_FINISH_DEMOTIONREPLY_V1 V1;
} DRS_MSG_FINISH_DEMOTIONREPLY;
```

V1: Version 1 reply.

4.1.7.1.4 DRS_MSG_FINISH_DEMOTIONREPLY_V1

The **DRS_MSG_FINISH_DEMOTIONREPLY_V1** structure defines the response message received from the [IDL DRSFinishDemotion](#) method.

```
typedef struct {
    DWORD dwOperationsDone;
    DWORD dwOpFailed;
    DWORD dwOpError;
} DRS_MSG_FINISH_DEMOTIONREPLY_V1;
```

dwOperationsDone: The set of operations that were successfully performed. This may include the following values: DS_DEMOTE_ROLLBACK_DEMOTE, DS_DEMOTE_COMMIT_DEMOTE, DS_DEMOTE_DELETE_CSMETA, DS_DEMOTE_UNREGISTER_SCPS, DS_DEMOTE_UNREGISTER_SPNS.

dwOpFailed: The set of operations that failed during demotion. This may include the same values as the dwOperationsDone field.

dwOpError: The Win32 error code (as specified in [\[MS-ERREF\]](#) section 2.2) of the first failed operation (if any).

4.1.7.2 Method-Specific Abstract Types and Procedures

4.1.7.2.1 RemoveADLDSServer

```
procedure RemoveADLDSServer(): DWORD
```

The RemoveADLDSServer procedure connects to any available replication partner and uses the [IDL_DRSRemoveDsServer](#) method to delete the [nTDSDSA](#) object that corresponds to this DC. If no such [nTDSDSA](#) object exists or if the deletion is successful, RemoveADLDSServer returns ERROR_SUCCESS; otherwise, it returns a Win32 error.

4.1.7.2.2 RemoveADLDSSCP

```
procedure RemoveADLDSSCP(): DWORD
```

The RemoveADLDSSCP procedure connects to an AD/DS DC and deletes any [serviceConnectionPoint](#) object that was created in AD/DS for this AD/LDS DC. See [\[MS-ADTS\]](#) section 7.3.8 for more information on AD/LDS [serviceConnectionPoint](#) objects. If no such [serviceConnectionPoint](#) object exists or if the deletion is successful, RemoveADLDSSCP returns ERROR_SUCCESS; otherwise, it returns a Win32 error.

4.1.7.2.3 RemoveADLDSSPNs

```
procedure RemoveADLDSSPNs(szScriptBase: unicodestring): DWORD
```

The RemoveADLDSSPNs procedure connects to an AD/DS DC and deletes any SPN values registered for the AD/LDS DC on its service account object in AD/DS. Sections [2.2.3.2](#) and [2.2.4.2](#) specify the SPN values removed by this procedure.

If no such SPN values exist or the deletion is successful, RemoveADLDSSPNs returns ERROR_SUCCESS; otherwise, it returns a Win32 error and writes a batch file in the folder specified by the szScriptBase parameter. This batch file contains commands that an administrator can run to clean up the SPNs.

4.1.7.3 Server Behavior of the IDL_DRSFinishDemotion Method

Informative summary of behavior: The [IDL_DRSFinishDemotion](#) method either performs one or more steps toward the complete removal of a DC from an AD/LDS forest, or undoes the effects of the first phase of removal (performed by IDL_DRSInitDemotion).

```
ULONG  
IDL_DRSFinishDemotion(  
    [in, ref] DRS_HANDLE hDrs,  
    [in] DWORD dwInVersion,  
    [in, ref, switch_is(dwInVersion)]
```

```

        DRS_MSG_FINISH_DEMOTIONREQ* pmsgIn,
        [out, ref] DWORD *pdwOutVersion,
        [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_FINISH_DEMOTIONREPLY* pmsgOut
    )

msgIn: DRS_MSG_FINISH_DEMOTIONREQ_V1
msgOut: DRS_MSG_FINISH_DEMOTIONREPLY_V1
ret: DWORD

if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
if DS_DEMOTE_OPT_FAIL_ON_UNKNOWN_OP in msgIn.dwOperations
    and msgIn.dwOperations - { DS_DEMOTE_COMMIT_DEMOTE,
        DS_DEMOTE_DELETE_CSMETA, DS_DEMOTE_UNREGISTER_SCPS,
        DS_DEMOTE_UNREGISTER_SPNS, DS_DEMOTE_OPT_FAIL_ON_UNKNOWN_OP }
    ≠ null then
        /* unknown operation bit is set */
        return ERROR_INVALID_PARAMETER
    endif
if DS_DEMOTE_UNREGISTER_SPNS in msgIn.dwOperations
    and msgIn.szScriptBase = null then
        /* szScriptBase must be specified when UNREGISTER_SPN is
        * requested */
        return ERROR_INVALID_PARAMETER
    endif
if not IsMemberOfBuiltinAdminGroup() then
    /* only BA is allowed to demote an AD/LDS service */
    return ERROR_ACCESS_DENIED
endif

msgOut.dwOperationDone := 0
msgOut.dwOpFailed := 0
msgOut.dwOpError := ERROR_SUCCESS

if DS_DEMOTE_ROLLBACK_DEMOTE in msgIn.dwOperations then
    /* undo the effects of IDL_DRSInitDemotion */
    ret := Mark database as read-write
    if ret = ERROR_SUCCESS then
        ret := Enable originating and replicated updates
    endif
    if ret = ERROR_SUCCESS then
        msgOut.dwOperationDone :=
            msgOut.dwOperationDone + {DS_DEMOTE_ROLLBACK_DEMOTE}
    else
        msgOut.dwOpFailed =
            msgOut.dwOpFailed + {DS_DEMOTE_ROLLBACK_DEMOTE}
        if msgOut.dwOpError = ERROR_SUCCESS then
            msgOut.dwOpError := ret
        endif
    endif
    /* no other operations are allowed on rollback */
else
    if DS_DEMOTE_COMMIT_DEMOTE in msgIn.dwOperations then
        ret := Mark database as demotion-complete
        if ret = ERROR_SUCCESS then

```

```

        msgOut.dwOperationDone :=
            msgOut.dwOperationDone + {DS_DEMOTE_COMMIT_DEMOTE}
    else
        msgOut.dwOpFailed =
            msgOut.dwOpFailed + {DS_DEMOTE_COMMIT_DEMOTE}
        if msgOut.dwOpError = ERROR_SUCCESS then
            msgOut.dwOpError := ret
        endif
    endif
endif
endif
if DS_DEMOTE_DELETE_CSMETA in msgIn.dwOperations then
    ret := RemoveADLDSServer()
    if ret = ERROR_SUCCESS then
        msgOut.dwOperationDone :=
            msgOut.dwOperationDone + {DS_DEMOTE_DELETE_CSMETA}
    else
        msgOut.dwOpFailed =
            msgOut.dwOpFailed + {DS_DEMOTE_DELETE_CSMETA}
        if msgOut.dwOpError = ERROR_SUCCESS then
            msgOut.dwOpError := ret
        endif
    endif
endif
endif
if DS_DEMOTE_UNREGISTER_SCPS in msgIn.dwOperations then
    ret := RemoveADLDSSCP()
    if ret = ERROR_SUCCESS then
        msgOut.dwOperationDone :=
            msgOut.dwOperationDone + {DS_DEMOTE_UNREGISTER_SCPS}
    else
        msgOut.dwOpFailed =
            msgOut.dwOpFailed + {DS_DEMOTE_UNREGISTER_SCPS}
        if msgOut.dwOpError = ERROR_SUCCESS then
            msgOut.dwOpError := ret
        endif
    endif
endif
endif
if DS_DEMOTE_UNREGISTER_SPNS in msgIn.dwOperations then
    ret := RemoveADLDSSPNs(msgIn.szScriptBase)
    if ret = ERROR_SUCCESS then
        msgOut.dwOperationDone :=
            msgOut.dwOperationDone + {DS_DEMOTE_UNREGISTER_SPNS}
    else
        msgOut.dwOpFailed =
            msgOut.dwOpFailed + {DS_DEMOTE_UNREGISTER_SPNS}
        if msgOut.dwOpError = ERROR_SUCCESS then
            msgOut.dwOpError := ret
        endif
    endif
endif
endif
pmsgOut^ := msgOut
pdwMsgOut^ := 1
return ERROR_SUCCESS

```

4.1.8 IDL_DRSGetMemberships (Opnum 9)

The **IDL_DRSGetMemberships** method retrieves group membership for an object.


```

ULONG IDL_DRSGetMemberships(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_REVMEMB_REQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_REVMEMB_REPLY* pmsgOut
);

```

hDrs: RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.8.1 Method-Specific Concrete Types

4.1.8.1.1 DRS_MSG_REVMEMB_REQ

The **DRS_MSG_REVMEMB_REQ** union defines the request messages sent to the [IDL_DRSGetMemberships](#) method. Only one version, identified by `dwInVersion = 1`, is currently defined.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_REVMEMB_REQ_V1 V1;
} DRS_MSG_REVMEMB_REQ;

```

V1: Version 1 request. Currently only one version is defined.

4.1.8.1.2 DRS_MSG_REVMEMB_REQ_V1

The **DRS_MSG_REVMEMB_REQ_V1** structure defines the request message sent to the [IDL_DRSGetMemberships](#) method.

```

typedef struct {
    [range(1,10000)] ULONG cDsNames;
    [size_is(cDsNames)] DSNAME** ppDsNames;
    DWORD dwFlags;
    [range(1,7)] REVERSE_MEMBERSHIP_OPERATION_TYPE OperationType;
    DSNAME* pLimitingDomain;
} DRS_MSG_REVMEMB_REQ_V1;

```

cDsNames: The count of items in the ppDsNames array.

ppDsNames: The [DSName](#) type of the object whose reverse membership is being requested, plus the [DSName](#) types of groups of the appropriate type(s) of which it is already known to be a member.

dwFlags: Zero or more of the following bit flags:

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	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

X: Unused. MUST be zero and ignored.

A (DRS_REVMEMB_FLAG_GET_ATTRIBUTES): Query the attributes that correspond to the group membership.

OperationType: The type of group membership evaluation to be performed.

pLimitingDomain: Domain filter; resulting objects that are not from this domain are neither returned nor followed transitively.

4.1.8.1.3 REVERSE_MEMBERSHIP_OPERATION_TYPE

The **REVERSE_MEMBERSHIP_OPERATION_TYPE** enumeration defines the type of reverse membership evaluation.

```
typedef enum
{
    RevMembGetGroupsForUser = 1,
    RevMembGetAliasMembership,
    RevMembGetAccountGroups,
    RevMembGetResourceGroups,
    RevMembGetUniversalGroups,
    GroupMembersTransitive,
    RevMembGlobalGroupsNonTransitive
} REVERSE_MEMBERSHIP_OPERATION_TYPE;
```

RevMembGetGroupsForUser: Nontransitive membership in nondomain local groups that are confined to a given domain, excluding built-in groups.

RevMembGetAliasMembership: Nontransitive membership in domain local groups that are confined to a given domain.

RevMembGetAccountGroups: Transitive membership in all account groups in a given domain, excluding built-in groups.

RevMembGetResourceGroups: Transitive membership in all domain local groups in a given domain, excluding built-in groups.

RevMembGetUniversalGroups: Transitive membership in all universal groups, excluding built-in groups.

GroupMembersTransitive: Transitive closure of members of a group based on the information present in the server's NC replicas, including the primary group.

RevMembGlobalGroupsNonTransitive: Nontransitive membership in global groups, excluding built-in groups.

4.1.8.1.4 DRS_MSG_REVMEMB_REPLY

The **DRS_MSG_REVMEMB_REPLY** union defines the response messages received from the [IDL DRSGetMemberships](#) method. Only one version, identified by `pdwOutVersion^ = 1`, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_REVMEMB_REPLY_V1 V1;
} DRS_MSG_REVMEMB_REPLY;
```

V1: Version 1 reply.

4.1.8.1.5 DRS_MSG_REVMEMB_REPLY_V1

The **DRS_MSG_REVMEMB_REPLY_V1** structure defines the response message received from the [IDL DRSGetMemberships](#) method.

```
typedef struct {
    ULONG errCode;
    [range(0,10000)] ULONG cDsNames;
    [range(0,10000)] ULONG cSidHistory;
    [size_is(cDsNames)] DSNAME** ppDsNames;
    [size_is(cDsNames)] DWORD* pAttributes;
    [size_is(cSidHistory)] NT4SID** ppSidHistory;
} DRS_MSG_REVMEMB_REPLY_V1;
```

errCode: 0 on success. On failure, this can be one of the following:

Value	Meaning
STATUS_INSUFFICIENT_RESOURCES 0xC000009A	Insufficient system resources exist to complete the request.
STATUS_TOO_MANY_CONTEXT_IDS 0xC000015A	The number of groups is greater than the number that can be returned to the caller.

cDsNames: Count of items in the `ppDsNames` array.

cSidHistory: Count of items in the `ppSidHistory` array.

ppDsNames: The filtered group membership. This list contains the combined membership for all the names specified in `ppDsNames` field of the input [DRS_MSG_REVMEMB_REQ_V1](#) structure.

pAttributes: Properties of the returned groups. Values are chosen from [SE_GROUP values](#).

ppSidHistory: SID history of the returned groups.

4.1.8.1.6 SE_GROUP Values

Attributes of a security group.

Symbolic name	Value
SE_GROUP_MANDATORY	0x00000001
SE_GROUP_ENABLED_BY_DEFAULT	0x00000002
SE_GROUP_ENABLED	0x00000004

4.1.8.2 Method-Specific Abstract Types and Procedures

4.1.8.2.1 Arc and ArcSet

```
type Arc = [initial: DSName, final: DSName]
type ArcSet = set of Arc
```

4.1.8.2.2 Closure

```
procedure Closure(
  vSet: set of DSName,
  aSet: ArcSet,
  v: DSName): set of DSName
```

The Closure procedure returns the set of vertices that can be reached from vertex *v* in the directed graph that consists of vertex set *vSet* and arc set *aSet*. A vertex *u* can be reached from *v* if and only if there is a sequence *v*[0], *v*[1], ... , *v*[*k*], where *v*[0]=*v*, *v*[*k*]=*u*, and *v*[*i*] is in *vSet* and [initial:*v*[*i*-1], final:*v*[*i*]] is in *aSet* for *i*=1,2,...,*k*.

4.1.8.2.3 DomainOf

```
procedure DomainOf(o: DSName): DSName
```

The DomainOf procedure returns the [DSName](#) of the domain NC to which the given [DSName](#) *o* belongs. It returns null upon failure.

4.1.8.2.4 GetDSNameFromPrimaryGroupId

```
procedure GetDSNameFromPrimaryGroupId(rid: Rid): DSName
```

This procedure constructs a SID *s* consisting of the domain SID of the DC's default domain and the given relative identifier **rid**, and returns the [DSName](#) of the object *o* for which *o*!objectSid = *s*. If no such object *o* exists, then this procedure will return null.

4.1.8.2.5 IsMatchedGroup

```
procedure IsMatchedGroup(  
    w: DSName,  
    op: REVERSE_MEMBERSHIP_OPERATION_TYPE,  
    limitingDomain: DSName): boolean
```

Informative summary of behavior: The IsMatchedGroup procedure checks whether an object should be included in the result for the specified [IDL DRSGetMemberships](#) operation.

```
    limitToDomain, filteroutBuiltin, result: boolean  
w: DSName  
limitToDomain := (op ≠ RevMembGetUniversalGroups) and  
    (limitingDomain ≠ null)  
filteroutBuiltin := (op ≠ RevMembGetAliasMembership)  
result := (GROUP_SECURITY_ENABLED in w!groupType)  
    and ((not limitToDomain) or (limitingDomain = DomainOf(w)))  
    and ((not filteroutBuiltin) or (not IsBuiltinPrincipal(w.sid)))  
    and ((op ≠ RevMembGetGroupsForUser)  
        or (w!groupType ∩ {GROUP_TYPE_RESOURCE_GROUP,  
            GROUP_TYPE_APP_BASIC_GROUP,  
            GROUP_TYPE_APP_QUERY_GROUP} = {}))  
    and ((op ≠ RevMembGetAliasMembership)  
        or (w!groupType ∩ {GROUP_TYPE_RESOURCE_GROUP,  
            GROUP_TYPE_APP_BASIC_GROUP,  
            GROUP_TYPE_APP_QUERY_GROUP} ≠ {}))  
    and ((op ≠ RevMembGetAccountGroups)  
        or (GROUP_TYPE_ACCOUNT_GROUP in w!groupType))  
    and ((op ≠ RevMembGetResourceGroups)  
        or (GROUP_TYPE_RESOURCE_GROUP in w!groupType))  
    and ((op ≠ RevMembGetUniversalGroups)  
        or (GROUP_TYPE_UNIVERSAL_GROUP in w!groupType))  
    and ((op ≠ RevMembGlobalGroupsNonTransitive)  
        or (GROUP_TYPE_ACCOUNT_GROUP in w!groupType))  
return result
```

4.1.8.2.6 Neighbors

```
procedure Neighbors(  
    vSet: set of DSName,  
    aSet: ArcSet,  
    v: DSName): set of DSName
```

The Neighbors procedure returns the set of vertices adjacent to vertex v in the directed graph that consists of vertex set vSet and arc set aSet.

A vertex u is adjacent to v if u is in vSet and [initial:v, final:u] is in aSet. Note that because this is a directed graph, the fact that vertex u is adjacent to vertex v does not imply that vertex v is adjacent to vertex u.

4.1.8.3 Server Behavior of the IDL_DRSGetMemberships Method

Informative summary of behavior: The [IDL_DRSGetMemberships](#) method constructs a directed graph $G(V,A)$. The vertex set of the graph includes all the objects in the scope of the forest if the server is a GC, or in the scope of the default domain NC otherwise. The arc set of the graph includes all the tuples [initial: u,final: v] if u is a member of v and both u and v are in the scope. This graph represents the membership relation in the given scope.

For a GroupMembersTransitive request, a reversed graph of G is used because member relation is queried rather than membership. The reversed graph has the same vertex set as G, but the arcs in the arc set are in opposite direction as those in A.

For other types of requests, a subgraph of G is used. The vertex set of this subgraph consists of only the [DSName](#) values of interest for that particular request type, and the arc set is reduced to the arcs that link two vertices in the vertex set of the subgraph.

Starting from the graph, this method computes a set of objects for each [DSName](#) in the input parameters. The set could be either transitive closure of the object or the immediate neighbors of the object in the graph, depending on the type of request. The union of these sets is returned as the result.

```
ULONG
IDL_DRSGetMemberships(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_REVMEMB_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_REVMEMB_REPLY *pmsgOut)

msgIn: DRS_MSG_REVMEMB_REQ V1
vSet, wSet, uSet: set of DSName
aSet, aSetR: ArcSet
u,v,w: DSName
op, i: integer
transitive: boolean
t: SID

msgIn := pmsgIn^.V1

if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
if not AccessCheckCAR(DefaultNC(), DS-Replication-Get-Changes) then
    return ERROR_DS_DRA_ACCESS_DENIED
endif

op := msgIn.OperationType
if (op = RevMembGetUniversalGroups) and not IsGC() then
    return ERROR_DS_GC_REQUIRED
endif

pdwOutVersion^ := 1
pmsgOut^.V1.cDsNames := 0
pmsgOut^.V1.cSidHistory := 0

/* Construct a membership graph. */
```

```

/* Vertices */
if IsGC() then
    vSet := select all v from all where true
else
    vSet := select all v from subtree DefaultNC() where true
Endif

/* Edges */
aSet := {}
aSetR := {}
foreach v in vSet
    foreach u in vSet
        if (u in v!memberOf)
            or (u = GetDSNameFromPrimaryGroupId(v!primaryGroupId)) then
                aSet := aSet + {[initial: v, final: u]}
                aSetR := aSetR + {[initial: u, final: v]}
            endif
        endfor
    endfor

/* Calculate GroupMembersTransitive. */
if op = GroupMembersTransitive then
    wSet := {}
    for i := 0 to msgIn.ppDsNames.cDsNames - 1
        u := msgIn.ppDsNames[i]
        if u in vSet then
            wSet := wSet + (Closure(uSet, aSetR, u) - {u})
        endif
    endfor

    foreach w in wSet
        pmsgOut^.V1.ppDsNames[pmsgOut^.V1.cDsNames] := w
        pmsgOut^.V1.cDsNames := pmsgOut^.V1.cDsNames + 1
    endfor

    return 0
endif

/* Calculate all other cases (where op ≠ GroupMembersInTransitive).*/
transitive := op in {RevMembGetAccountGroups,
                    RevMembGetResourceGroups,
                    RevMembGetUniversalGroups}

/* Get the initial result set from the graph. */
wSet := {}
for i := 0 to msgIn.ppDsNames.cDsNames - 1
    u := msgIn.ppDsNames[i]
    if u in vSet then
        /* Get the subgraph by applying the predicate IsMatchedGroup
        * on each element in the vertex set, plus u itself. */
        uSet := {u} + select all v from vSet where
            IsMatchedGroup(v, op, msgIn.pLimitingDomain^)
        if transitive then
            wSet := wSet + (Closure(uSet, aSet, u) - {u})
        else
            wSet := wSet + (Neighbors(uSet, aSet, u) - {u})
        endif
    endif
endfor

```

```

endfor

/* Construct the result message. */
pmsgOut^.V1.cSidHistory := 0
pmsgOut^.V1.cDsNames := 0

foreach w in wSet
  foreach t in w!sIDHistory
    if not (t in pmsgOut^.V1.ppSidHistory) then
      pmsgOut^.V1.ppSidHistory[pmsgOut^.V1.cSidHistory] := t
      pmsgOut^.V1.cSidHistory := pmsgOut^.V1.cSidHistory + 1
    endif
  endfor

  pmsgOut^.V1.ppDsNames[pmsgOut^.V1.cDsNames] := w

  if (DRS_REVMEMB_FLAG_GET_ATTRIBUTES in msgIn.dwFlags) then
    pmsgOut^.V1.pAttributes[pmsgOut^.V1.cDsNames] :=
      {SE_GROUP_MANDATORY, SE_GROUP_ENABLED_BY_DEFAULT,
       SE_GROUP_ENABLED}
  else
    pmsgOut^.V1.pAttributes[pmsgOut^.V1.cDsNames] := 0
  endif
  pmsgOut^.V1.cDsNames := pmsgOut^.V1.cDsNames + 1
endfor

return 0

```

4.1.9 IDL_DRSGetMemberships2 (Opnum 21)

The **IDL_DRSGetMemberships2** method retrieves group memberships for a sequence of objects.

```

ULONG IDL_DRSGetMemberships2(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)]
    DRS_MSG_GETMEMBERSHIPS2_REQ* pmsgIn,
  [out, ref] DWORD* pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_GETMEMBERSHIPS2_REPLY* pmsgOut
);

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

Return Values: 0 if successful; otherwise, a Windows error code.

4.1.9.1 Method-Specific Concrete Types

4.1.9.1.1 DRS_MSG_GETMEMBERSHIPS2_REQ

The **DRS_MSG_GETMEMBERSHIPS2_REQ** union defines request messages sent to the [IDL DRSGetMemberships2](#) method. Only one version, -+identified by dwInVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_GETMEMBERSHIPS2_REQ_V1 V1;
} DRS_MSG_GETMEMBERSHIPS2_REQ;
```

V1: Version 1 request.

4.1.9.1.2 DRS_MSG_GETMEMBERSHIPS2_REQ_V1

The **DRS_MSG_GETMEMBERSHIPS2_REQ_V1** structure defines the request message sent to the [IDL DRSGetMemberships2](#) method.

```
typedef struct {
    [range(1,10000)] ULONG Count;
    [size_is(Count)] DRS_MSG_REVMEMB_REQ_V1* Requests;
} DRS_MSG_GETMEMBERSHIPS2_REQ_V1;
```

Count: Count of items in the Requests array.

Requests: Sequence of reverse membership requests.

4.1.9.1.3 DRS_MSG_GETMEMBERSHIPS2_REPLY

The **DRS_MSG_GETMEMBERSHIPS2_REPLY** union defines response messages received from the [IDL DRSGetMemberships2](#) method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_GETMEMBERSHIPS2_REPLY_V1 V1;
} DRS_MSG_GETMEMBERSHIPS2_REPLY;
```

V1: Version 1 response.

4.1.9.1.4 DRS_MSG_GETMEMBERSHIPS2_REPLY_V1

The **DRS_MSG_GETMEMBERSHIPS2_REPLY_V1** structure defines the response message received from the [IDL_DRSGetMemberships2](#) method.

```
typedef struct {
    [range(0,10000)] ULONG Count;
    [size_is(Count)] DRS_MSG_REVMEMB_REPLY_V1* Replies;
} DRS_MSG_GETMEMBERSHIPS2_REPLY_V1;
```

Count: Count of items in the Replies array.

Replies: Sequence of reverse membership replies, in the same order as the Requests field of the request message.

4.1.9.2 Server Behavior of the IDL_DRSGetMemberships2 Method

Informative summary of behavior: The [IDL_DRSGetMemberships2](#) method is merely a way to execute a series of IDL_DRSGetMemberships RPC calls via a single RPC request.

```
ULONG
IDL_DRSGetMemberships2(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_GETMEMBERSHIPS2_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_GETMEMBERSHIPS2_REPLY *pmsgOut)
error, i: ULONG
dummyVersion: DWORD
if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
pdwOutVersion^ := 1
pmsgOut^.V1.Count := pmsgIn^.V1.Count
for i := 0 to pmsgIn^.V1.Count - 1
    /* Call IDL_DRSGetMemberships as a local procedure. */
    error := IDL_DRSGetMemberships(null, 1, ADR(pmsgIn^.V1.Request[i]),
        ADR(dummyVersion), ADR(pmsgOut^.V1.Replies[i]))
    if error ≠ 0 then
        return error
    endif
endfor
return 0
```

4.1.10 IDL_DRSGetNCChanges (Opnum 3)

The **IDL_DRSGetNCChanges** method replicates updates from an NC replica on the server.

```
ULONG IDL_DRSGetNCChanges(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
```

```

[in, ref, switch_is(dwInVersion)]
    DRS_MSG_GETCHGREQ* pmsgIn,
[out, ref] DWORD* pdwOutVersion,
[out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_GETCHGREPLY* pmsgOut
);

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: Version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.10.1 Overview

A client DC sends an [IDL_DRSGetNCChanges](#) request (msgIn, of a type in the union [DRS_MSG_GETCHGREQ](#)) to a server to replicate directory objects in a given NC from the server NC replica to the client NC replica.

The response (msgOut, of a type in the union [DRS_MSG_GETCHGREPLY](#)) contains a set of updates that the client is to apply to its NC replica. Commonly, this set of updates is too large to send in a single response; in this case, multiple [IDL_DRSGetNCChanges](#) requests and responses must be sent before the server sends a response that indicates no additional updates are available.

This sequence of requests and responses is called a replication cycle, or "cycle". A client DC can request an action on a FSMO role (for example, a change in the FSMO role owner) by using a special replication cycle called an extended operation.

4.1.10.1.1 Cycle Start and Finish

There are five types of cycle starts:

1. The client explicitly signals the start of a special single-response cycle when it requests an extended operation. Such cycles always consist of a single response which sets msgOut.fMoreData = false.
2. The client explicitly signals the start of a cycle by sending msgIn.uuidInvocIdSrc = 0 or msgIn.usnvecFrom = 0.
3. The client sends values of msgIn.uuidInvocIdSrc and msgIn.usnvecFrom that were returned by the server as msgOut.uuidInvocIdSrc and msgOut.usnvecTo in the final message of some other cycle.
4. The server detects either that msgIn.uuidInvocIdSrc ≠ [dc](#).invocationId or that msgIn.uuidInvocIdSrc and msgIn.usnvecFrom were not returned by the server in the final message of some other cycle.
5. The server implementation MAY [≤11>](#) declare the supplied values of msgIn.uuidInvocIdSrc and msgIn.usnvecFrom as too stale to use.

If the server starts a new cycle based on items 4 or 5, the server ignores msgIn.usnvecFrom, treating it as though it were zero.

The fields msgOut.usnvecTo and msgIn.usnvecFrom have the same type, [USN_VECTOR](#). The internal format of **USN_VECTOR** is determined entirely by the server implementation and is subject only to the requirement that msgIn.usnvecFrom = 0 represents the start of a cycle. The server MAY [<12>](#) use **USN_VECTOR** to encode the start of a cycle.

Any server response message with msgOut.fMoreData = false is the final response in a cycle.

4.1.10.1.2 Cycle Goal

For any cycle that is not an extended operation, the goal of the server is to send updates such that at the conclusion of the cycle, the client NC replica contains all updates that were present in the server NC replica at the start of the cycle. More concretely, if cycleStartUtd is the server's msgIn.pNC^![replUpToDateVector](#) on receipt of the first request in a cycle where msgIn.ulExtendedOp = 0, then the final response in the cycle MUST contain msgOut.pUpToDateVecSrc such that HasUpdateKnowledge(msgOut.pUpToDateVecSrc^, cycleStartUtd) = true:

```
procedure HasUpdateKnowledge(  
    utd1: UPTODATE_VECTOR_V2_EXT,  
    utd2: UPTODATE_VECTOR_V2_EXT): boolean  
begin  
    i: integer  
    j: integer  
  
    /* Return true if and only if utd1 asserts the presence of all  
     * updates asserted by utd2. */  
    for i := 0 to utd2.cNumCursors - 1  
        j := select one k from [0 .. utd1.cNumCursors - 1] where  
            utd1.rgCursors[k].uuidDsa = utd2.rgCursors[i].uuidDsa  
            if j = null or utd1.rgCursors[j].usnHighPropUpdate <  
                utd2.rgCursors[i].usnHighPropUpdate then  
                return false  
            endif  
        endfor  
  
    return true  
end HasUpdateKnowledge
```

The server MAY [<13>](#) advance the cycle goal on each request such that it includes updates that the server has applied since the first request in the cycle.

The cycle goal includes a cursor c for the server DC such that:

- c.uuidDsa is the value of the [invocationId](#) attribute of the server's [nTDSDSA](#) object.
- c.usnHighPropUpdate is the highest USN such that the server can assert that, including the updates in this response, the client has applied any update with stamp s where s.uuidOriginating = c.uuidDsa and s.usnOriginating ≤ c.usnHighPropUpdate. If the server has originated no updates in the NC, it MAY [<14>](#) set c.usnHighPropUpdate to 0.
- c.timeLastSyncSuccess is the time at which the server sends the final response.

4.1.10.1.3 Extended Operations

The extended operation specified by `msgIn.ulExtendedOp` is one of the following:

- Request Role (`EXOP_FSMO_REQ_ROLE`, `EXOP_FSMO_REQ_PDC`, `EXOP_FSMO_RID_REQ_ROLE`): Changes the FSMO role owner from the server to the client DC, and then adds all changed objects and link values in the FSMO role to the response, including but not limited to the FSMO role owner change.
- Abandon Role (`EXOP_FSMO_ABANDON_ROLE`): Performs a chained request to the current FSMO role owner to make the server DC the FSMO role owner. This request is sent to help avoid entering a state in which no DC considers itself the owner of the role.
- Allocate RIDs (`EXOP_FSMO_REQ_RID_ALLOC`): Allocates a new block of RIDs to the client DC.
- Replicate Single Object (`EXOP_FSMO_REPL_OBJ`): Adds any changes to the specified object to the response.

4.1.10.2 Method-Specific Concrete Types

4.1.10.2.1 DRS_MSG_GETCHGREQ

The **DRS_MSG_GETCHGREQ** union defines request messages that are sent to the [IDL DRSGetNCChanges](#) method. There are no V1, V2, V3, or V6 messages.

```
typedef
[switch_type(DWORD)]
union {
    [case(4)]
        DRS_MSG_GETCHGREQ_V4 V4;
    [case(5)]
        DRS_MSG_GETCHGREQ_V5 V5;
    [case(7)]
        DRS_MSG_GETCHGREQ_V7 V7;
    [case(8)]
        DRS_MSG_GETCHGREQ_V8 V8;
} DRS_MSG_GETCHGREQ;
```

V4: Version 4 request (Windows 2000 SMTP replication [\[MS-SRPL\]](#)).

V5: Version 5 request (Windows 2000 RPC replication).

V7: Version 7 request (Windows Server 2003 SMTP replication [\[MS-SRPL\]](#)).

V8: Version 8 request (Windows Server 2003 RPC replication).

4.1.10.2.2 DRS_MSG_GETCHGREQ_V3

The **DRS_MSG_GETCHGREQ_V3** structure defines a portion of the request message that is sent to the [IDL DRSGetNCChanges](#) method as part of SMTP replication ([\[MS-SRPL\]](#)). This is not a complete request message; it is embedded in [DRS MSG GETCHGREQ V4](#) and [DRS MSG GETCHGREQ V7.<15>](#)

```
typedef struct {
```

```

UUID uuidDsaObjDest;
UUID uuidInvocIdSrc;
[ref] DSNAME* pNC;
USN_VECTOR usnvecFrom;
[unique] UPTODATE_VECTOR_V1_EXT* pUpToDateVecDestV1;
[unique] PARTIAL_ATTR_VECTOR_V1_EXT* pPartialAttrVecDestV1;
SCHEMA_PREFIX_TABLE PrefixTableDest;
ULONG ulFlags;
ULONG cMaxObjects;
ULONG cMaxBytes;
ULONG ulExtendedOp;
} DRS_MSG_GETCHGREQ_V3;

```

uuidDsaObjDest: DSA GUID of the client DC.

uuidInvocIdSrc: Invocation ID of the server DC.

pNC: NC root of the replica to replicate or the FSMO role object for an extended operation.

usnvecFrom: Data that is used to correlate calls to **IDL_DRSGetNCChanges**.

pUpToDateVecDestV1: Stamp filter describing updates that the client has already applied.

pPartialAttrVecDestV1: A set of one or more attributes whose values are to be replicated to the client's partial replica.

PrefixTableDest: Prefix table with which to convert the [ATTRTYP](#) values in pPartialAttrVecDestV1 to OIDs.

ulFlags: A [DRS_OPTIONS](#) bit field.

cMaxObjects: An approximate cap on the number of objects to include in the reply.

cMaxBytes: An approximate cap on the number of bytes to include in the reply.

ulExtendedOp: 0 or an [EXOP_REQ](#) code.

4.1.10.2.3 DRS_MSG_GETCHGREQ_V4

The **DRS_MSG_GETCHGREQ_V4** structure defines the request message sent to the [IDL_DRSGetNCChanges](#) method. This message version is supported by Windows 2000 and later releases of Windows and is a superset of [DRS_MSG_GETCHGREQ_V3.<16>](#)

```

typedef struct {
    UUID uuidTransportObj;
    [ref] MTX_ADDR* pmtxReturnAddress;
    DRS_MSG_GETCHGREQ_V3 V3;
} DRS_MSG_GETCHGREQ_V4;

```

uuidTransportObj: An [objectGUID](#) of the [interSiteTransport](#) object that identifies the transport by which to send the reply.

pmtxReturnAddress: The transport-specific address to which to send the reply.

V3: Version 3 request.

4.1.10.2.4 DRS_MSG_GETCHGREQ_V5

The **DRS_MSG_GETCHGREQ_V5** structure defines the request message sent to the [IDL_DRSGetNCChanges](#) method. This message version is supported by Windows 2000 and later releases of Windows.

```
typedef struct {
    UUID uuidDsaObjDest;
    UUID uuidInvocIdSrc;
    [ref] DSNAME* pNC;
    USN_VECTOR usnvecFrom;
    [unique] UPTODATE_VECTOR_V1_EXT* pUpToDateVecDestV1;
    ULONG ulFlags;
    ULONG cMaxObjects;
    ULONG cMaxBytes;
    ULONG ulExtendedOp;
    ULARGE_INTEGER liFsmoInfo;
} DRS_MSG_GETCHGREQ_V5;
```

uuidDsaObjDest: DSA GUID of the client DC.

uuidInvocIdSrc: Invocation ID of the server DC.

pNC: NC root of the replica to replicate or of the FSMO role object for an extended operation.

usnvecFrom: Data used to correlate calls to **IDL_DRSGetNCChanges**.

pUpToDateVecDestV1: Stamp filter that describes updates the client has already applied.

ulFlags: [DRS_OPTIONS](#) bit field.

cMaxObjects: Approximate cap on the number of objects to include in the reply.

cMaxBytes: Approximate cap on the number of bytes to include in the reply.

ulExtendedOp: 0 or an extended operation request code.

liFsmoInfo: 0 or a value specific to the requested extended operation.

4.1.10.2.5 DRS_MSG_GETCHGREQ_V7

The **DRS_MSG_GETCHGREQ_V7** structure defines the request message sent to the [IDL_DRSGetNCChanges](#) method. This message version is supported by Windows Server 2003 and later releases of Windows and is a superset of [DRS_MSG_GETCHGREQ_V4](#).<17>

```
typedef struct {
    UUID uuidTransportObj;
    [ref] MTX_ADDR* pmtxReturnAddress;
    DRS_MSG_GETCHGREQ_V3 V3;
    [unique] PARTIAL_ATTR_VECTOR_V1_EXT* pPartialAttrSet;
    [unique] PARTIAL_ATTR_VECTOR_V1_EXT* pPartialAttrSetEx;
    SCHEMA_PREFIX_TABLE PrefixTableDest;
} DRS_MSG_GETCHGREQ_V7;
```

uuidTransportObj: An [objectGUID](#) of the [interSiteTransport](#) object the identifies the transport by which to send the reply.

pmtxReturnAddress: Transport-specific address to which to send the reply.

V3: Version 3 request.

pPartialAttrSet: A set of one or more attributes whose values are to be replicated to the client's partial replica, or null if the client has a full replica.

pPartialAttrSetEx: A set of one or more attributes whose values are to be added to the client's existing partial replica, or null.

PrefixTableDest: Prefix table with which to convert the [ATTRTYP](#) values in pPartialAttrSet and pPartialAttrSetEx to OIDs.

4.1.10.2.6 DRS_MSG_GETCHGREQ_V8

The **DRS_MSG_GETCHGREQ_V8** structure defines the request message sent to the [IDL_DRSGetNCChanges](#) method. This message version is supported by Windows Server 2003 and later releases of Windows and is a superset of [DRS_MSG_GETCHGREQ_V5](#).

```
typedef struct {
    UUID uuidDsaObjDest;
    UUID uuidInvocIdSrc;
    [ref] DSNAME* pNC;
    USN_VECTOR usnvecFrom;
    [unique] UPTODATE_VECTOR_V1_EXT* pUpToDateVecDest;
    ULONG ulFlags;
    ULONG cMaxObjects;
    ULONG cMaxBytes;
    ULONG ulExtendedOp;
    ULARGE_INTEGER liFsmoInfo;
    [unique] PARTIAL_ATTR_VECTOR_V1_EXT* pPartialAttrSet;
    [unique] PARTIAL_ATTR_VECTOR_V1_EXT* pPartialAttrSetEx;
    SCHEMA_PREFIX_TABLE PrefixTableDest;
} DRS_MSG_GETCHGREQ_V8;
```

uuidDsaObjDest: DSA GUID of the client DC.

uuidInvocIdSrc: Invocation ID of the server DC.

pNC: NC root of the replica to replicate or the FSMO role object for an extended operation.

usnvecFrom: Data used to correlate calls to **IDL_DRSGetNCChanges**.

pUpToDateVecDest: Stamp filter describing updates the client has already applied.

ulFlags: A [DRS_OPTIONS](#) bit field.

cMaxObjects: Approximate cap on the number of objects to include in the reply.

cMaxBytes: Approximate cap on the number of bytes to include in the reply.

ulExtendedOp: 0 or an extended operation request code.

liFsmoInfo: 0 or a value specific to the requested extended operation.

pPartialAttrSet: A set of one or more attributes whose values are to be replicated to the client's partial replica, or null if the client has a full replica.

pPartialAttrSetEx: A set of one or more attributes whose values are to be added to the client's existing partial replica, or null.

PrefixTableDest: Prefix table with which to convert the [ATTRTYP](#) values in pPartialAttrSet and pPartialAttrSetEx to OIDs.

4.1.10.2.7 DRS_MSG_GETCHGREPLY

The **DRS_MSG_GETCHGREPLY** union defines the response messages received from the [IDL DRSGetNCChanges](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_GETCHGREPLY_V1 V1;
    [case(2)]
        DRS_MSG_GETCHGREPLY_V2 V2;
    [case(6)]
        DRS_MSG_GETCHGREPLY_V6 V6;
    [case(7)]
        DRS_MSG_GETCHGREPLY_V7 V7;
} DRS_MSG_GETCHGREPLY;
```

V1: Version 1 response (Windows 2000).

V2: Version 2 response (compressed V1).

V6: Version 6 response (Windows Server 2003).

V7: Version 7 response (compressed V6).

4.1.10.2.8 DRS_MSG_GETCHGREPLY_V1

The **DRS_MSG_GETCHGREPLY_V1** structure defines the response message received from the [IDL DRSGetNCChanges](#) method. This message version is supported by Windows 2000 and later releases of Windows.

```
typedef struct {
    UUID uuidDsaObjSrc;
    UUID uuidInvocIdSrc;
    [ref] DSNAME* pNC;
    USN_VECTOR usnvecFrom;
    USN_VECTOR usnvecTo;
    [unique] UPTODATE_VECTOR_V1_EXT* pUpToDateVecSrcV1;
    SCHEMA_PREFIX_TABLE PrefixTableSrc;
```

```

    ULONG ulExtendedRet;
    ULONG cNumObjects;
    ULONG cNumBytes;
    [unique] REPLENTINFLIST* pObjects;
    BOOL fMoreData;
} DRS_MSG_GETCHGREPLY_V1;

```

uuidDsaObjSrc: DSA GUID of the server DC.

uuidInvocIdSrc: Invocation ID of the server DC.

pNC: NC root of the replica to replicate or of the FSMO role object for an extended operation.

usnvecFrom: Data used to correlate calls to **IDL_DRSGetNCChanges**.

usnvecTo: Data used to correlate calls to **IDL_DRSGetNCChanges**.

pUpToDateVecSrcV1: Stamp filter that describes updates the server has already applied.

PrefixTableSrc: Table for translating [ATTRTYP](#) values in the response to OIDs.

ulExtendedRet: 0 or an [EXOP_ERR](#) code.

cNumObjects: Count of items in the pObjects linked list.

cNumBytes: Size in bytes of items in or referenced by elements in the pObjects linked list.

pObjects: Linked list of object updates that the client can apply to its NC replica.

fMoreData: False if and only if this is the last response in a replication cycle.

4.1.10.2.9 DRS_MSG_GETCHGREPLY_V2

The **DRS_MSG_GETCHGREPLY_V2** structure defines the compressed [DRS MSG GETCHGREPLY_V1](#) message received from the [IDL_DRSGetNCChanges](#) method. This message version is supported by Windows 2000 and later releases of Windows.

```

typedef struct {
    DRS_COMPRESSED_BLOB CompressedV1;
} DRS_MSG_GETCHGREPLY_V2;

```

CompressedV1: Compressed **DRS_MSG_GETCHGREPLY_V1** response.

4.1.10.2.10 DRS_MSG_GETCHGREPLY_V6

The **DRS_MSG_GETCHGREPLY_V6** structure defines the response message received from the [IDL_DRSGetNCChanges](#) method. This message version is supported by Windows Server 2003 and later releases of Windows and is a superset of [DRS MSG GETCHGREPLY_V1](#).

```

typedef struct {
    UUID uuidDsaObjSrc;
    UUID uuidInvocIdSrc;

```

```

[ref] DSNAME* pNC;
USN_VECTOR usnvecFrom;
USN_VECTOR usnvecTo;
[unique] UPTODATE_VECTOR_V2_EXT* pUpToDateVecSrc;
SCHEMA_PREFIX_TABLE PrefixTableSrc;
ULONG ulExtendedRet;
ULONG cNumObjects;
ULONG cNumBytes;
[unique] REPLENTINFLIST* pObjects;
BOOL fMoreData;
ULONG cNumNcSizeObjects;
ULONG cNumNcSizeValues;
[range(0,1048576)] DWORD cNumValues;
[size_is(cNumValues)] REPLVALINF* rgValues;
DWORD dwDRSError;
} DRS_MSG_GETCHGREPLY_V6;

```

uuidDsaObjSrc: DSA GUID of the server DC.

uuidInvocIdSrc: Invocation ID of the server DC.

pNC: NC root of the replica to replicate or the FSMO role object for an extended operation.

usnvecFrom: Data used to correlate calls to **IDL_DRSGetNCChanges**.

usnvecTo: Data used to correlate calls to **IDL_DRSGetNCChanges**.

pUpToDateVecSrc: Stamp filter that describes updates the server has already applied.

PrefixTableSrc: Table for translating [ATTRTYP](#) values in the response to OIDs.

ulExtendedRet: 0 or an extended operation error code.

cNumObjects: Count of items in the pObjects linked list.

cNumBytes: Size in bytes of items in or referenced by elements in the pObjects linked list.

pObjects: Linked list of object updates for the client to apply to its NC replica.

fMoreData: False if and only if this is the last response in a replication cycle.

cNumNcSizeObjects: Estimated number of objects in the server's NC replica.

cNumNcSizeValues: Estimated number of link values in the server's NC replica.

cNumValues: Count of items in the rgValues array.

rgValues: Link value updates for the client to apply to its NC replica.

dwDRSError: 0 if successful, otherwise a Windows error code.

4.1.10.2.11 DRS_MSG_GETCHGREPLY_V7

The **DRS_MSG_GETCHGREPLY_V7** structure defines a compressed [DRS_MSG_GETCHGREPLY_V6](#) message received from the [IDL_DRSGetNCChanges](#) method. This message version is supported by Windows Server 2003 and later releases of Windows.

```
typedef struct {
    DWORD dwCompressedVersion;
    DRS_COMP_ALG_TYPE CompressionAlg;
    DRS_COMPRESSED_BLOB CompressedAny;
} DRS_MSG_GETCHGREPLY_V7;
```

dwCompressedVersion: Version of the response in CompressedAny; MUST be set to 6.

CompressionAlg: Algorithm used to compress the response.

CompressedAny: Compressed **DRS_MSG_GETCHGREPLY_V6** response.

4.1.10.2.12 COMPRESSED_DATA

The **COMPRESSED_DATA** structure defines a sequence of compressed (if cbDecompressedSize ≠ cbCompressedSize) or uncompressed (if cbDecompressedSize = cbCompressedSize) bytes.

```
typedef struct {
    ULONG cbDecompressedSize;
    ULONG cbCompressedSize;
    BYTE data[];
} COMPRESSED_DATA;
```

cbDecompressedSize: Decompressed size of data.

cbCompressedSize: Compressed size of data.

data: Data stream. The data is padded with zeros, if necessary, so that the block ends on a double word boundary.

4.1.10.2.13 DRS_COMP_ALG_TYPE

The **DRS_COMP_ALG_TYPE** enumeration is a concrete type for identifying a compression algorithm.

```
typedef enum
{
    DRS_COMP_ALG_NONE = 0,
    DRS_COMP_ALG_MSZIP = 2,
    DRS_COMP_ALG_WIN2K3 = 3
} DRS_COMP_ALG_TYPE;
```

DRS_COMP_ALG_NONE: No compression.

DRS_COMP_ALG_MSZIP: MSZIP algorithm.

DRS_COMP_ALG_WIN2K3: Windows Server 2003 compression

4.1.10.2.14 DRS_COMPRESSED_BLOB

The **DRS_COMPRESSED_BLOB** structure defines a concrete type that results from marshaling a data structure into a byte stream by using RPC and compressing that byte stream.

```
typedef struct {  
    DWORD cbUncompressedSize;  
    [range(1,10485760)] DWORD cbCompressedSize;  
    [size_is(cbCompressedSize)] BYTE* pbCompressedData;  
} DRS_COMPRESSED_BLOB;
```

cbUncompressedSize: Size in bytes of the uncompressed byte stream.

cbCompressedSize: Size in bytes of the pbCompressedData array.

pbCompressedData: Compressed byte stream.

Padding: Data is padded with zeros, if necessary, so that the block ends on an alignment boundary of [LONG](#).

4.1.10.2.15 ENCRYPTED_PAYLOAD

The ENCRYPTED_PAYLOAD packet is the concrete type for a value of an encrypted attribute.

```
typedef struct {  
    UCHAR Salt[16];  
    ULONG CheckSum;  
    UCHAR EncryptedData[];  
} ENCRYPTED_PAYLOAD;
```

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
Salt																															
...																															
...																															
...																															
Checksum																															
EncryptedData (variable)																															
...																															

Salt: A 128-bit randomly generated value.

Checksum: A 32-bit CRC32 checksum of the data that is encrypted along with the data.

EncryptedData: A variable-length byte array that represents the encrypted value.

4.1.10.2.16 EXOP_ERR Codes

The following values are error codes for an extended operation request to the [IDL_DRSGetNCChanges](#) method.

Symbolic name	Value
EXOP_ERR_SUCCESS	0x00000001
EXOP_ERR_UNKNOWN_OP	0x00000002
EXOP_ERR_FSMO_NOT_OWNER	0x00000003
EXOP_ERR_UPDATE_ERR	0x00000004
EXOP_ERR_EXCEPTION	0x00000005
EXOP_ERR_UNKNOWN_CALLER	0x00000006
EXOP_ERR_RID_ALLOC	0x00000007
EXOP_ERR_FSMO_OWNER_DELETED	0x00000008
EXOP_ERR_FSMO_PENDING_OP	0x00000009
EXOP_ERR_MISMATCH	0x0000000A
EXOP_ERR_COULDNT_CONTACT	0x0000000B

Symbolic name	Value
EXOP_ERR_FSMO_REFUSING_ROLES	0x0000000C
EXOP_ERR_DIR_ERROR	0x0000000D
EXOP_ERR_FSMO_MISSING_SETTINGS	0x0000000E
EXOP_ERR_ACCESS_DENIED	0x0000000F
EXOP_ERR_PARAM_ERROR	0x00000010

4.1.10.2.17 EXOP_REQ Codes

The following values are request codes for extended operation.

Symbolic name	Value
EXOP_FSMO_REQ_ROLE	0x00000001
EXOP_FSMO_REQ_RID_ALLOC	0x00000002
EXOP_FSMO_RID_REQ_ROLE	0x00000003
EXOP_FSMO_REQ_PDC	0x00000004
EXOP_FSMO_ABANDON_ROLE	0x00000005
EXOP_REPL_OBJ	0x00000006
EXOP_REPL_SECRETS	0x00000007

4.1.10.2.18 PROPERTY_META_DATA

The **PROPERTY_META_DATA** structure contains attribute and stamp information. For more information, see section [4.1.10.5.8](#).

The binary portion of the DNBinary value of the [msDS-RevealedUsers](#) attribute contains this structure.

```
typedef struct PROPERTY_META_DATA {
    ATTRTYP attrType;
    PROPERTY_META_DATA_EXT propMetadataExt;
} PROPERTY_META_DATA;
```

attrType: Attribute whose value was revealed.

propMetadataExt: Stamp of revealed attribute value.

4.1.10.3 Method-Specific Abstract Types and Procedures

4.1.10.3.1 AbstractLinkValStampFromConcreteLinkValStamp

```
procedure AbstractLinkValStampFromConcreteLinkValStamp(
```

```

concreteStamp: VALUE_META_DATA_EXT_V1,
timeDeleted: DSTIME) : LinkValueStamp

```

Informative summary of behavior: The `AbstractLinkValStampFromConcreteLinkValStamp` procedure converts a [LinkValueStamp](#) to the [VALUE_META_DATA_EXT_V1](#) concrete type.

```

linkValueStamp : LinkValueStamp

linkValueStamp := concreteStamp.MetaData
linkValueStamp.timeCreated := concreteStamp.timeCreated
linkValueStamp.timeDelete := timeDeleted

return linkValueStamp

```

4.1.10.3.2 AbstractPASFromConcretePAS

```

procedure AbstractPASFromConcretePAS(
    concretePAS: PARTIAL_ATTR_VECTOR_V1_EXT,
    prefixTable: PrefixTable): sequence of ATTRTYP

```

Informative summary of behavior: The `AbstractPASFromConcretePAS` procedure translates a concrete [PARTIAL_ATTR_VECTOR_V1_EXT](#) to a sequence of [ATTRTYP](#), using `prefixTable` to translate the concretePAS entries.

```

abstractPAS: sequence of ATTRTYP
i: DWORD
for i := 0 to (concretePAS.cAttrs - 1)
    abstractPAS[i] := LocalAttidFromRemoteAttid(
        prefixTable, concretePAS.rgPartialAttr[i])
endfor
return abstractPAS

```

4.1.10.3.3 AbstractUTDFromConcreteUTD

```

procedure AbstractUTDFromConcreteUTD(
    concreteUTD: UPTODATE_VECTOR_V2_EXT): sequence of ReplUpToDateVector

```

Informative summary of behavior: The `AbstractUTDFromConcreteUTD` procedure translates the [UPTODATE_VECTOR_V2_EXT](#) structure to the [ReplUpToDateVector](#) abstract type.

```

abstractUTD: ReplUpToDateVector

for i := 0 to (concreteUTD.length - 1)
    abstractUTD[i].uuidDsa := concreteUTD.rgCursors[i].uuidDsa
    abstractUTD[i].usnHighPropUpdate :=
        concreteUTD.rgCursors[i].usnHighPropUpdate
    abstractUTD[i].timeLastSyncSuccess :=
        concreteUTD.rgCursors[i].timeLastSyncSuccess
endfor

```



```
return concreteUTD
```

4.1.10.3.4 AttributeAndStamp

```
type AttributeAndStamp = [attribute: ATTRTYP, stamp: AttributeStamp]
```

This abstract type encapsulates the [ATTRTYP](#) of an attribute (based on [dc.prefixTable](#)) and its associated [AttributeStamp](#) on an object.

4.1.10.3.5 AttributeStampCompare

```
procedure AttributeStampCompare(  
    stamp1: AttributeStamp,  
    stamp2: AttributeStamp): integer
```

Informative summary of behavior: The AttributeStampCompare procedure compares two [AttributeStamp](#) values, stamp1 and stamp2. If stamp1 > stamp2 then the procedure returns an integer with a value greater than 0. If stamp1 = stamp2 then the procedure returns 0. If stamp1 < stamp2 then the procedure returns an integer value less than 0.

```
d: integer  
/* The value of d will be the result of stamp1.dwVersion -  
 * stamp2.dwVersion, cast as a 32-bit integer. For example, if  
 * stamp1.dwVersion is 1 and stamp2.dwVersion is 3, d is -2. If  
 * stamp1.dwVersion is 5 and stamp2.dwVersion is 0xFFFFFFFF, d is 11.  
 */  
d := stamp1.dwVersion - stamp2.dwVersion  
if d = 0 then  
    if stamp1.timeChanged > stamp2.timeChanged then  
        d := 1  
    else if stamp1.timeChanged < stamp2.timeChanged then  
        d := -1  
    endif  
endif  
if d = 0 then  
    if stamp1.uuidOriginating > stamp2.uuidOriginating then  
        d := 1  
    else if stamp1.uuidOriginating < stamp2.uuidOriginating then  
        d := -1  
    endif  
endif  
return d
```

4.1.10.3.6 ConcretePASFromAbstractPAS

```
procedure ConcretePASFromAbstractPAS(  
    abstractPAS: sequence of ATTRTYP) : PARTIAL_ATTR_VECTOR_V1_EXT
```

Informative summary of behavior: The ConcretePASFromAbstractPAS procedure translates a sequence of [ATTRTYP](#) to [PARTIAL_ATTR_VECTOR_V1_EXT](#). This translation does not require a prefix table.

```
concretePAS : PARTIAL_ATTR_VECTOR_V1_EXT
i: DWORD
concretePAS.dwVersion := 1
concretePAS.dwReserved1 := 0
concretePAS.cAttrs := abstractPAS.length
for i := 0 to (abstractPAS.length - 1)
    concretePAS.rgPartialAttr[i] := abstractPAS[i]
endfor
return concretePAS
```

4.1.10.3.7 ConcreteUTDFromAbstractUTD

```
procedure ConcreteUTDFromAbstractUTD(
    abstractUTD: sequence of ReplUpToDateVector):
    UPTODATE_VECTOR_V1_EXT
```

Informative summary of behavior: The ConcreteUTDFromAbstractUTD procedure translates a sequence of abstract [ReplUpToDateVector](#) tuples to [UPTODATE_VECTOR_V1_EXT](#).

```
concreteUTD: UPTODATE_VECTOR_V1_EXT
concreteUTD.dwVersion := 1
concreteUTD.dwReserved1 := 0
concreteUTD.dwReserved2 := 0
concreteUTD.cNumCursors := abstractUTD.length
for i := 0 to (abstractUTD.length - 1)
    concreteUTD.rgCursors[i].uuidDsa := abstractUTD[i].uuidDsa
    concreteUTD.rgCursors[i].usnHighPropUpdate :=
        abstractUTD[i].usnHighPropUpdate
endfor
return concreteUTD
```

4.1.10.3.8 GetNCChangesV6Reply

```
procedure GetNCChangesV6Reply(
    replyMessage: DRS_MSG_GETCHGREPLY,
    version: DWORD): DRS_MSG_GETCHGREPLY_V6
```

Informative summary of behavior: The GetNCChangesV6Reply procedure transforms a [DRS_MSG_GETCHGREPLY](#) union of version 1 or 6 to a [DRS_MSG_GETCHGREPLY_V6](#) structure.

```
msgReplyV6: DRS_MSG_GETCHGREPLY_V6
i: DWORD
if (version = 1) then
    msgReplyV6 := 0
    msgReplyV6 := replyMessage.V1

/* Convert UPTODATE_VECTOR_V1_WIRE structure to
 * UPTODATE_VECTOR_V2_WIRE structure. */
```

```

msgReplyV6.pUpToDateVecSrc^.dwVersion := 2
msgReplyV6.pUpToDateVecSrc^.cNumCursors :=
    replyMessage.V1.pUpToDateVecSrcV1^.cNumCursors
for i := 0 to (replyMessage.V1.pUpToDateVecSrcV1^.cNumCursors - 1)
    msgReplyV6.pUpToDateVecSrc^.rgCursors[i] :=
        replyMessage.V1.pUpToDateVecSrcV1^.rgCursors[i]
    msgReplyV6.pUpToDateVecSrc^.rgCursors[i].timeLastSyncSuccess := 0
endfor
return msgReplyV6
else
    return replyMessage.V6
endif

```

4.1.10.3.9 GetStampsForUpdate

```

procedure GetStampsForUpdate(
    replEntinfList: REPLENTINFLIST,
    prefixTable: PrefixTable): set of AttributeAndStamp

```

Informative summary of behavior: The GetStampsForUpdate procedure retrieves the [AttributeStamp](#) type associated with an attribute in the [REPLENTINFLIST](#) update and constructs a set of AttributeAndStamp tuples.

```

tupleEntry: AttributeAndStamp
attrStamps: set of AttributeStamp
i: DWORD

for i := 0 to (replEntinfList.pMetaDataExt.cNumProps - 1)
    tupleEntry.attribute := LocalAttidFromRemoteAttid(
        prefixTable, replEntinfList.Entinf.AttrBlock.pAttr[i].attrTyp)
    tupleEntry.stamp := AbstractAttrStampFromConcreteAttrStamp(
        replEntinfList.pMetaDataExt.rgMetaData[i])
    attrStamps := attrStamps + {tupleEntry}
endfor
return attrStamps

```

4.1.10.3.10 GetWellKnownObject

```

procedure GetWellKnownObject(
    nc: DSName,
    guid: GUID): DSName

```

Informative summary of behavior: The GetWellKnownObject procedure returns the [DSName](#) of the well-known object with the given guid in a specified NC replica.

```

attrVals: set of attribute value
attrVal: DNBinary

attrVals := {nc!wellKnownObjects}
for each attrVal in attrVals do
    if (attrVal.binary = guid) then
        return attrVal.dn

```

```

    endif
endfor

return null

```

4.1.10.3.11 IsProtectedObject

```

procedure IsProtectedObject(obj: DSName): boolean

```

The IsProtectedObject procedure returns true if obj is the [nTDSDSA](#) object of the DC or an ancestor of the [nTDSDSA](#) object of the DC. Otherwise, the procedure returns false.

4.1.10.3.12 IsSecretAttribute

```

procedure IsSecretAttribute(attribute : ATTRTYP): boolean

```

The IsSecretAttribute procedure returns true if attribute is an attribute that contains secret data. Otherwise, the procedure returns false.

```

return (attribute in
{currentValue, dbcsPwd, initialAuthIncoming, initialAuthOutGoing,
lmPwdHistory, ntPwdHistory, priorValue, supplementalCredentials,
trustAuthIncoming, trustAuthOutgoing, unicodePwd} )

```

4.1.10.3.13 IsUserIncluded

```

procedure IsUserIncluded(
    userSid: SID
    groupOrAccountSid: SID)

```

The IsUserIncluded procedure returns true if userSid = groupOrAccountSid, or if the object identified by userSid is a member of the set returned by [IDL DRSGetMemberships \(section 4.1.8\)](#) with the GroupMembersTransitive option applied to the object identified by groupOrAccountSid.

4.1.10.3.14 LinkValueStampCompare

```

procedure LinkValueStampCompare(
    LinkValueStamp stamp1,
    LinkValueStamp stamp2): integer

```

Informative summary of behavior: The LinkValueStampCompare procedure compares two [LinkValueStamp](#) types, stamp1 and stamp2. If stamp1 > stamp2 then the procedure returns an integer with a value greater than 0. If stamp1 = stamp2 then the procedure returns 0. If stamp1 < stamp2 then the procedure returns an integer value less than 0.

```

d: integer
d := 0
if stamp1.dwVersion ≠ 0 and stamp2.dwVersion = 0 then

```

```

    d := 1
else if stamp1.dwVersion = 0 and stamp2.dwVersion ≠ 0 then
    d := -1
endif
if d = 0 then
    if stamp1.timeCreated > stamp2.timeCreated then
        d := 1
    else if stamp1.timeCreated < stamp2.timeCreated then
        d := -1
    endif
endif
endif
if d = 0 then
    /* The value of d will be the result of stamp1.dwVersion -
    * stamp2.dwVersion, cast as a 32-bit integer. For example, if
    * stamp1.dwVersion is 1 and stamp2.dwVersion is 3, d is -2. If
    * stamp1.dwVersion is 5 and stamp2.dwVersion is 0xFFFFFFFF,
    * d is 11.
    */
    d := stamp1.dwVersion - stamp2.dwVersion
endif
if d = 0 then
    if stamp1.timeChanged > stamp2.timeChanged then
        d := 1
    else if stamp1.timeChanged < stamp2.timeChanged then
        d := -1
    endif
endif
endif
if d = 0 then
    if stamp1.uuidOriginating > stamp2.uuidOriginating then
        d := 1
    else if stamp1.uuidOriginating < stamp2.uuidOriginating then
        d := -1
    endif
endif
endif
return d

```

4.1.10.3.15 ObjAtts

```
type ObjAtts = [obj: DSName, atts: sequence of ATTRTYP]
```

The ObjAtts abstract type encapsulates the identity of an object (obj) and a sequence of [ATTRTYP](#) values (atts, based on [dc.prefixTable](#)) for attributes of that object.

4.1.10.3.16 ObjAttVal

```
type ObjAttVal = [obj: DSName; att: ATTRTYP, val: attribute value]
```

The ObjAttVal abstract type encapsulates the identity of an object (obj), the [ATTRTYP](#) of an attribute of that object (att, based on [dc.prefixTable](#)), and a value of that attribute (val).

4.1.10.3.17 PerformModifyDNOperation

```
procedure PerformModifyDNOperation(
```

```
currentDN: DN,  
newParentObject: DSName,  
newRDN: RDN)
```

The PerformModifyDNOperation procedure performs a Modify DN operation on an object with the DN currentDN by setting its new parent to newParentObject and by setting the RDN value to newRDN. See [\[MS-ADTS\]](#) section 3.1.1.5.4 for more information.

4.1.10.3.18 RemoveAttrVal

```
procedure RemoveAttrVal(  
  obj: DSName,  
  attr: ATTRTYP,  
  attributeValue: attribute value)
```

This procedure removes value attributeValue from attribute attr on object with [DSName](#) obj.

4.1.10.3.19 SetAttrStamp

```
procedure SetAttrStamp(  
  obj: DSName,  
  attr: ATTRTYP,  
  stamp: AttributeStamp)
```

The SetAttrStamp procedure sets [AttributeStamp](#) to stamp for the attr attribute on the obj object.

4.1.10.3.20 SetAttrVal

```
procedure SetAttrVal(  
  obj: DSName,  
  attr: ATTRTYP,  
  attributeValue: attribute value)
```

This procedure sets value attributeValue for attribute attr on object obj.

4.1.10.3.21 SetLinkStamp

```
procedure SetLinkStamp(  
  obj: DSName,  
  attr: ATTRTYP,  
  val: attribute value,  
  stamp: AttributeStamp)
```

The SetLinkStamp procedure sets [LinkValueStamp](#) to stamp for the attribute value val on the attribute attr on the object obj.

4.1.10.4 Client Behavior When Sending the IDL_DRSGetCCChanges Request

Informative summary of behavior: The following three tasks can be accomplished by sending an [IDL_DRSGetNCChanges](#) request to a server:

1. Replicate objects from the server's NC replica. The [ReplicateNCRequestMsg](#) procedure specifies the process of building **DRS_MSG_GETCHGREQ** to perform this task.
2. Replicate a single object from the server's NC replica. The [ReplSingleObjRequestMsg](#) procedure specifies the process of building **DRS_MSG_GETCHGREQ** to perform this task.
3. Perform extended operations. The [PerformExtendedOpRequestMsg](#) procedure specifies the process of building **DRS_MSG_GETCHGREQ** to perform this task.

After the DC constructs the request message, it sends the message by using the specified transport: SMTP (as specified in [\[MS-SRPL\]](#)) if `rf` \neq null and if `rf.uuidTransport` is the [objectGUID](#) of the [interSiteTransport](#) object `t`, where `t.cn` = "SMTP"; otherwise, the IP transport (RPC over TCP).

4.1.10.4.1 ReplicateNCRequestMsg

```
procedure ReplicateNCRequestMsg(
    hDrs: DRS_HANDLE,
    version: DWORD,
    nc: DSName,
    rf: RepsFrom,
    ulFlags: ULONG,
    cMaxObjects: ULONG,
    cMaxBytes: ULONG,
    var msgRequest: DRS_MSG_GETCHGREQ)
```

Informative summary of behavior: The client sends an [IDL_DRSGetNCChanges](#) request to a server to replicate the server's changes in an NC replica. The `ReplicateNCRequestMsg` procedure specifies how the client constructs the request message for this operation. The procedure has the following arguments:

- `hDrs`: The [DRS_HANDLE](#) that is derived by sending an [IDL_DRSBind](#) message to the server.
- `version`: The version number of the input message negotiated between the client and server.
- `nc`: The [DSName](#) of the root of the NC replica that is to be replicated.
- `rf`: The [RepsFrom](#) attribute that corresponds to the server from which to replicate.
- `ulFlags`: Zero or more of the following bit flags. The client MUST supply the same flags for each request in a given replication cycle, with the exception of `DRS_ADD_REF`, `DRS_GET_ANC`, `DRS_USE_COMPRESSION` and `DRS_GET_NC_SIZE`.
 - **DRS_ADD_REF**: Requests that the server add an entry to the [repsTo](#) attribute for the client on the root object of the NC replica that is being replicated.
 - **DRS_WRIT_REP**: Indicates that the client contains (or is constructing) a full, writable NC replica.
 - **DRS_ASYNC_REP**: Requests that the server send only the root object of the NC replica.
 - **DRS_CRITICAL_ONLY**: Signals the server not to send objects `o` where `o.isCriticalSystemObject` is absent or `o.isCriticalSystemObject` is false.
 - **DRS_GET_ANC**: Signals the server to send all updates for each ancestor object of object `o` before sending updates for object `o`.

- **DRS_GET_NC_SIZE**: Signals the server to set cNumNcSizeObjects in pmsgOut to an estimate of the number of objects in its NC replica.
- **DRS_FULL_SYNC_PACKET**: Requests that the server send all attributes of the objects in its reply, rather than sending only the updated attributes.
- **DRS_SYNC_FORCED**: Signals the server to honor the request even if its replication has otherwise been disabled.
- **DRS_USE_COMPRESSION**: Requests that the server reply by using one of the compressed reply versions ([DRS_MSG_GETCHGREPLY_V2](#) or [DRS_MSG_GETCHGREPLY_V7](#)).
- **DRS_SYNC_PAS**: Indicates replication of additional attributes to the partial replica already present on the client.
- **DRS_SPECIAL_SECRET_PROCESSING**: Requests that the server not ship attribute values of attributes that contain secret data. Servers prior to Windows Server 2008 ignore this flag.
- **DRS_GET_ALL_GROUP_MEMBERSHIP**: Requests that the server ship all group membership. If this flag is not specified, then the server ships only universal group membership. Servers prior to Windows Server 2008 ignore this flag.
- **DRS_REF_GCSPN**: Requests that the server adds an entry to [repsTo](#) for the client on the root object of the NC replica that is being replicated. When [repsTo](#) is set using this flag, the server contacts the client using GC SPN (section [2.2.3.2](#)).
- cMaxObjects: Recommended limit on the number of objects to include in the reply.
- cMaxBytes: Recommended limit on the number of bytes to include in the reply.
- msgRequest: The procedure populates corresponding fields in this structure depending on the value that is passed in the parameter version.

```

msgInV8: DRS_MSG_GETCHGREQ_V8
msgRequest: DRS_MSG_GETCHGREQ
prefixEntry: PrefixTableEntry
partialAttrSetSeq: sequence of DSName
schemaSignature: sequence of BYTE
ncType: ULONG

/* NTDSDSA_OPT_DISABLE_INBOUND_REPL defined in
 * [MS-ADTS] section 7.1.1.2.2.1.2.1.1 */
if NTDSDSA_OPT_DISABLE_INBOUND_REPL in DSAObj()!options and
    not DRS_SYNC_FORCED in ulFlags then
    return ERROR_DS_DRA_SINK_DISABLED
endif

msgInV8.cMaxObjects := cMaxObjects
msgInV8.cMaxBytes := cMaxBytes
msgInV8.ulExtendedOp := 0
msgInV8.uuidDsaObjDest := dc.serverGuid
msgInV8.pNC := ADR(nc)
msgInV8.liFsmoInfo := 0

if (ObjExists(nc)) then
    msgInV8.pUpToDateVecDest :=
        ConcreteUTDFromAbstractUTD(nc!replUpToDateVector)
else

```



```

    msgInV8.pUpToDateVecDest := null
endif

/* Fill usnvecFrom and uuidInvocIdSrc fields.
 * usnvecFrom: This field contains the value of the usnVec field in
 * RepsFrom tuple corresponding to the IDL_DRSGetNCChanges server
 * DC, or zeros if no such repsFrom is present.
 * uuidInvocIdSrc: If the usnvecFrom field is not zeros, this field
 * MUST contain the uuidInvocId from the same tuple from which the
 * usnVec field was retrieved. Otherwise, this field contains
 * zeros.*/

if (rf = null) then
    msgInV8.usnvecFrom := 0
    msgInV8.uuidInvocIdSrc := 0
else
    msgInV8.usnvecFrom := rf.usnVec
    msgInV8.uuidInvocIdSrc := rf.uuidInvocId
endif

if AmIRODC() then
    if DRS_WRIT_REP in ulFlags then
        return ERROR_INVALID_PARAMETER
    endif
    ext := ServerExtensions(hDrs)
    if not DRS_EXT_LH_BETA2 in ext.dwFlags and
        msgInV8.pNC^ = SchemaNC() then
        ulFlags := ulFlags + {DRS_WRIT_REP}
    endif
endif

ncType = GetNCType(nc)
if not NCT_GC_PARTIAL in ncType then
    ulFlags := ulFlags + {DRS_GET_ALL_GROUP_MEMBERSHIP}
endif
msgInV8.ulFlags := ulFlags

if (DRS_WRIT_REP in ulFlags) or
    (not DRS_SYNC_PAS in ulFlags) then
    msgInV8.pPartialAttrSetEx := null
else
    msgInV8.pPartialAttrSetEx := ConcretePASFromAbstractPAS(rf.pasData)
endif

/* set msgInV8.pPartialAttrSet field */
if ObjExists(nc) and nc!partialAttributeSet ≠ null then
    msgInV8.pPartialAttrSet := ConcretePASFromAbstractPAS(
        nc!partialAttributeSet)
else
    if (NCT_GC_PARTIAL in ncType and
        NCT_FILTERED_ATTRIBUTE_SET in ncType) then
        msgInV8.pPartialAttrSet := FilteredGCPAS()
    else if NCT_FILTERED_ATTRIBUTE_SET in ncType then
        msgInV8.pPartialAttrSet := FilteredPAS()
    else if NCT_GC_PARTIAL in ncType then
        msgInV8.pPartialAttrSet := GCPAS()
    else
        msgInV8.pPartialAttrSet := null
    endif
endif

```

```

endif

msgInV8.PrefixTableDest = ConcretePTFromAbstractPT(dc.prefixTable)

/* Add schema signature to msgInV8.PrefixTableDest */
schemaSignature := SchemaNC()!schemaInfo
prefixEntry.ndx := 0
prefixEntry.prefix.length := schemaSignature.length
prefixEntry.prefix.element := elements of schemaSignature
Append prefixEntry to msgInV8.PrefixTableDest.pPrefixEntry
msgInV8.PrefixTableDest.PrefixCount :=
    msgInV8.PrefixTableDest.PrefixCount + 1

if version = 5 then
    msgRequest.V5 := msgInV8
    msgRequest.V5.pUpToDateVecDestV1 := msgInV8.pUpToDateVecDest
else
    msgRequest.V8 := msgInV8
endif
endif

```

4.1.10.4.2 ReplSingleObjRequestMsg

```

procedure ReplSingleObjRequestMsg(
    hDrs: DRS_HANDLE,
    version: DWORD,
    nc: DSName,
    object: DSName,
    rf: RepsFrom,
    ulFlags: ULONG,
    cMaxObjects: ULONG,
    cMaxBytes: ULONG,
    fWithSecrets: boolean,
    var msgRequest: DRS_MSG_GETCHGREQ): DWORD

```

Informative summary of behavior: The client can send an [IDL DRSGetNCChanges](#) request to the server to replicate changes from a single object. The ReplSingleObjRequestMsg procedure specifies how the request message is constructed for this operation. The arguments for this method are the same as those for the procedure [ReplicateNCRequestMsg](#), with the following exceptions:

- object: The [DSName](#) type of the object that should be replicated.
- fWithSecrets: The object's secret attributes should be replicated. Only RODCs need to make, and can make, this request.

The procedure returns a Windows error code if it cannot construct msgRequest.

```

msgRequest: DRS_MSG_GETCHGREQ
msgInV8: DRS_MSG_GETCHGREQ_V8
ncType: ULONG

/* An NC replica with root of DSName nc should already exist on the
client */
if (not PartialGCReplicaExists(nc) and
    not FullReplicaExists(nc)) then
    return ERROR_DS_DRA_BAD_NC

```

```

endif

/* Only RODCs are allowed to request secrets explicitly */
if fWithSecrets and not AmIRODC() then
    return ERROR_INVALID_PARAMETER
endif

if fWithSecrets then
    msgInV8.ulExtendedOp := EXOP_REPL_SECRETS
else
    msgInV8.ulExtendedOp := EXOP_REPL_OBJ
endif

if AmIRODC() then
    if DRS_WRIT_REP in ulFlags then
        return ERROR_INVALID_PARAMETER
    endif
    ext := ServerExtensions(hDrs)
    if not DRS_EXT_LH_BETA2 in ext.dwFlags and
        msgInV8.pNC^ = SchemaNC() then
        ulFlags := ulFlags + {DRS_WRIT_REP}
    endif
endif

ncType = GetNCType(nc)
if not NCT_GC_PARTIAL in ncType then
    ulFlags := ulFlags + {DRS_GET_ALL_GROUP_MEMBERSHIP}
endif

msgInV8.ulFlags := ulFlags
msgInV8.cMaxObjects := cMaxObjects
msgInV8.cMaxBytes := cMaxBytes
msgInV8.uuidDsaObjDest := dc.serverGuid
msgInV8.pNC := ADR(object)
msgInV8.liFsmoInfo := 0
msgInV8.pUpToDateVecDest :=
    ConcreteUTDFromAbstractUTD(nc!replUpToDateVector)
msgInV8.pPartialAttrSetEx := null

/* set msgInV8.pPartialAttrSet field */
if ObjExists(nc) and nc!partialAttributeSet ≠ null then
    msgInV8.pPartialAttrSet := ConcretePASFromAbstractPAS(
        nc!partialAttributeSet)
else
    if (NCT_GC_PARTIAL in ncType and
        NCT_FILTERED_ATTRIBUTE_SET in ncType) then
        msgInV8.pPartialAttrSet := FilteredGCPAS()
    else if NCT_FILTERED_ATTRIBUTE_SET in ncType then
        msgInV8.pPartialAttrSet := FilteredPAS()
    else if NCT_GC_PARTIAL in ncType then
        msgInV8.pPartialAttrSet := GCPAS()
    else
        msgInV8.pPartialAttrSet := null
    endif
endif

msgInV8.PrefixTableDest = ConcretePTFromAbstractPT(dc.prefixTable)

```

```

/* Fill usnvecFrom and uuidInvocIdSrc fields.
 * usnvecFrom: This field contains the value of the usnVec field in
 * RepsFrom tuple corresponding to the IDL_DRSGetNCChanges server
 * DC, or zeros if no such repsFrom is present.
 * uuidInvocIdSrc: If the usnvecFrom field is not zeros, this field
 * MUST contain the uuidInvocId from the same tuple from which the
 * usnVec field was retrieved. Otherwise, this field contains
 * zeros.*/

if (rf = null) then
    msgInV8.usnvecFrom := 0
    msgInV8.uuidInvocIdSrc := 0
else
    msgInV8.usnvecFrom := rf.usnVec
    msgInV8.uuidInvocIdSrc := rf.uuidInvocId
endif

if version = 5 then
    msgRequest.V5 := msgInV8
    msgRequest.V5.pUpToDateVecDestV1 := msgInV8.pUpToDateVecDest
else
    msgRequest.V8 := msgInV8
endif

return 0

```

4.1.10.4.3 PerformExtendedOpRequestMsg

```

procedure PerformExtendedOpRequestMsg (
    hDrs: DRS_HANDLE,
    version: DWORD,
    nc: DSName,
    roleOwnerObject: DSName,
    rf: RepsFrom,
    ulFlags: ULONG,
    ulExtendedOp: ULONG,
    cMaxObjects: ULONG,
    cMaxBytes: ULONG,
    var msgRequest: DRS_MSG_GETCHGREQ): DWORD

```

Informative summary of behavior: A client sends an [IDL_DRSGetNCChanges](#) request to a server to perform an extended operation. The procedure `PerformExtendedOpRequestMsg` specifies how the request message is constructed for this operation. The arguments for this method are the same as those for the procedure [ReplicateNCRequestMsg](#), with the following exceptions:

- `ulExtendedOp`: The requested extended operation. The client MUST supply the same value of this field for each request in a given replication cycle. The possible values are:
 - `EXOP_FSMO_REQ_ROLE`, for a FSMO role owner transfer.
 - `EXOP_FSMO_REQ_RID_ALLOC`, for a RID allocation from the RID Master FSMO role owner.
 - `EXOP_FSMO_RID_REQ_ROLE`, for transfer of the RID Master FSMO role.
 - `EXOP_FSMO_REQ_PDC`, for transfer of the PDC FSMO role.

- EXOP_FSMO_ABANDON_ROLE, to request the server to request an extended operation role transfer from the client.
- roleOwnerObject: The client sets this value based on the value of ulExtendedOp, as per the following table:

ulExtendedOp	roleOwnerObject
EXOP_FSMO_REQ_ROLE	The DSName of the FSMO role object.
EXOP_FSMO_REQ_RID_ALLOC	The value of the rIDManagerReference attribute of DefaultNC() .
EXOP_FSMO_RID_REQ_ROLE	The value of the rIDManagerReference attribute of DefaultNC() .
EXOP_FSMO_REQ_PDC	DefaultNC() .
EXOP_FSMO_ABANDON_ROLE	The DSName of the FSMO role object.

The procedure returns a Windows error code if it not able to construct msgRequest.

```

msgInV8: DRS_MSG_GETCHGREQ_V8
serverObj: DSName
computerObj: DSName
ridSetReferences: DSName

/* An NC replica with root nc must already exist on the client */
if (not MasterReplicaExists(nc)) then
    return ERROR_DS_DRA_BAD_NC
endif

msgInV8.ulFlags := ulFlags
msgInV8.cMaxObjects := cMaxObjects
msgInV8.cMaxBytes := cMaxBytes
msgInV8.ulExtendedOp := ulExtendedOp
msgInV8.uuidDsaObjDest := dc.serverGuid
msgInV8.pNC := ADR(roleOwnerObject)
msgInV8.pUpToDateVecDest :=
    ConcreteFromAbstractUTD(nc!replUpToDateVector)
msgInV8.pPartialAttrSetEx := null
msgInV8.pPartialAttrSet := null
msgInV8.PrefixTableDest := 0

if (ulExtendedOp = EXOP_FSMO_REQ_RID_ALLOC) then
    serverObj := DSAObj()!parent
    computerObj := serverObject!serverReference
    ridSetReferences := computerObj!ridSetReferences
    if ((not ridSetReferences = null) and
        (ridSetReferences!isDeleted = false)) and
        (not ridSetReferences!rIDNextRid = null) and
        (not ridSetReferences!rIDNextRid = 0) and
        (not ridSetReferences!rIDAllocationPool = null)) then
        msgInV8.liFsmoInfo := ridSetReferences!rIDAllocationPool
    else
        msgInV8.liFsmoInfo := 0
    endif
else
    msgInV8.liFsmoInfo := 0
endif
endif

```

```

/* Fill usnvecFrom and uuidInvocIdSrc fields.
 * usnvecFrom: This field contains the value of the usnVec field in
 * RepsFrom tuple corresponding to the IDL_DRSGetNCChanges server
 * DC, or zeros if no such repsFrom is present.
 * uuidInvocIdSrc: If the usnvecFrom field is not zeros, this field
 * MUST contain the uuidInvocId from the same tuple from which the
 * usnVec field was retrieved. Otherwise, this field contains
 * zeros.*/

if (rf = null) then
    msgInV8.usnvecFrom := 0
    msgInV8.uuidInvocIdSrc := 0
else
    msgInV8.usnvecFrom := rf.usnVec
    msgInV8.uuidInvocIdSrc := rf.uuidInvocId
endif

if version = 5 then
    msgRequest.V5 := msgInV8
    msgRequest.V5.pUpToDateVecDestV1 := msgInV8.pUpToDateVecDest
else
    msgRequest.V8 := msgInV8
endif

return 0

```

4.1.10.5 Server Behavior of the IDL_DRSGetNCChanges Method

Informative summary of behavior: The [IDL_DRSGetNCChanges](#) method returns a response for a single request in a cycle.

This method is invoked through the [drsuapi](#) RPC interface. It is also invoked as a local procedure for requests that are received using the SMTP transport ([\[MS-SRPL\]](#)).

```

ULONG
IDL_DRSGetNCChanges(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch is(dwInVersion)]
        DRS_MSG_GETCHGREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch is(*pdwOutVersion)]
        DRS_MSG_GETCHGREPLY *pmsgOut)

err: ULONG
ext: DRS_EXTENSIONS_INT
msgIn: DRS_MSG_GETCHGREQ_V8
ncRoot: DSName
obj: DSName
msgOut: DRS_MSG_GETCHGREPLY_V6
schemaSignature: sequence of BYTE
prefixEntry: PrefixTableEntry
responseSmtAddress: unicodestring
fullReplicaFlags: set of integer

```

```

/* Default response, in case of error. */
pdwOutVersion^ := 1
pmsgOut^ := 0

ext := ClientExtensions(hDrs)
if ext.dwReplEpoch ≠ dc.replEpoch then
    return ERROR_DS_DIFFERENT_REPL_EPOCHS
endif

err := TransformInput(ext, dwInVersion, pmsgIn^, msgIn,
    pdwOutVersion, responseSmtpAddress)
if err ≠ 0 then
    return err
endif

/* Perform access checks. */
if msgIn.pNC = null then
    return ERROR_INVALID_PARAMETER
endif
ncRoot := GetObjectNC(msgIn.pNC^)

if ncRoot = null then
    return ERROR_DS_CANT_FIND_EXPECTED_NC
endif

if IsGetNCChangesPermissionGranted(msgIn) == FALSE then
    return ERROR_DRA_ACCESS_DENIED
endif

/* Validate inputs. */
obj := msgIn.pNC^
if AmILHServer() = false then
    /* Downlevel OS does not understand
       DRS_SPECIAL_SECRET_PROCESSING flags. They just ignore it.
    */
    msgIn.ulFlags := msgIn.ulFlags - {DRS_SPECIAL_SECRET_PROCESSING}
endif

if msgIn.ulExtendedOp = 0 then
    /* Validate normal replication request. */
    if not FullReplicaExists(obj) and not PartialGCReplicaExists(obj)
    then
        return ERROR_DS_CANT_FIND_EXPECTED_NC
    endif
else
    /* Validate extended operation request. */
    if not ObjExists(obj) then
        return ERROR_DS_CANT_FIND_EXPECTED_NC
    endif
endif

if AmILHDC() then
    if (msgIn.pPartialAttributeSet = null and
        msgIn.pPartialAttributeSetEx = null) then
        fullReplicaRequest := true
    else
        fullReplicaRequest := false
    endif
endif

```

```

else
    if (DRS_WRITE_REP in msgIn.ulFlags) then
        fullReplicaRequest := true
    else
        fullReplicaRequest := false
    endif
endif

if (fullReplicaRequest) then
    /* Validate Full Replica request. */
    if not IT_WRITE in obj!instanceType then
        return ERROR_DRA_SOURCE_IS_PARTIAL
    endif
    if DRS_SYNC_PAS in msgIn.ulFlags then
        return ERROR_INVALID_PARAMETER
    endif
else
    /* Validate Partial Replica request. */
    if msgIn.pPartialAttrSet = null
        or msgIn.pPartialAttrSet.cAttrs = 0 then
        return ERROR_INVALID_PARAMETER
    endif
    if DRS_SYNC_PAS in msgIn.ulFlags and
        (msgIn.pPartialAttrSetEx = null
        or msgIn.pPartialAttrSetEx.cAttrs = 0) then
        return ERROR_INVALID_PARAMETER
    endif
    if msgIn.PrefixTableDest.PrefixCount = 0 then
        return ERROR_INVALID_PARAMETER
    endif
endif

if IT_NC_GOING in ncRoot!instanceType
    /* NC replica is no longer accepting requests. */
    return ERROR_DRA_NO_REPLICA
endif

if msgIn.uuidInvocIdSrc ≠ DSAObj()!invocationId then
    msgIn.usnvecFrom := 0
endif

/* Construct response. */
if msgIn.ulExtendedOp = 0 then
    /* Perform normal replication. */
    err := GetReplChanges(hDrs, msgIn, msgOut)
else
    /* Perform extended operation. Errors are returned in
    * msgOut.ulExtendedErr. */
    ProcessFsmoRoleRequest(hDrs, msgIn, msgOut)
    err := 0
endif

if err = 0 then
    msgOut.pNC := msgIn.pNC
    msgOut.usnvecFrom := msgIn.usnvecFrom
    msgOut.uuidDsaObjSrc := dc.serverGuid
    msgOut.PrefixTableSrc := ConcretePTFromAbstractPT(dc.prefixTable)
    msgOut.uuidInvocIdSrc := DSAObj()!invocationId

```



```

/* Sort msgOut.rgValues into ascending order. */
SortResponseLinks(msgOut)

/* Add schema signature to msgOut.PrefixTableSrc. */
schemaSignature := SchemaNC()!schemaInfo
prefixEntry.ndx := 0
prefixEntry.prefix.length := schemaSignature.length
prefixEntry.prefix.element := elements of schemaSignature
Append prefixEntry to msgOut.PrefixTableSrc.pPrefixEntry
msgOut.PrefixTableSrc.PrefixCount :=
    msgOut.PrefixTableSrc.PrefixCount+1

err := TransformOutput(msgOut, msgIn.ulFlags, pdwOutVersion^,
    pmsgOut)
endif

if responseSmtpAddress ≠ null then
    Send the response using the SMTP transport to
    responseSmtpAddress
endif

return err

```

4.1.10.5.1 TransformInput

```

procedure TransformInput(
    ext: DRS_EXTENSIONS_INT,
    requestVersion: DWORD,
    requestUnion: DRS_MSG_GETCHGREQ,
    var nativeRequest: DRS_MSG_GETCHGREQ_V8,
    pdwOutVersion: ADDRESS OF DWORD,
    var responseSmtpAddress: unicodestring): ULONG

```

Informative summary of behavior: The TransformInput method transforms the received request message into a V8 request, which is a superset of the supported request messages.

```

partialAttrObjs: sequence of DSName
i: integer

responseSmtpAddress := null

if requestVersion = 8
    /* Windows Server 2003 RPC request. */
    nativeRequest := requestUnion.V8

    if not DRS_EXT_GETCHGREQ_V6 in ext.dwFlags then
        return ERROR_REVISION_MISMATCH
    endif
    pdwOutVersion^ := 6
else if requestVersion = 4 then
    /* Windows Server 2003 SMTP request. */
    responseSmtpAddress := requestUnion.V7.pmtxReturnAddress^.mtx_name
    nativeRequest := requestUnion.V7.V3
    nativeRequest.pUpToDateVecDest :=
        requestUnion.V7.V3.pUpToDateVecDestV1

```

```

    nativeRequest.pPartialAttrSet := requestUnion.V7.pPartialAttrSet
    nativeRequest.pPartialAttrSetEx :=
        requestUnion.V7.pPartialAttrSetEx
    nativeRequest.PrefixTableDest := requestUnion.V7.PrefixTableDest
    pdwOutVersion^ := 6
else if requestVersion = 5 then
    /* Windows 2000 RPC request. */
    nativeRequest := requestUnion.V5
    nativeRequest.pUpToDateVecDest :=
        requestUnion.V5.pUpToDateVecDestV1
    nativeRequest.pPartialAttrSetEx := null
    nativeRequest.PrefixTableDest :=
        ConcretePTFromAbstractPT(dc.prefixTable)

    if ({DRS_WRIT_REP} ∩ nativeRequest.ulFlags) = null then
        nativeRequest.pPartialAttrSet := GCPAS()
    endif

    pdwOutVersion^ := 1

else if requestVersion = 4 then
    /* Windows 2000 SMTP request. */
    responseSmtpAddress := requestUnion.V4.pmtxReturnAddress^.mtx_name
    nativeRequest := requestUnion.V4.V3
    nativeRequest.pUpToDateVecDest :=
        requestUnion.V4.V3.pUpToDateVecDestV1
    nativeRequest.pPartialAttrSet :=
        requestUnion.V4.V3.pPartialAttrVecDestV1
    nativeRequest.pPartialAttrSetEx := null
    pdwOutVersion^ := 1
else
    /* Unsupported request. */
    return ERROR_REVISION_MISMATCH
endif

if ({DRS_WRIT_REP} ∩ nativeRequest.ulFlags) ≠ null then
    nativeRequest.ulFlags :=
        nativeRequest.ulFlags + {DRS_GET_ALL_GROUP_MEMBERSHIP}
endif

if (responseSmtpAddress = null) ≠ (not DRS_MAIL_REP in
    nativeRequest.ulFlags) then
    return ERROR_INVALID_PARAMETER
endif

return 0

```

4.1.10.5.2 GetReplChanges

```

procedure GetReplChanges(
    hDrs: DRS_HANDLE,
    msgIn: DRS_MSG_GETCHGREQ_V8,
    var msgOut: DRS_MSG_GETCHGREPLY_V6): ULONG

```

Informative summary of behavior: The GetReplChanges procedure processes a normal replication request; that is, an [IDL DRSGetNCChanges](#) request that is not a FSMO role request. It adds changed objects and link values to the response, subject to the scope (msgIn.pNC^, msgIn.ulFlags), filter criteria (msgIn.pUpToDateVecDest, msgIn.ulFlags, msgIn.pPartialAttrSet, msgIn.pPartialAttrSetEx), response limits (msgIn.cMaxObjects, msgIn.cMaxBytes), and the previous server cookie (msgIn.usnvecFrom) in the request. It returns 0 if successful, otherwise a Windows error code.

```

ncRoot: DSName
pUtd: ADDRESS OF UPTODATE_VECTOR_V1_EXT
scope: set of DSName
attribute: ATTRTYP
partialAttrs: set of ATTRTYP
partialAttrSetEx: set of ATTRTYP
changedObjs: set of ObjAtts
changedLinks: set of ObjAttVal
responseObjs: set of ObjAtts
responseLinks: set of ObjAttVal
anc: ObjAtts
clientDSA : DSName
updRefs: DRS_MSG_UPDREFS /* See IDL_DRSUpdateRefs structures. */

if AmIRODC() then
    return ERROR_DS_DRA_SOURCE_DISABLED
endif

/* check whether outbound replication is disabled */
/* NTDSDSA_OPT_DISABLE_OUTBOUND_REPL defined in
 * [MS-ADTS] section 7.1.1.2.2.1.2.1.1 */
if NTDSDSA_OPT_DISABLE_OUTBOUND_REPL in DSAObj()!options and
    not DRS_SYNC_FORCED in msgIn.ulFlags then
    return ERROR_DS_DRA_SOURCE_DISABLED
endif

ncRoot := GetObjectNC(msgIn.pNC^)

/* Determine stamp filter to apply to the response. */
if DRS_FULL_SYNC_PACKET in msgIn.ulFlags then
    pUtd := null
else
    pUtd := msgIn.pUpToDateVecDest
endif

/* Determine attribute filters to apply to the response. */
if msgIn.pPartialAttrSet = null
    partialAttrs := null
else
    partialAttrs := {}
    foreach id in msgIn.pPartialAttrSet
        attribute := LocalAttidFromRemoteAttid(msgIn.PrefixTableDest, id)
        if (not IT_WRITE in ncRoot!instanceType) and
            (not attribute in ncRoot!partialAttributeSet) then
            return ERROR_DS_DRA_INCOMPATIBLE_PARTIAL_SET
        endif
        partialAttrs := partialAttrs + { attribute }
    endfor
endif
if msgIn.pPartialAttrSetEx = null

```

```

    partialAttrsEx := null
else
    partialAttrsEx := {}
    foreach id in msgIn.pPartialAttrSetEx
        attribute := LocalAttidFromRemoteAttid(msgIn.PrefixTableDest, id)
        if (not IT_WRITE in ncRoot!instanceType) and
            (not attribute in ncRoot!partialAttributeSet) then
            return ERROR_DS_DRA_INCOMPATIBLE_PARTIAL_SET
        endif
        partialAttrsEx:= partialAttrsEx + { attribute }
    endfor
endif

/* Get nTDSDSA of the client */
clientDSA := select one o from ConfigNC() where
    o!objectGUID = msgIn.uuidDsaObjDest

/* Get the set of all objects that are in scope. */
scope := GetReplScope(msgIn)

/* Get object and link value changes in scope. */
GetChangesInScope(scope, pUtd, partialAttrs, partialAttrsEx,
    changedObjs, changedLinks)

/* Choose subsets of changedObjs and changedLinks to include in this
 * response. Set usnvecTo and fMoreData in out to indicate the
 * subset to return in the next response, if any. */
GetResponseSubset(msgIn, changedObjs, changedLinks, msgOut,
    responseObjs, responseLinks)

/* Add responseObjs to response. */
foreach o in responseObjs
    if DRS_GET_ANC in msgIn.ulFlags then
        /* Ancestors predicate: insert any changes to parent before any
         * changes to child. */
        foreach n in Ancestors of o.obj, most distant ancestor first
            anc := select one a from changedObjs where a.obj = n
            if anc ≠ null then
                AddObjToResponse(
                    hDrs, anc, ncRoot, msgIn.ulFlags, 0, clientDSA, msgOut)
            endif
        endfor
    endif
    AddObjToResponse(
        hDrs, o, ncRoot, msgIn.ulFlags, 0, clientDSA, msgOut)
endfor

/* Add responseLinks to response. */
foreach v in responseLinks
    if DRS_GET_ANC in msgIn.ulFlags then
        /* Ancestors predicate: insert any changes to object before any
         * changes to its link values. */
        anc := select one a from changedObjs where a.obj = v.obj
        if anc ≠ null then
            AddObjToResponse(hDrs, anc, ncRoot, msgOut)
        endif
    endif
    AddLinkToResponse(v, msgIn, msgOut)

```

```

endifor

if not msgOut.fMoreData
  msgOut.pUpToDateVecSrc := The cycle goal, as specified in
    section 4.1.10.1.2.
endif

if DRS_GET_NC_SIZE in msgIn.ulFlags then
  msgOut.cNumNcSizeObjects := Approximate number of objects in
    NC replica msgIn.pNC^
  msgOut.cNumNcSizeValues := Approximate number of link values
    with stamps in NC replica msgIn.pNC^
endif

if (DRS_ADD_REF in msgIn.ulFlags ≠ null) then
  /* Client has requested the server to add a repsTo entry. */
  updRefs.uuidDsaDes := msgIn.uuidDsaObjDest
  updRefs.pszDsaDest := NetworkAddress of DC corresponding to
    msgIn.uuidDsaObjDest
  updRefs.ulOptions := {DRS_ASYNC_OP, DRS_ASYNC_OP,
    DRS_GETCHG_CHECK} +
    { msgIn.ulFlags ∩ {DRS_ADD_REF, DRS_REF_GCSPN}}
  if DRS_WRIT_REP in msgIn.ulFlags then
    updRefs.ulOptions := updRefs.ulOptions | DRS_WRIT_REP
  endif
  Using updRefs update repsTo on msgIn.pNC^ as defined in
    IDL_DRSUpdateRefs.
endif

return 0

```

4.1.10.5.3 GetReplScope

```

procedure GetReplScope(
  msgIn: DRS_MSG_GETCHGREQ_V8): set of DSName

```

Informative summary of behavior: The GetReplScope method returns the set of objects considered for normal replication: the objects in the requested NC replica (msgIn.pNC^) or a subset thereof, as indicated by the request flags (msgIn.ulFlags). If the DRS_ASYNC_REP request flag is specified, the subset includes only the NC root. If the DRS_CRITICAL_ONLY request flag is specified, the subset includes only those objects with [isCriticalSystemObject](#) = true and their ancestors.

```

scope: set of DSName
ncRoot: DSName
anc: DSName
ncRoot := GetObjectNC(msgIn.pNC^)
if DRS_ASYNC_REP in msgIn.ulFlags then
  scope := {ncRoot}
else if DRS_CRITICAL_ONLY in msgIn.ulFlags then
  scope := select all o from subtree-ts-included ncRoot where
    o.isCriticalSystemObject = true
  foreach o in scope
    foreach anc in Ancestors of o
      if not anc in scope then
        scope := scope + {anc}
      endif
    endforeach
  endforeach
endif

```

```

        endif
    endif
endfor
else
    scope := select all o from subtree-ts-included ncRoot where true
    foreach o in ncRoot!subRefs
        scope := scope + {o}
    endfor
endif
return scope

```

4.1.10.5.4 GetChangesInScope

```

procedure GetChangesInScope(
    scope: set of DSName,
    pUtd: ADDRESS OF UPTODATE_VECTOR_V1_EXT,
    partialAttrs: set of ATTRTYP,
    partialAttrsEx: set of ATTRTYP,
    var changedObjs: set of ObjAtts,
    var changedLinks: set of ObjAttVal)

```

Informative summary of behavior: The GetChangesInScope method inspects the objects in scope and returns the object and link value updates that must be sent to the client over the course of the replication cycle, as determined by the up-to-date vector (pUtd) and the partial replica attribute filters (partialAttrs and partialAttrsEx).

```

o: DSName
a: ATTRTYP
attrs: set of ATTRTYP
stamp: AttributeStamp

/* Get the set of objects in scope with attribute stamps that the
 * client did not have knowledge of at the beginning of this
 * cycle. */
changedObjs := {}
foreach o in scope
    attrs := {}
    foreach a of o's object class
        stamp := AttrStamp(o, a)
        if stamp ≠ null and
            (a = instanceType or a = proxiedObjectName
             or not FilterAttribute(o, a, stamp, pUtd,
                                   partialAttrs, partialAttrsEx) then
            attrs := attrs + {a}
        endif
    endfor
    if attrs - (attrs ∩ {instanceType, proxiedObjectName}) ≠ {} then
        changedObjs := changedObjs + [obj: o, atts: attrs]
    endif
endfor

/* Get the set of link values in scope with stamps that the client
 * did not have knowledge of at the beginning of this cycle. */
if (GetForestFunctionalLevel() ≥ 1) then
    changedLinks := {}

```

```

foreach o in scope
  foreach a in Link Attributes of o's object class
    foreach v in GetAttVals(o, a, true)
      stamp := LinkStamp(o, a, v)
      /* If v was last updated in win2k forest mode
       * then it does not have LinkValueStamp associated with it.
       * LinkStamp() returns null in that case and this value will
       * not be added to changedLinks.
       */
      if stamp ≠ null
        and not FilterAttribute(o, a, stamp, pUtd,
          partialAttrs, partialAttrsEx) then
          changedLinks := changedLinks + [obj: o, att: a, val: v]
        endif
      endfor
    endfor
  endfor
endif

```

4.1.10.5.5 FilterAttribute

```

procedure FilterAttribute(
  o: DSName,
  attribute: ATTRTYP,
  s: AttributeStamp,
  pUtd: ADDRESS OF UPTODATE_VECTOR_V1_EXT,
  partialAttrs: set of ATTRTYP,
  partialAttrsEx: set of ATTRTYP): boolean

```

Informative summary of behavior: The FilterAttribute method determines whether an update (attribute or link value) that is in scope should be filtered out of the set of changes to send in the replication cycle. The rules are as follows:

- If the client's up-to-date vector pUtd asserts that the client has already applied the update with stamp s, the update is filtered out, provided that attribute is not in the partialAttrsEx set. The elements of partialAttrsEx are not subject to filtering by the up-to-date vector.
- If partialAttrs is not null (indicating the client has a partial replica) and attribute is not in partialAttrs + partialAttrsEx, then the update is filtered out.
- If partialAttrs is not null, attribute is [member](#), o is of class [group](#), and o is not a universal group, then the update is filtered out.
- If attribute is the naming attribute for the object class of o, then the update is filtered out. See, in the following examples, [cn](#) for objects of class [container](#).

```

filtered: boolean
cursor: UPTODATE_CURSOR_V2
filtered := false
if pUtd ≠ null and partialAttrsEx ≠ null
  and not attribute in partialAttrsEx then
    /* Filter updates with stamps that the client's up-to-date vector
     * asserts the client has already applied to its NC replica.
     */
    cursor := select one c from pUtd^.rgCursors where c.uuidDsa =

```

```

        s.uuidOriginating
    if cursor ≠ null and cursor.usnHighPropUpdate ≥ s.usnOriginating
        then
            filtered := true
        endif
    endif
    if not filtered and partialAttrs ≠ null then
        /* Filter updates to attributes that are not in the client's
         * partial replica.
         */
        if not attribute in partialAttrs + partialAttrsEx then
            filtered := true
        endif
    endif
    if not filtered and partialAttrs ≠ null and attribute = member then
        /* Filter updates to the member attribute from the client's
         * partial replica if the group is not a universal group.
         */
        if group in o!objectClass and
            not GROUP_TYPE_UNIVERSAL_GROUP in o!groupType then
            filtered := true
        endif
    endif
    if not filtered then
        /* Filter updates to the naming attribute of o. */
        if attribute = o!rdnType then
            filtered := true
        endif
    endif
    endif
    return filtered

```

4.1.10.5.6 GetResponseSubset

```

procedure GetResponseSubset(
    msgIn: DRS_MSG_GETCHGREQ_V8,
    changedObjs: set of ObjAtts,
    changedLinks: set of ObjAttVal,
    var msgOut: DRS_MSG_GETCHGREPLY_V6,
    var responseObjs: set of ObjAtts,
    var responseLinks: set of ObjAttVal)

```

The GetResponseSubset method selects subsets of the changed objects and link values to include in this response. This method utilizes the cookie msgIn.usnvecFrom—which is zero or a value returned to the client in a previous response—and the client-requested limits msgIn.cMaxObjects and msgIn.cMaxBytes to determine the subsets. This method then sets msgOut.usnvecTo to a new cookie that the client presents in its next request as msgIn.usnvecFrom.

The server SHOULD [<18>](#) choose a subset such that the response will contain no more objects than msgIn.cMaxObjects and no more bytes (before any compression is applied) than msgIn.cMaxBytes.

It is valid for the response to contain no objects or link values. It is valid for an object or a link value to appear multiple times in a single response, and the object's attribute values or the link values need not be identical. It is valid for an object or a link value to appear both in the current response and in an earlier response in the same cycle, and the object's attribute values or the link values need not be identical.

If the server determines, by using state that is maintained via msgIn.usnvecFrom and msgOut.usnvecTo, that inclusive of what it is sending in this response, it will have sent at least changedObjs and changedLinks to the client, then it concludes the cycle by returning with msgOut.fMoreData = false. Therefore, if this is the first response message of a cycle, the server only returns with msgOut.fMoreData = false if responseObjs = changedObjs and responseLinks = changedLinks.

Subject to resource constraints on the server, if neither changedObjs nor changedLinks increases during a sequence of calls, the server eventually returns msgOut.fMoreData = false.

4.1.10.5.7 AddObjToResponse

```
procedure AddObjToResponse(
    hDrs: DRS_HANDLE,
    o: ObjAtts,
    ncRoot: DSName,
    ulFlags: set of integer,
    ulExtendedOp: DWORD,
    clientDSA: DSName,
    var msgOut: DRS_MSG_GETCHGREPLY_V6)
```

Informative summary of behavior: The AddObjToResponse method constructs a [REPLENTINFLIST](#) structure for a changed object and appends it to the response.

```
re: REPLENTINFLIST
pAttr: ADDRESS OF ATTR
attribute: ATTRTYP
attrObj: DSName
attrVals: sequence of attribute values
i: DWORD
j: DWORD

/* Construct a REPLENTINFLIST to represent the changes. */
re := all zeros
re.fIsNcPrefix := (o.obj = ncRoot)
if name in o.atts and not re.fIsNcPrefix then
    re.pParentGuid := ADR(o.obj!parent)
endif
re.EntInf.pName := ADR(o.obj)
re.EntInf.AttrBlock.pAttr := array of ATTR of size o.atts.length
re.EntInf.AttrBlock.attrCount := o.atts.length
re.pMetaDataExt := PROPERTY_META_DATA_EXT_VECTOR
    with rgMetaData of size o.atts.length
re.pMetaDataExt^.cNumProps := o.atts.length
for i := 0 to o.atts.length - 1
    attribute := o.atts[i]
    attrObj := SchemaObj(attribute)
    re.pMetaDataExt^.rgMetaData[i] = AttrStamp(o.obj, attribute)
    pAttr := ADR(re.EntInf.AttrBlock.pAttr[i])
    pAttr^.attrTyp := attribute
    pAttr^.AttrVal.valCount := 0
    if AmILHServer() and
        DRS_SPECIAL_SECRET_PROCESSING in ulFlags and
        IsSecretAttribute(attribute) then
        /* secret attribute, send a null value */
        pAttr^.AttrVal.pAVal = null
```

```

    re.pMetaDataExt^.rgMetaData[i].timeChanged = 0
else
    /* not a secret attribute */
    attrVals := GetAttrVals(o, attribute, false)
    pAttr^.AttrVal.pAVal := ARRAY OF ATTRVAL
        WITH SIZE attrVals.length
    for j := 0 to attrVals.length - 1
        /* If attribute is a link value attribute, then add it to the
        * response here only if it does not have a LinkValueStamp
        * associated with it. This can happen if the current forest
        * functional level is DS_BEHAVIOR_WIN2000 or the attribute
        * value attrVals[j] was last updated when the forest
        * functional level was DS_BEHAVIOR_WIN2000. If the
        * attribute value has a LinkValueStamp associated with it,
        * then it will be sent in the response packet by method
        * AddLinkToResponse. Forest functional levels are listed
        * in [MS-ADTS] section 7.1.4.4.
        */
        if (attrObj!linkID = null) or
            ((attrObj!linkID ≠ null) and
                (LinkStamp(o.obj, attribute, attrVals[j]) = null) then
            pAttr^.AttrVal.pAVal[j] := ATTRVALFromValue(
                attrVals[j], Syntax(attribute), dc.prefixTable)
            pAttr^.AttrVal.valCount := pAttr^.AttrVal.valCount + 1
        endif
    endfor /* j := */
endif
EncryptValuesIfNecessary(hDrs, pAttr^)
/* if we are sending secrets to RODC then log it to revealed
* list */
if (EXOP_REPL_SECRETS in ulExtendedOp) then
    UpdateRevealedList(clientDSA, o.obj, attribute)
endif
endfor /* i := */

/* Add re to the response. */
Add re to the end of the linked list msgOut.pObjects
msgOut.cNumObjects := msgOut.cNumObjects + 1

```

4.1.10.5.8 UpdateRevealedList

```

procedure UpdateRevealedList(
    rodcdsa: DSName,
    revealedObject: DSName,
    attribute: ATTRTYP)

```

Informative summary of behavior: The UpdateRevealedList method adds or updates an entry for the attribute of the object revealedObject on the [msDS-RevealedUsers](#) attribute of the computer object that corresponds to the [nTDSDSA](#) object rodcdsa. The [msDS-RevealedUsers](#) attribute is of type [DNBinary](#). The binary portion of attribute value contains the **PROPERTY META DATA** structure in its binary form. The DN portion of attribute value contains revealedObject.

```

serverObj: DSName
computerObj: DSName

```

```

attrSchemaObj: DSName
revealedObjectsNew: set of DNBinary
obj: DNBinary
propMetadata: PROPERTY_META_DATA
propMetadataCurrent: PROPERTY_META_DATA
newRevealedObjectVal: DNBinary

/* Revealed list has entries only for secret attributes */
if not IsSecretAttribute(attribute) then
    return
endif

/* Get the computer object corresponding to nTDSDSA object rodcdsa */
serverObj := rodcdsa!parent
computerObj := serverObj!serverReference

/* filter superceded entries from the msDS-RevealedUsers set */
revealedObjectsNew := {}
foreach obj in computerObj!msDS-RevealedUsers
    propMetadata := loophole(obj.binary, PROPERTY_META_DATA)
    if (obj.object_dn ≠ revealedObject) or
        (propMetadata.attrType ≠ attribute) or
        (StampCompare(propMetadata.propMetadataExt,
            AttrStamp(revealedObject, attribute) > 0) then
        revealedObjectsNew := revealedObjectsNew + { obj }
    endif
endfor

/* add the new entry to the set */
propMetadataCurrent.attrType := attribute
propMetadataCurrent.propMetadataExt :=
    AttrStamp(revealedObject, attribute)
newRevealedObjectVal.binary :=
    loophole(propMetadataCurrent, sequence of byte)
newRevealedObjectVal.object_dn := revealedObject
revealedObjectsNew := revealedObjectsNew + { newRevealedObjectVal }

/* set attribute value to new set */
computerObj!msDS-RevealedUsers := revealedObjectsNew

```

4.1.10.5.9 AddLinkToResponse

```

procedure AddLinkToResponse(
    v: ObjAttVal,
    msgIn: DRS_MSG_GETCHGREQ_V8,
    var msgOut: DRS_MSG_GETCHGREPLY_V6)

```

Informative summary of behavior: The AddLinkToResponse method constructs a [REPLVALINE](#) structure for a changed link value and appends it to the response.

```

rv: REPLVALINF
rvs: sequence of REPLVALINF
stamp: LinkValueStamp
filterGroups: boolean

```

```

filterGroups := true;

if AmILHServer() then
    if DRS_GET_ALL_GROUP_MEMBERSHIP in msgIn.ulFlags then
        filterGroups := false
    endif
else
    if DRS_WRITE_REP in msgIn.ulFlags then
        filterGroups := false
    endif
endif

if filterGroups = true and
    group in v.obj!objectClass and
    not GROUP_TYPE_UNIVERSAL_GROUP in v.obj!groupType and
    v.att = member
/* non-universal group membership is replicated out unless
   explicitly requested */
    return
endif

/* Construct a REPLVALINF to represent the changes to send. */
rv.pObject = v.obj
rv.attrType := v.att
rv.AVal := ATTRVALFromValue(v.val, Syntax(v.att), dc.prefixTable)
stamp := LinkStamp(v.obj, v.att, v.val)
rv.fIsPresent := stamp.timeDeleted = 0
rv.MetaData := stamp

/* Add rv to the response. */
if msgOut.cNumValues ≠ 0
    Copy elements from msgOut.rgValues to rvs
endif
rvs[msgOut.cNumValues] := rv
msgOut.rgValues := elements of rvs
msgOut.cNumValues := msgOut.cNumValues + 1

```

4.1.10.5.10 EncryptValuesIfNecessary

```

procedure EncryptValuesIfNecessary(
    hDrs: DRS_HANDLE,
    var attr: ATTR)

```

Informative summary of behavior: The EncryptValuesIfNecessary method encrypts the values of attributes that contain secret data. This method performs the encryption by using an MD5 digest (as specified in [\[RFC1321\]](#)), a CRC32 checksum (as specified in [\[ISO/IEC 13239\]](#)), and an RC4 stream cipher (as specified in [\[RC4\]](#)). This encryption is in addition to the encryption that is provided by RPC privacy.

```

sessionKey: sequence of BYTE
i: integer
salt: sequence of BYTE
md5Context: MD5_CTX
crc: ULONG
pPayload: ADDRESS OF ENCRYPTED_PAYLOAD

```

```

if not IsSecretAttribute(attr.attrTyp) then
    /* No additional encryption necessary. */
    return
endif

/* Get session key associated with the RPC connection. */
sessionKey := session key associated with security context of hDrs,
    as specified by [MS-RPCE] section 3.3.1.5.2 and [MS-KILE]
    section 3.1.1.2

/* Encrypt each value of this attribute. */
for i := 0 to attr.AttrVal.valCount - 1
    salt := randomly generated 128-bit number

    /* Calculate checksum of the clear value. */
    crc := CRC32 [ISO/IEC 13239] of the attr.AttrVal.pAVal[i].valLen
        bytes starting at attr.AttrVal.pAVal[i].pVal

    /* Compute encryption key. */
    MD5Init(md5Context)
    MD5Update(md5Context, sessionKey, sessionKey.length)
    MD5Update(md5Context, salt, 16)
    MD5Final(md5Context)

    /* Construct payload, encrypting its contents with the exception of
     * the Salt field. */
    pPayload := New ENCRYPTED_PAYLOAD, sized to hold
        attr.AttrVal.pAVal[i].valLen bytes in the EncryptedData field
    pPayload^.Salt := salt
    pPayload^.Checksum := crc
    Copy attr.AttrVal.pAVal[i].valLen bytes from
        attr.AttrVal.pAVal[i].pVal to pPayload^.EncryptedData
    Encrypt attr.AttrVal.pAVal[i].valLen + 4 bytes starting at the
        address of pPayload^.Checksum using the RC4 stream cipher
        algorithm [RC4] with encryption key md5Context.digest

    /* Replace the clear value with the encrypted value. */
    attr.AttrVal.pAVal[i].pVal := pPayload
    attr.AttrVal.pAVal[i].valLen := attr.AttrVal.pAVal[i].valLen + 20
endfor

```

4.1.10.5.11 ProcessFsmoRoleRequest

```

procedure ProcessFsmoRoleRequest(
    hDrs: DRS_HANDLE,
    msgIn: DRS_MSG_GETCHGREQ_V8,
    var msgOut: DRS_MSG_GETCHGREPLY_V6)

```

Informative summary of behavior: The ProcessFsmoRoleRequest method performs the requested FSMO role operation indicated by msgIn.ulExtendedOp.

```

fsmoObj: DSName
clientDsaObj: DSName
serverObj: DSName: DSName

```

```

rodObj: DSNAME: DSName
clientComputerObj: DSName
clientRidSetObj: DSName
ownerDsaObj: DSName
scope: set of DSName
ridAllocLoHi: ULONGLONG
ridAllocHi: DWORD
ridReqHi: DWORD
ridAvailLoHi: ULONGLONG
ridAvailLo: DWORD
ridAvailHi: DWORD
changedObjs: set of ObjAtts
changedLinks: set of ObjAttVal

fsmoObj := msgIn.pNC^
if not ObjExists(fsmoObj) then
    msgOut.ulExtendedRet := EXOP_ERR_UPDATE_ERR
    return
endif
clientDsaObj := select one o from ConfigNC()where
    o!objectGUID = msgIn.uuidDsaObjDest
if clientDsaObj = null then
    msgOut.ulExtendedRet := EXOP_ERR_UNKNOWN_CALLER
    return
endif

scope := {}

if msgIn.ulExtendedOp in {EXOP_FSMO_REQ_ROLE, EXOP_FSMO_REQ_PDC,
    EXOP_FSMO_RID_REQ_ROLE} then
    /* Change the FSMO role owner from the server to the client. */
    if fsmoObj!fsmoRoleOwner ≠ DSAObj() then
        msgOut.ulExtendedRet := EXOP_ERR_NOT_OWNER
        return
    endif

    fsmoObj!fsmoRoleOwner := clientDsaObj
    scope := GetRoleScope(fsmoObj)
else if msgIn.ulExtendedOp = EXOP_FSMO_ABANDON_ROLE then
    /* Request a change in the FSMO role owner from the current owner
    * to the server. */
    if fsmoObj!fsmoRoleOwner ≠ DSAObj() then
        ownerDsaObj := fsmoObj!fsmoRoleOwner
        if not ObjExists(ownerDsaObj) then
            msgOut.ulExtendedRet := EXOP_ERR_UNKNOWN_CALLER
            return
        endif

        Call IDL_DRSGetNCChanges as a client to the server identified by
        ownerDsaObj to perform a EXOP_FSMO_REQ_ROLE extended
        operation; see the client request generation and response
        processing
        sections
        if fsmoObj!fsmoRoleOwner ≠ DSAObj() then
            /* Transfer failed. */
            msgOut.ulExtendedRet := EXOP_ERR_COULDNT_CONTACT
            return
        endif
    endif

```

```

endif
else if msgIn.ulExtendedOp = EXOP_FSMO_REQ_RID_ALLOC then
/* Allocate a block of RIDs for the client DC. */
if fsmoObj ≠ DefaultNC()!rIDManagerReference then
msgOut.ulExtendedRet := EXOP_ERR_MISMATCH
return
else if fsmoObj!fSMORoleOwner ≠ DSAObj() then
msgOut.ulExtendedRet := EXOP_ERR_NOT_OWNER
return
endif

/* Locate or create the RID Set object for the client DC. */
clientComputerObj := clientDsaObj!serverReference
if clientComputerObj!rIDSetReference = null then
clientRidSetObj := An implementation defined DSName in the
default NC such that not ObjExists(clientRidSetObj)
Create object with DSName clientRidSetObject such that
rIDSet in clientRidSetObject!objectClass
/* Windows Behavior: Windows sets clientRidSetObj to be a child
* of clientComputerObj. */
clientComputerObj!rIDSetReference := clientRidSetObj
else
clientRidSetObj := clientComputerObj!rIDSetReference
endif

/* Get the current RID allocation for the client DC. */
ridAllocLoHi := clientRidSetObj!rIDAllocationPool
ridAvailHi := most significant 32 bits of ridAvailLoHi
ridReqHi := most significant 32 bits of msgIn.liFsmoInfo
if ridAllocLoHi = 0 or ridAvailHi = 0 or ridReqHi ≥ ridAvailHi then
/* The client DC has indeed exhausted its current allocation,
* according to our records. */

/* Get the range of RIDs that have not yet been allocated to any
* DC. */
ridAvailLoHi := fsmoObj!rIDAvailablePool
ridAvailLo := least significant 32 bits of ridAvailLoHi
ridAvailHi := most significant 32 bits of ridAvailLoHi

/* Select a subset of the unallocated RIDs and allocate them to
* the client. */
Assign a value to ridAllocHi according to any implementation-
defined policy such that ridAvailLo < ridAllocHi < ridAvailHi.
/* Windows Behavior: By default, Windows sets ridAllocHi to
* ridAvailLo + 500. */
ridAllocLoHi := ridAvailLo as least significant 32 bits and
ridAllocHi as most significant 32 bits
ridAvailLo := ridAllocHi + 1
ridAvailLoHi := ridAvailLo as least significant 32 bits and
ridAvailHi as most significant 32 bits
fsmoObj!rIDAvailablePool := ridAvailLoHi
clientRidSetObj!rIDAllocationPool := ridAllocLoHi
msgOut.liFsmoInfo := ridAllocLoHi
endif

scope := GetRoleScope(fsmoObj) +
{clientComputerObj, clientRidSetObj}
else if EXOP_REPL_SECRETS in msgIn.ulExtendedOp and

```

```

        AmILHServer() then
/* Request replication of a single object with secret.
* Secret replication is allowed only if these three conditions
* hold:
*   1. Caller is an RODC. An RODC will always be a member of
*      "Enterprise read-only domain controllers" (SID S-1-5-22)
*      [MS-ADTS] section 7.1.1.2.6.9.
*   2. The object is configured to reveal secrets.
*   3. Outbound secret replication is not disabled.
*/
        serverObj := clientDsaObj!parent
        rodcObj := serverObj!serverReference
        if CheckGroupMembership(
            GetCallerAuthorizationInfo(), SidFromStringSid("S-1-5-22"))
            and RevealSecretsForUserAllowed(rodcObj, fsmoObj)
            and (not NTDSDSA_OPT_DISABLE_OUTBOUND_REPL_SECRET
                in DSAObj()!options
                or DRS_SYNC_FORCED in msgIn.ulFlags) then
            scope := {fsmoObj}
        else
            scope := {}
        endif
    else if EXOP_REPL_OBJ in msgIn.ulExtendedOp
        if AmILHServer() = true and
            NTDSDSA_OPT_DISABLE_OUTBOUND_REPL_OBJ in DSAObj()!options and
            not DRS_SYNC_FORCED in msgIn.ulFlags then
            /* replication of single object is disabled */
            return ERROR_DS_DRA_SOURCE_DISABLED
        endif
        scope := {fsmoObj}
    else
        /* Unrecognized request. */
        msgOut.ulExtendedRet := EXOP_ERR_UNKNOWN_OP
        return
    endif

    if scope ≠ {} then
        /* Add updates in scope to the response. */
        GetChangesInScope(scope, msgIn.pUpToDateVecDest, null, null,
            changedObjs, changedLinks)
        foreach o in changedObjs
            AddObjToResponse(
                hDrs, o, ncRoot, msgIn.ulFlags, msgIn.ulExtendedOp, msgOut)
        endfor
        foreach v in changedLinks
            AddLinkToResponse(v, msgIn, msgOut)
        endfor
    endif

    msgOut.ulExtendedRet := EXOP_ERR_SUCCESS
    return

```

4.1.10.5.12 RevealSecretsPolicy

```

typedef enum
{
    RevealSecretsDeny = 0,

```



```

    RevealSecretsAllow = 1,
    RevealSecretsNoPolicy = 2
} RevealSecretsPolicy;

```

4.1.10.5.13 GetRevealSecretsPolicyForUser

```

procedure GetRevealSecretsPolicyForUser(
    rodObj: DSName, userObj: DSName): RevealSecretsPolicy

```

Informative summary of behavior: The GetRevealSecretsPolicyForUser method returns the policy that indicates whether the server that holds the secrets of the user object userObj is allowed to send those secrets to the RODC identified by the RODC object rodObj. If the policy explicitly prohibits the RODC from receiving the secrets, RevealSecretsDenied is returned. If the policy explicitly allows the RODC to receive the secrets, RevealSecretsAllow is returned. In all other cases, RevealSecretsNoPolicy is returned.

```

neverRevealObj: DSName
revealObj: DSName
/* An RODC can always cache secrets of its own account
*/
if rodObj = userObj /* see section 5 DSNAME for DSName equality */
then
    return RevealSecretsAllow
endif
/* An RODC can always cache secrets of its own
* secondary Kerberos TGT account but not other
* secondary Kerberos TGT accounts.
* See [MS-KILE]
*/
if rodObj!msDS-KrbTgtLink = userObj then
    return RevealSecretsAllow
endif
krbtgts = select o from children DefaultNC() where
    o!msDS-KrbTgtLink!=NULL
foreach krbtgt in krtgts do
    if userObj = krbtgt!msDS-KrbTgtLink then
        return RevealSecretsDeny
    endif
endfor

/* Never reveal secrets of inter-domain
* trust accounts
*/
if userObj!UserAccountControl ∩ {ADS_UF_INTERDOMAIN_TRUST_ACCOUNT}
    ≠ {} then
    return RevealSecretsDeny
endif

/* Never reveal secrets of users reachable from
* rodObj!msDS-NeverRevealGroup
*/
foreach neverRevealObj in rodObj!msDS-NeverRevealGroup
    if IsUserIncluded(
        userObj!objectSid, neverRevealObj!objectSid) then

```

```

        return RevealSecretsDeny
    endif
endfor

/* Only reveal secrets of users reachable from
 * rodObj!msDS-RevealOnDemandGroup
 */
foreach revealObj in rodObj!msDS-RevealOnDemandGroup
    if IsUserIncluded(
        userObj!objectSid, revealObj!objectSid) then
        return RevealSecretsAllow
    endif
endfor
return RevealSecretsNoPolicy

```

4.1.10.5.14 RevealSecretsForUserAllowed

```

procedure RevealSecretsForUserAllowed(
    rodObj: DSName, userObj: DSName): boolean

```

Informative summary of behavior: The RevealSecretsForUserAllowed method returns true if a server that holds secrets of the [user](#) object userObj is allowed to send those secrets to the RODC identified by RODC object rodObj.

```

policy: RevealSecretsPolicy
allowed: boolean
policy = GetRevealSecretsPolicyForUser(rodObj, userObj)
if (policy = RevealSecretsDeny) then
    allowed := false
else if (policy = RevealSecretsAllow) then
    allowed := true
else
    allowed := false
endif

return allowed

```

4.1.10.5.15 GetRoleScope

```

procedure GetRoleScope(fsmoObj: DSName): set of DSName

```

Informative summary of behavior: The GetRoleScope method returns the set of objects in the FSMO role identified by the FSMO role object fsmoObj.

```

scope: set of DSName
partitionsFsmoObj: DSName
schemaFsmoObj: DSName
ridFsmoObj: DSName
pdcFsmoObj: DSName
c: DSName
r: set of DSName

```

```

partitionsFsmoObj := select one o from children ConfigNC()
    where o!name = "Partitions"
schemaFsmoObj := SchemaNC()
infrastructureFsmoObj := select one o from children DefaultNC()
    where o!name = "Infrastructure"
ridFsmoObj := DefaultNC()!rIDManagerReference

/* Scope always includes fsmoObj. For the PDC Emulation Role, scope
 * includes only fsmoObj. */
scope := {fsmoObj}

if fsmoObj = partitionsFsmoObj then
    /* Partition Naming Master Role: Add to scope the children of the
     * Partitions container. */
    r := select all o from children partitionsFsmoObj where true
    scope := scope + r
else if fsmoObj = schemaFsmoObj then
    /* Schema Master Role: Set scope to all objects in the Schema
     * NC. */
    scope := select all o from subtree SchemaNC() where true
else if fsmoObj = infrastructureFsmoObj then
    /* Infrastructure Master Role: Add to scope all objects in the
     * subtree rooted at CN=DomainUpdates,CN=System,DefaultNC(). */
    c := select one o from children DefaultNC() where o!name = "System"
    c := select one o from children c where o!name = "DomainUpdates"
    r := select all o from subtree c where true
    scope := scope + r
else if fsmoObj = ridFsmoObj then
    /* RID Allocation Master Role: Add to scope all children of
     * CN=Infrastructure,DefaultNC() that are of class
     * infrastructureUpdate and have a value for the proxiedObjectName
     * attribute. */
    r := select all o from children-ts-included infrastructureFsmoObj
        where infrastructureUpdate in o!objectClass and
        not o!proxiedObjectName = null
    scope := scope + r
endif

return scope

```

4.1.10.5.16 SortResponseLinks

```

procedure SortResponseLinks(var msgOut: DRS_MSG_GETCHGREPLY_V6)

```

The SortResponseLinks method sorts the contents of msgOut.rgValues in ascending order according to the comparison method CompareLinks():

```

procedure CompareLinks(REPLVALINF val1, REPLVALINF val2): integer
begin
    c: integer
    dsname1: DSName
    dsname2: DSName

    /* Returns 1
     * if val1 > val2, 0 if val1 = val2, or -1 if val1 < val2. */

```

```

/* Compare by ascending host object objectGUID. */
c := result of ANSI C function memcmp()
    applied to val1.pObject^.Guid and val2.pObject^.Guid,
    in little-endian byte order

/* Then by ascending attribute ID. */
if c = 0 then
    if val1.attrTyp < val2.attrTyp then
        c := -1
    else if val1.attrTyp > val2.attrType then
        c := 1
    endif
endif

/* Then by ascending "is present". */
if c = 0 then
    if not val1.fIsPresent and val2.fIsPresent then
        c := -1
    else if val1.fIsPresent and not val2.fIsPresent then
        c := 1
    endif
endif

/* Then by ascending referenced object objectGUID. */
if c = 0 then
    dsname1 := Value of val1.AVal.pVal^
    dsname2 := Value of val2.AVal.pVal^

    c := result of ANSI C function memcmp() applied to dsname1.Guid
        and dsname2.Guid, in little-endian byte order
endif

return c
end

```

4.1.10.5.17 TransformOutput

```

procedure TransformOutput(
    msgOut: DRS_MSG_GETCHGREPLY_V6,
    flags: DRS_OPTIONS,
    pdwOutVersion: ADDRESS OF DWORD,
    pmsgOut: ADDRESS OF DRS_MSG_GETCHGREPLY): ULONG

```

Informative summary of behavior: The TransformOutput method transforms the V6 reply (a superset of all supported reply messages) into the reply version supported by the client, optionally compressing it by using the algorithms defined in section [4.1.10.6.14](#).

```

pickled: sequence of BYTE
compressed: sequence of BYTE
allowedAlgs: set of DRS_COMP_ALG_TYPE
compressAlg: DRS_COMP_ALG_TYPE
compress: boolean
/* The SMTP transport [MS-SRPL] performs its own compression. */
compress := DRS_USE_COMPRESSION in flags

```

```

        and not DRS_MAIL_REP in flags
if pdwOutVersion^ = 6 then
    /* Return V6 (uncompressed) or V7 (compressed V6). */
    if compress then
        /* Return V7 (compressed V6). */
        if not DRS_EXT_GETCHGREPLY_V7 in ext.dwFlags then
            return ERROR_REVISION_MISMATCH
        endif
        /* Serialize msgOut into a byte stream. */
        pickled := Pickling of msgOut, as specified by
            [C311] Part 2, "IDL/NDR Pickles" and
            [MS-RPCE] sections 2.2.6 and 2.2.7
        /* Select a compression algorithm. */
        allowedAlgs := {DRS_COMP_ALG_NONE, DRS_COMP_ALG_MSZIP}
        if DRS_EXT_W2K3_DEFLATE in ext.dwFlags then
            allowedAlgs := allowedAlgs + {DRS_COMP_ALG_WIN2K3}
        endif
        compressAlg := One of allowedAlgs, selected by an
            implementation-defined policy.
        /* Compress the serialized msgOut. */
        compressed :=
            Compression of pickled using compressAlg, as specified
            by section 4.1.10.6.14.
        pdwOutVersion^ := 7
        pmsgOut^.V7.dwCompressedVersion := 6
        pmsgOut^.V7.CompressionAlg := compressAlg
        pmsgOut^.V7.CompressedAny.cbUncompressedSize := pickled.length
        pmsgOut^.V7.CompressedAny.cbCompressedSize := compressed.length
        pmsgOut^.V7.CompressedAny.pbCompressedData := bytes in compressed
    else
        /* Return V6 (uncompressed). */
        pdwOutVersion^ := 6
        pmsgOut^.V6 := msgOut
    endif
else
    /* Return V1 (uncompressed) or V2 (compressed V1). */
    /* First, convert to V1. */
    pdwOutVersion^ := 1
    pmsgOut^.V1 := msgOut
    pmsgOut^.V1.pUpToDateVecSrc := Convert msgOut.pUpToDateVecSrc (of
        type UPTODATE_VECTOR_V1_EXT) to UPTODATE_VECTOR_V2_EXT by
        creating a new UPTODATE_VECTOR_V1_EXT with a V1 cursor for each
        V2 cursor, sans the timeLastSyncSuccess field.
    /* V1 has the NC size in the ulExtendedRet field. */
    if msgOut.cNumNcSizeObjects > 0 then
        pmsgOut^.V1.ulExtendedRet := msgOut.cNumNcSizeObjects
    endif
    if compress then
        /* Serialize msgOut into a byte stream. */
        pickled := Pickling of pmsgOut^.V1, as specified by
            [C311] Part 2, "IDL/NDR Pickles" and
            [MS-RPCE] sections 2.2.6 and 2.2.7
        /* Select a compression algorithm. */
        allowedAlgs := {DRS_COMP_ALG_NONE, DRS_COMP_ALG_MSZIP}
        compressAlg := One of allowedAlgs, selected by an
            implementation-defined policy.
        /* Compress the serialized msgOut. */
        compressed :=

```

```

        Compression of pickled using compressAlg, as specified
        by section 4.1.10.6.14.
    pdwOutVersion^ := 2
    pmsgOut^.V2.CompressedV1.cbUncompressedSize := pickled.length
    pmsgOut^.V2.CompressedV1.cbCompressedSize := compressed.length
    pmsgOut^.V2.CompressedV1.pbCompressedData := bytes in compressed
endif
endif
return 0

```

4.1.10.6 Client Behavior When Receiving the IDL_DRSGetNCChanges Response

The client processes an IDL_DRSGetNCChanges response in relation to the current state of its NC replica as detailed in ProcessGetNCChangesReply below. This processing, though sometimes complex, is critical to ensuring that each NC replica arrives at the same abstract state.

4.1.10.6.1 ProcessGetNCChangesReply

```

procedure ProcessGetNCChangesReply(
    rf: RepsFrom,
    msgInV8: DRS_MSG_GETCHGREQ_V8,
    dwOutVersion: ULONG,
    msgOut: DRS_MSG_GETCHGREPLY)

```

Informative summary of behavior: The ProcessGetNCChangesReply procedure is invoked when an [IDL_DRSGetNCChanges](#) response is received over RPC or SMTP, as specified in [\[MS-SRPL\]](#). Processing of a given response can be separated into five distinct phases: decompression, attribute value decryption, processing object updates, processing link value updates, and updating the "watermark" information.

The arguments to this procedure are as follows:

- rf: [RepsFrom](#) for the server.
- msgInV8: **IDL_DRSGetNCChanges** request message sent to the server.
- dwOutVersion: Version of response message received from the server.
- msgOut: Response message received from the server.

```

msgReplyV6: DRS_MSG_GETCHGREPLY_V6
replEntinflist: REPLENTINFLIST
continueProcessing: boolean
writableReplica: boolean
sourcePrefixTable: PrefixTable
attributesAndStamps: set of AttributeAndStamp
linkValueCount: DWORD
clientSchemaSignature: sequence of BYTE
serverSchemaSignature: sequence of BYTE
fServerSchemaMoreRecent: boolean
lastElement: DWORD

ulResult := msgReplyV6.dwDRSError

```

```

if (ulResult = 0) then
    /* Decompress and/or translate the response to a V6 response,
    * as necessary. */
    if (dwOutVersion = 0x2) or (dwOutVersion = 0x7) then
        msgReplyV6 := DecompressReplyMessage(msgOut, dwOutVersion)
    else
        msgReplyV6 := GetNCChangesV6Reply(msgOut, dwOutVersion)
    endif

    sourcePrefixTable :=
        AbstractPTFromConcretePT(msgReplyV6.PrefixTableSrc)

    /* Check whether the schema on client and server match. */
    lastElement := sourcePrefixTable.length - 1
    serverSchemaSignature :=
        copy sourcePrefixTable[lastElement].prefix.length bytes of data
        from sourcePrefixTable[lastElement].prefix.elements
    clientSchemaSignature := SchemaNC()!schemaInfo
    if clientSchemaSignature # serverSchemaSignature then
        ulResult := ERROR_DS_DRA_SCHEMA_MISMATCH
    endif
    Remove sourcePrefixTable[lastElement] from sourcePrefixTable
endif

/* Process object updates. */
replEntinfList := msgReplyV6.pObjects^
while (ulResult = 0) and (not replEntinfList = null)
    /* Decrypt any encrypted attribute values. */
    ulResult := DecryptEncryptedAttributeVals(
        replEntinfList.Entinf.AttrBlock,
        sourcePrefixTable)

    if (ulResult = 0) then
        attributesAndStamps := GetStampsForUpdate(
            replEntinfList,
            sourcePrefixTable)

        /* Process objects that are moved across an NC. */
        continueProcessing := PrepareCrossNCMove(
            replEntinfList,
            sourcePrefixTable)
    endif

    if continueProcessing and (ulResult = 0) then
        if (DRS_WRIT_REP in msgInV8.ulFlags) then
            writableReplica := true
        else
            writableReplica := false
        endif
        continueProcessing := AdjustInstanceTypeAttrVal(
            msgReplyV6.pNC^,
            writableReplica,
            replEntinfList,
            prefixTable)
    endif

    if continueProcessing and (ulResult = 0) then
        if (not ObjExists(replEntinfList.Entinf.pName^)) then
            ulResult := AddObject(
                replEntinfList,
                sourcePrefixTable,

```

```

        attributesAndStamps)
    else
        ulResult := UpdateObject(
            replEntinfList,
            sourcePrefixTable,
            attributesAndStamps)
    endif
endif
replEntinfList := replEntinfList.pNextEntInf^
endwhile

/* Process link value updates. */
linkValueCount := 0
while (ulResult = 0) and (linkValueCount < msgReplyV6.cNumValues)
    ulResult := ProcessLinkValue(
        msgReplyV6.rgValues[linkValueCount],
        msgReplyV6.pNC^,
        prefixTable)
    linkValueCount := linkValueCount + 1
endwhile

if (ulResult = ERROR_DS_DRA_MISSING_PARENT) then
    Send IDL_DRSGetNCChanges message again with the same input
    parameters specified in msgInV8 but this time with msgInV8.ulFlags
    containing DRS_GET_ANC field set. It is an error for this
    condition to occur if (DRS_GET_ANC in msgInV8.ulFlags) is true
else if (msgInV8.ulExtendedOp = 0) then
    /* Not an extended operation. Update "watermark" information. */
    UpdateRepsFrom(
        rf,
        msgReplyV6,
        dsaServer,
        ulResult)

    if (ulResult = 0) and (msgReplyV6.fMoreData = false) then
        UpdateUTDandPAS(
            msgReplyV6,
            msgInV8.partialAttrSetEx^
        )
    endif
endif
endif

return

```

4.1.10.6.2 PrepareCrossNCMove

```

procedure PrepareCrossNCMove(
    replEntinfList: REPLENTINFLIST,
    sourcePrefixTable: PrefixTable): boolean

```

Informative summary of behavior: The **PrepareCrossNCMove** procedure determines whether the object specified by the replEntinfList argument is being moved from one NC to another and, if so, the procedure performs preparatory work and/or terminates further processing of replEntinfList. The procedure returns true if further processing of replicated update in replEntinfList must be performed. Otherwise, it returns false.


```

proxiedNameAttrVal: ATTRVAL
proxiedNameValue: DNBinary
localProxiedNameValue: DNBinary
proxyEpoch: DWORD
localProxyEpoch: DWORD
proxyObject: DSName
proxyObjectNameValue: DNBinary
isProxy: boolean
objClassVal: ATTRVAL

proxiedNameAttrVal := ENTINF_GetValue(
    replEntinfList.Entinf,
    proxiedObjectName,
    sourcePrefixTable)
if (proxiedNameAttrVal = null) then
    /* Update is not related to cross NC move. Therefore, continue
       processing the replicated update. */
    return true
endif

/* replEntinfList corresponds to an object that has moved across an
   * NC /
proxiedNameValue := ValueFromATTRVAL(
    proxiedNameAttrVal, Syntax(proxiedObjectName), sourcePrefixTable)
proxyEpoch := GetProxyEpoch(proxiedNameValue)

/* Check whether the objectClass is infrastructureUpdate. */
objClassVal := ENTINF_GetValue(replEntinfList.Entinf, objectClass,
    sourcePrefixTable)
if LocalAttidFromRemoteAttid(
    sourcePrefixTable, objClassVal.pAVal^.pVal^)
    = infrastructureUpdate then
    isProxy := true
else
    isProxy := false
endif

if not isProxy then
    /* Replicated update is not for an infrastructureUpdate object. */
    proxyObject := replEntinfList.Entinf.pName^
    if (ObjExists(proxyObject)) and
        (not proxyObject!proxiedObjectName = null) then
        localProxyEpoch := GetProxyEpoch(proxyObject!proxiedObjectName)
    else
        localProxyEpoch := 0
    endif
    if (localProxyEpoch > proxyEpoch) then
        /* Local EPOCH value is higher. Don't continue processing the
           * replicated update. */
        return false
    else if (localProxyEpoch < proxyEpoch) and
        (ObjExists(proxyObject)) then
        Expunge(proxyObject)
    endif
else
    proxyObjectNameValue :=
        ValueFromATTRVAL(proxiedNameAttrVal.pVal,
            Syntax(proxiedObjectName),

```

```

        sourcePrefixTable)
proxyObject := proxyObjectNameValue.dn
if (ObjExists(proxyObject)) then
    localProxiedNameValue = proxyObject!proxiedObjectName
    if (localProxiedNameValue = null) then
        localProxyEpoch := 0
    else
        localProxyEpoch := GetProxyEpoch(localProxiedNameValue)
    endif
    if (localProxyEpoch < proxyEpoch) then
        Expunge(proxyObject)
    endif
endif
endif
return true /* Continue processing the replicated update. */

```

4.1.10.6.3 AdjustInstanceTypeAttrVal

```

procedure AdjustInstanceTypeAttrVal(
    ncReplicated: DSName,
    writableReplica: DSName,
    var replEntinfList: REPLENTINFLIST,
    prefixTable: PrefixTable) : boolean

```

Informative summary of behavior: This AdjustInstanceTypeAttrVal procedure adjusts the attribute value of [instanceType](#) attribute in replEntinfList parameter to an appropriate value that suits the NC replica on the client. The procedure returns true if further processing of replicated update in replEntinfList must be performed. Otherwise, it returns false.

```

instanceTypeAttrVal: ATTRVAL
instanceTypeAdjustedAttrVal: ATTRVAL
instanceTypeVal: ULONG
instanceTypeAdjustedVal: ULONG
ncSubRef: DSName

instanceTypeAttrVal := ENTINF_GetValue(replEntinfList.Entinf,
    instanceType, prefixTable)
if (instanceTypeAttrVal = null) then
    /* If instanceType attribute is not present in Entinf
    * then there is no value to adjust. */
    return true
endif

instanceTypeVal := ValueFromATTRVAL(
    instanceTypeAttrVal, Syntax(instanceType), prefixTable)

if (IT_NC_HEAD in instanceTypeVal) and
    (not ncReplicated = replEntinfList.Entinf.pName^)
/* If IT_NC_HEAD is set in instanceTypeVal and
* replEntinfList.Entinf.pName is not the DSName of the root of the
* NC replica that the client is replicating. Take this opportunity
* to ensure that ncReplicated!subRefs has an entry for the
* subordinate NC.
*/
    ncSubRef := replEntinfList.Entinf.pName^

```

```

    if (not ncSubRef in ncReplicated!subRefs) then
        ncReplicated!subRefs := ncReplicated!subRefs + {ncSubRef}
    endif
    return false /* Skip processing this entry. */
endif

if (not writableReplica) and
    (IT_WRITE in instanceTypeVal) then
    /* If the client NC replica is a partial replica then remove the
     * IT_WRITE flag from the instanceTypeVal to mark the object as
     * read-only.
     */
    instanceTypeAdjustedVal := instanceTypeVal - {IT_WRITE}
else
    instanceTypeAdjustedVal := instanceTypeVal
endif

/* Set or reset instance type bits other than IT_WRITE and
 * IT_NC_HEAD. */
instanceTypeAdjustedVal :=
    SetResetInstanceTypeBits(instanceTypeAdjustedVal)

instanceTypeAdjustedAttrVal := ATTRVALFromValue(
    instanceTypeAdjusted, Syntax(instanceType), prefixTable)

ENTINF_SetValue(replEntinfList.Entinf, instanceType,
    instanceTypeAdjustedAttrVal, prefixTable)

return true

```

4.1.10.6.4 SetResetInstanceTypeBits

```

procedure SetResetInstanceTypeBits(y: DWORD): DWORD

```

The SetResetInstanceTypeBits procedure is an implementation-specific function that MAY [<19>](#) set or reset bits in y other than IT_WRITE and IT_NC_HEAD. It returns the updated value.

4.1.10.6.5 AddObject

```

procedure AddObject(
    replEntinfList: REPLENTINFLIST,
    sourcePrefixTable: PrefixTable,
    nc: DSName,
    attributesAndStamps: set of AttributeAndStamp): DWORD

```

Informative summary of behavior: The AddObject procedure performs a replicated update by adding an object to the NC replica. This procedure has the following input parameters:

- replEntinfList: The replicated update to be applied.
- sourcePrefixTable: The prefix table from the server to translate attribute IDs.
- nc: The root of the NC replica that is replicated.

- attributesAndStamps: The [AttributeAndStamp](#) set that corresponds to the replicated update.

The procedure returns a Windows error code if it encounters an error while adding the object.

```

newObjectDN: DN
parentObject: DSName
newObject: DSName

parentObject := select one o from all-ts-included where
  (o!objectGUID = replEntinfList.pParentGuid^)
if (parentObject = null) then
  /* The client will stop processing the reply message. It will
   * resend the IDL DRSGetNCChanges request with DRS_GET_ANC set in
   * ulFlags. It is an error for this condition to occur if the
   * request already included DRS_GET_ANC in ulFlags.
   */
  return ERROR_DS_DRA_MISSING_PARENT
endif

if (not GetObjectNC(parentObject) = nc) then
  /* If parentObject exists in an NC replica other than that
   * being replicated, the client stops processing the response.
   * This condition indicates that parentObject has moved from one
   * NC replica to another and that update has not yet been applied
   * to the client NC replica containing parentObject.
   * This will be rectified when the client replicates the NC
   * replica containing parentObject.
   */
  return ERROR_DS_DRA_OBJ_NC_MISMATCH
endif

/* Find an appropriate parent object for the object. If the parent
 * object is deleted and if the new object is not a deleted object
 * then FindBestParentObject will return DSName of "Lost and Found
 * container". Otherwise, the parent object will remain the same. */
parentObject := FindBestParentObject(parentObject, replEntinfList,
  sourcePrefixTable, nc, attributesAndStamps)

/* Check whether there is a name conflict (see [MS-ADTS] section
 * 3.1.1) and resolve it before adding the object. */
newObjectDN := ResolveNameConflict(replEntinfList, newParentObject,
  sourcePrefixTable, attributesAndStamps)

/* Set the new DN in the ENTINF. */
Copy value of newObjectDN to replEntinfList.Entinf.pName^.StringName
and update the value in replEntinfList.Entinf.pName^.structLen and
replEntinfList.Entinf.pName^.NameLen accordingly.

ulResult := PerformAddOperation(replEntinfList.Entinf, newObject,
  sourcePrefixTable)

/* Update attribute stamps. */
if (ulResult = 0) then
  for each e in attributesAndStamps do
    SetAttrStamp(newObject, e.attribute, e.stamp)
  endfor
endif

```

```
return ulResult
```

4.1.10.6.6 UpdateObject

```
procedure UpdateObject(  
    replEntinfList: REPLENTINFLIST,  
    sourcePrefixTable: PrefixTable,  
    nc: DSName,  
    attributesAndStamps: set of AttributeAndStamp): DWORD
```

Informative summary of behavior: The UpdateObject procedure performs a replicated update by applying changes on an existing object in an NC replica. This procedure has the following input parameters:

- replEntinfList: The replicated update to be applied.
- sourcePrefixTable: The prefix table from the server to translate attribute IDs.
- nc: The root of NC replica that is replicated.
- attributesAndStamps: The [AttributeAndStamp](#) set that corresponds to the replicated update.

The method returns a Windows error code if encounters an error while updating the object.

```
updateObject: DSName  
newParentObject: DSName  
stampRemote: AttributeStamp  
stampLocal: AttributeStamp  
renameObjectDN: DN  
newRDN: RDN  
attribute: ATTRTYP  
nameAttrAndStamp: AttributeAndStamp  
attrAndStamp: AttributeAndStamp  
isDeletedAttrAndStamp: AttributeAndStamp  
isDeletedAttrVal: ATTRVAL  
delete: boolean  
  
updateObject := replEntinfList.Entinf.pName^  
  
/* Determine whether replEntinfList indicates a delete operation. */  
isDeletedAttrVal :=  
    ENTINF_GetValue(replEntinfList.Entinf,  
                    isDeleted, sourcePrefixTable)  
delete := (isDeletedAttrVal ≠ null) and  
    (ValueFromATTRVAL(  
        isDeletedAttrVal, Syntax(isDeleted), sourcePrefixTable) = true)  
if delete and not AmIRODC() and IsProtectedObject(updateObject) then  
    /* This is a deletion of a protected object -- an object that  
    * *this* DC does not allow to be deleted. Undelete it as an  
    * originating update. */  
    UndeleteObject(updateObject, attributesAndStamps)  
    return 0  
endif  
  
/* Determine whether replEntinfList indicates a rename operation. */
```

```

nameAttrAndStamp := select one e from attributesAndStamps where
    (e.attribute = name)
if (nameAttrAndStamp = null) then
    stampRemote := null
else
    stampRemote := nameAttrAndStamp.stamp
endif

stampLocal := AttrStamp(updateObject, name)

if (not stampRemote = null) and
    (AttributeStampCompare(stampRemote, stampLocal) > 0) then
    /* This indicates that replEntinfList provides a more recent
    * DN for updateObject. It is important to note here that a change
    * in the name attribute is interpreted as a potential change in
    * the full DN, not just the RDN. */
    newParentObject = select one o from subtree-ts-included nc where
        (o.objectGUID = replEntinfList.pParentGuid^)
    if newParentObject = null then
        /* The client will stop processing the reply message.
        * It will resend the IDL_DRSGetNCChanges request with
        * DRS_GET_ANC set in ulFlags. It is an error for this condition
        * to occur if the request already included DRS_GET_ANC in
        * ulFlags.
        */
        return ERROR_DS_DRA_MISSING_PARENT
    endif

    /* Find an appropriate new parent. If the parent object is deleted
    * and if the updated object is not a deleted object then
    * FindBestParentObject will return DSName of "Lost and Found
    * container". Otherwise, the new parent object will remain the
    * same
    */
    newParentObject := FindBestParentObject(newParentObject,
        replEntinfList, sourcePrefixTable, nc, attributesAndStamps)

    /* Check whether there is a name conflict (see [MS-ADTS]
    * section 3.1.1) and, if so, resolve it before renaming the
    * object.
    */
    renameObjectDN := ResolveNameConflict(
        replEntinfList.Entinf, newParentObject, attributesAndStamps)
    newRDN := leftmost RDN of renameObjectDN
    ulResult := PerformModifyDNOperation(
        updateObject!distinguishedName, newParentObject, newRDN)
    if (not ulResult = 0)
        return ulResult
    endif
endif

/* Perform modify operation. */

/* Compare local and remote attribute stamps and update object
* attribute only if the changes are more recent than what the
* client has seen. */
for i := 0 to (replEntinfList.Entinf.AttrBlock.attrCount-1)
    attribute := LocalAttidFromRemoteAttid(

```

```

        sourcePrefixTable,
        replEntinfList.Entinf.AttrBlock.pAttr[i].attrTyp);
attrAndStamp := select one e from attributeAndStamps where
    (e.attribute = attribute)
stampRemote := attrAndStamp.stamp
stampLocal := AttrStamp(updateObject, attribute)
if (not stampLocal = null) and
    (AttributeStampCompare(stampRemote, stampLocal) <= 0) then
    /* This indicates the attribute on the object in the client is
    * more up to date. Do not apply the replicated update
    * corresponding to that attribute.
    */
    ENTINF_SetValue(replEntinfList.Entinf, attribute, null,
        sourcePrefixTable)
    attributesAndStamps := attributesAndStamps - {attrAndStamp}
endif
endfor

ulResult := PerformModifyOperation(replEntinfList.Entinf,
    updateObject,
    sourcePrefixTable)

if (ulResult = 0) and delete then
    /* Delete the object. */
    ulResult := RemoveObj(updateObject)
    if (ulResult = 0) then
        attrAndStamp := select one e from attributesAndStamps where
            (e.attribute = isDeleted)
        endif
    endif
endif

/* Update attribute stamps on the object to those corresponding to
* the replicated updates. */
if (ulResult = 0) then
    for each e in attributesAndStamps do
        SetAttrStamp(newObject, e.attribute, e.stamp)
    endfor
endif

return ulResult

```

4.1.10.6.7 FindBestParentObject

```

procedure FindBestParentObject(
    parentObject: DSName,
    replEntinfList: REPLENTINFLIST,
    sourcePrefixTable: PrefixTable,
    nc: DSName,
    var attributesAndStamps: set of AttributeAndStamp): DSName

```

Informative summary of behavior: Given a desired parent object, the FindBestParentObject procedure validates whether the desired parent object is deleted. If the object that is being updated is not a deleted object and the desired parent object is deleted, this procedure returns the [DSName](#) of the lost and found container. Following are the input parameters for this procedure:

- `parentObject`: The [DSName](#) type of the desired parent object.
- `replEntinfList`: The replicated update that should be applied.
- `sourcePrefixTable`: The prefix table from the server.
- `nc`: The [DSName](#) type of the root of the NC replica.
- `attributesAndStamps`: The [AttributeAndStamp](#) set that corresponds to the replicated update (can be modified by this procedure).

```

isDeletedAttr: ATTRVAL
isDeletedValue: boolean
attrAndStamp: AttributeAndStamp
isDeletedAttr := ENTINF_GetValue(
    replEntinfList.Entinf,
    isDeleted,
    sourcePrefixTable)
if (isDeletedAttr = null) then
    isDeletedValue := false
else
    isDeletedValue := ValueFromATTRVal(
        isDeletedAttr, Syntax(isDeleted), sourcePrefixTable)
endif

if isDeletedValue = false and parentObject!isDeleted = true then
    /* This indicates that an object was moved/created under
    * parentObject in one NC replica while parentObject was deleted
    * in another NC replica. In this case move/add an object under
    * the "lost and found" container.
    */

    /* Remove attribute stamp for name so that the update is seen
    * as an originating update. */
    attrAndStamp := select one from attributesAndStamps where
        (e.attribute = name)
    attributesAndStamps := attributesAndStamps - {attrAndStamp}
    return GetWellKnownObject(nc, GUID_LOSTANDFOUND_CONTAINER_W)
endif
return parentObject

```

4.1.10.6.8 ResolveNameConflict

```

procedure ResolveNameConflict(
    replEntinfList: REPLENTINFList,
    parentObject: DSName,
    var attributesAndStamps: set of AttributeAndStamp): DN

```

Informative summary of behavior: The `ResolveNameConflict` procedure checks whether there is a name conflict (see [\[MS-ADTS\]](#) section 3.1.1) while applying a replicated update. If there is a name conflict, the procedure changes the desired DN of the object for which the replicated update is applied, or changes the DN of the existing object so that there is no name conflict. Following are the input parameters for this procedure:

`replEntinfList`: Update to be applied.

parentObject: The [DSName](#) type of the parent object.

attributesAndStamps: The [AttributeAndStamp](#) set that corresponds to the replicated update (can be modified by this procedure).

```
objectRDN: RDN
objectDN: DN
rdnValue: unicodestring
duplicateObject: DSName
nameAttrStamp: AttributeAndStamp
guidUpdateObj: GUID
stampExistingObj: AttributeStamp
stampUpdateObj: AttributeStamp

objectRDN := leftmost RDN of replEntinfList.Entinf.pName^.StringName
rdnValue := AttributeValue portion of objectRDN (see [RFC2253])
objectDN := objectRDN followed by RDNs of
    parentObject!distinguishedName
duplicateObject := select one d from children parentObject where
    (d!name = rdnValue)
if (not duplicateObject = null) and
    (not duplicateObject!objectGUID =
        replEntinfList.Entinf.pName^.Guid^) then
/* There already exists a child object (duplicateObject) of
 * parentObject whose name attribute value will be same as the name
 * attribute value of the object being renamed/added. */
guidUpdateObj := replEntinfList.Entinf.pName^.Guid^
nameAttrStamp := select v from attributesAndStamps where
    (v.attribute = name)
stampUpdateObj := nameAttrStamp.stamp
stampExistingObj := AttrStamp(duplicateObject, name)
if (stampExistingObj.timeChanged > stampUpdateObj.timeChanged)
    or ((stampExistingObj.timeChanged =
        stampUpdateObj.timeChanged)
        and (existingObject!objectGUID > guidUpdateObj)) then
/* Rename the replicated object. */
newDN = MakeConflictDN(objectDN, guidUpdateObj)
/* Remove attribute stamp for name so that the update is seen
 * as an originating update. */
attributesAndStamps := attributesAndStamps - {nameAttrStamp}
return newDN
else
/* Rename the existing object. */
newDN = MakeConflictDN(
    existingObject!distinguishedName,
    existingObject!objectGUID)
newRDN = The left most RDN of newDN
PerformModifyDNOperation(existingObject!distinguishedName,
    null, newRDN)
return objectDN
endif
else
return objectDN /* No conflict case */
endif
```

4.1.10.6.9 MakeConflictDN

```
procedure MakeConflictDN(oldDN: DN, guid: GUID): DN
```

The MakeConflictDN procedure is used during name conflict resolution. For more information, see section [4.1.10.6.8](#).

A conflict name for [DN](#) oldDN and [GUID](#) guid is the [DN](#) newDN, such that newDN is the same as oldDN with the exception of the AttributeValue portion (as specified in [\[RFC2253\]](#)) of the first [RDN](#). This portion is the concatenation of:

- The AttributeValue portion of the first [RDN](#) of oldDN.
- The Unicode character 0x000A.
- The Unicode string "CNF:".
- The dashed string representation of guid.

For example, given oldDN = "CN=Engineering,DC=Fabrikam,DC=com" and guid = a746b716-0ac0-11d2-b376-0000f87a46c8, newDN is "CN=Engineering#CNF:a746b716-0ac0-11d2-b376-0000f87a46c8,DC=Fabrikam,DC=com", where the # represents the Unicode character 0x000A.

The procedure returns newDN.

4.1.10.6.10 ProcessLinkValue

```
procedure ProcessLinkValue(  
    replValinf: REPLVALINF,  
    sourcePrefixTable: PrefixTable,  
    nc: DSName): DWORD
```

Informative summary of behavior: The ProcessLinkValue procedure applies the replicated update of a link value. Following are the input parameters for this procedure.

- replValinf: Link value replicated update.
- sourcePrefixTable: Prefix table from the server.
- nc: The [DSName](#) type of the root of NC replica where the replicated update is applied.

```
updateObject: DSName  
isDeleted: boolean  
attribute: ATTRTYP  
attributeValue: attribute value  
attributeValues: set of attribute value  
updateObject := replValinf.pObject^  
if (not ObjExists(updateObject)) then  
    /* The client will stop processing the reply message. It will  
    * resend the IDL_DRSGetNCChanges request with DRS_GET_ANC set  
    * in ulFlags. It is an error for this condition to occur if the  
    * request already included DRS_GET_ANC in ulFlags. */  
    return ERROR_DS_DRA_MISSING_PARENT  
endif  
isDeleted := updateObject.isDeleted
```

```

if (not isDeleted = null) and (isDeleted = true)
    /* Object is deleted. Replicated update is not applied on a deleted
    * object. */
    return 0
endif
attribute := replValinf.attrTyp
attributeValues := GetAttrVals(updateObject, attribute, true)
attributeValue := select one k from attributeValues where
    (k = ValueFromATTRVAL(
        sourcePrefixTable, Syntax(attribute), replValInf.pAval))
if (attributeValue = null) then
    localValueStamp := null
else
    /* If attributeValue was last updated when the forest functional
    * level was DS_BEHAVIOR_WIN2000, no LinkValueStamp is
    * associated with attributeValue. In that case the procedure
    * LinkStamp() returns null.
    */
    localValueStamp :=
        LinkStamp(updateObject, attribute, attributeValue)
endif
remoteValueStamp := AbstractLinkValStampFromConcreteLinkValStamp(
    replValinf.Metadata)
if (localValueStamp = null) or
    (LinkValueStampCompare(localValueStamp, remoteValueStamp) < 0)
    then
    /* The replicated update is more up to date. Apply that change and
    * modify the stamp. */
    if (not attributeValue = null) then
        /* Remove the old attribute value. */
        RemoveAttrVal(updateObject, attribute, attributeValue)
    endif
    newAttributeValue = ValueFromATTRVAL(
        sourcePrefixTable, Syntax(attribute), replValInf.pAval)
    SetAttrVal(updateObject, attribute, newAttributeValue)
    /* If the abstract variable timeDeleted associated with the
    * attribute value has a non-zero value, it indicates that the
    * value has been deleted from the NC replica. */
    if (replValInf.fIsPresent = false) then
        remoteValueStamp.timeDeleted := current time on the client
    else
        remoteValueStamp.timeDeleted := 0
    endif
    SetLinkStamp(updateObject, attribute, newAttributeValue,
        remoteValueStamp)
endif
return 0

```

4.1.10.6.11 UpdateRepsFrom

```

procedure UpdateRepsFrom(
    rf: RepsFrom,
    msgReplyV6: DRS_MSG_GETCHGREPLY_V6,
    dsaServer: DSName,
    ulResult: DWORD)

```

Informative summary of behavior: Using the UpdateRepsFrom procedure, the client updates the [repsFrom](#) abstract variable after it applies the response message received from the server. Following are the input parameters for this procedure.

- rf: The [RepsFrom](#) attribute for the server.
- msgReplyV6: The [IDL_DRSGetNCChanges](#) response from the server.
- dsaServer: The [DSName](#) type of nTDSDSA object of the server.
- ulResult: The Windows error code that tells whether or not the replicated updates in the response message are applied successfully.

```

rfOld: RepsFrom
currentTime: DSTIME
nc: DSName
nc := msgReplyV6.pNC^
rfOld := select one v from nc!repsFrom where
    (v.uuidDsa = dsaServer!objectGUID)
if rfOld ≠ null then
    nc!repsFrom := nc!repsFrom - {rfOld} /* remove old entry */
endif
currentTime := current time on the client
rf.timeLastAttempt := currentTime
if (ulResult = 0) then
    rf.consecutiveFailures := 0
    rf.timeLastSuccess := currentTime
    rf.resultLastAttempt := 0
    rf.uuidInvocId := msgReplyV6.uuidInvocIdSrc
    rf.usnVec := msgReplyV6.usnvecTo
    rf.resultLastAttempt := 0
else
    rf.consecutiveFailures := rf.consecutiveFailures + 1
    rf.resultLastAttempt := ulResult
endif
nc!repsFrom := nc!repsFrom + {rfNew}

```

4.1.10.6.12 UpdateUTDandPAS

```

procedure UpdateUTDandPAS(
    msgReplyV6: DRS_MSG_GETCHGREPLY_V6,
    partialAttrSetEx: PARTIAL_ATTR_VECTOR_V1_EXT,
    nc: DSName)

```

Informative summary of behavior: If the client has applied all replicated updates in the response message of [IDL_DRSGetNCChanges](#) from the server, and if the replication cycle is complete, then the client updates the [replUpToDateVector](#) and [partialAttributeSet](#) abstract attributes, as specified in the UpdateUTDandPAS procedure. This procedure has the following input parameters.

- msgReplyV6: The IDL_DRSGetNCChanges response from the server.
- partialAttrSetEx: The [PARTIAL_ATTR_VECTOR_V1_EXT](#) structure that contains attributes to be added to the [partialAttributeSet](#) abstract variable.
- nc: The [DSName](#) type of the root of the NC replica where the replicated update is applied.

```

partialAttrSetAdd: sequence of ATTRTYP
remoteCursor: UPTODATE_CURSOR_V2
localCursor: ReplUpToDateVector
newCursor: ReplUpToDateVector
nc: DSName
i: DWORD
nc := msgReplyV6.pNC^

/* Update partialAttributeSet abstract attribute. */
if (not partialAttrSetEx.cAttrs = 0) then
    partialAttrSetAdd = AbstractPASFromConcretePAS(partialAttrSetEx)
    nc!partialAttributeSet :=
        nc!partialAttributeSet + partialAttrSetAdd
endif

/* Merge replUpToDateVector abstract attribute */
for i := 0 to msgReplyV6.pUpToDateVecSrc^.cNumCursors - 1
    remoteCursor := msgReplyV6.pUpToDateVecSrc^.rgCursors[i]
    localCursor := select one v from nc!replUpToDateVector where
        (v.uuidDsa = remoteCursor.uuidDsa)
    if (localCursor = null) then
        /* An entry for the server does not exist; add it. */
        newCursor.uuidDsa := remoteCursor.uuidDsa
        newCursor.usnHighPropUpdate := remoteCursor.usnHighPropUpdate
        newCursor.timeLastSyncSuccess := remoteCursor.timeLastSyncSuccess
        nc!replUpToDateVector := nc!replUpToDateVector + {newCursor}
    else
        /* Update existing entry for the server. */
        if (localCursor.usnHighPropUpdate <
            remoteCursor.usnHighPropUpdate) then
            newCursor.usnHighPropUpdate := remoteCursor.usnHighPropUpdate
            newCursor.timeLastSyncSuccess :=
                remoteCursor.timeLastSyncSuccess
            newCursor.uuidDsa := remoteCursor.uuidDsa
            nc!replUpToDateVector :=
                nc!replUpToDateVector - {localCursor} + {newCursor}
        endif
    endif
endfor

return

```

4.1.10.6.13 DecryptValuesIfNecessary

```

procedure DecryptValuesIfNecessary(
    hDrs: DRS_HANDLE,
    prefixTable: PrefixTable,
    var attrBlock: ATTRBLOCK): DWORD

```

Informative summary of behavior: The values of several attributes are encrypted by the server and conversely must be decrypted by the client before processing object updates. The client decrypts the encrypted data by using MD5 digest (as specified in [RFC1321](#)), a CRC32 checksum (as specified in [ISO/IEC 13239](#)), and RC4 stream cipher (as specified in [RC4](#)). The DecryptValuesIfNecessary procedure specifies the process of attribute value decryption.

Following are the input parameters for this method.

- hDrs: The [DRS_HANDLE](#) type derived by sending [IDL_DRSBind](#) to the server.
- prefixTable: The prefix table used to translate attribute IDs.
- attrBlock: The [ATTRBLOCK](#) structure that is derived from the response of the [IDL_DRSGetNCChanges](#) message. If attrBlock has attribute values that need to be decrypted, then the values are decrypted in place. That is, at the end of the procedure call, the pVal field in the ATTRVAL structure refers to the decrypted attribute value.

The procedure returns a Windows error code on failure. Otherwise, it returns 0.

```
localAttid: ATTRTYP
attr: ATTR
pPayload: ADDRESS OF ENCRYPTED_PAYLOAD
salt: sequence of BYTE
sessionKey: sequence of BYTE
i: integer
j: integer
crcComputed: ULONG
crcReceived: ULONG
md5Context: MD5_CTX

/* Get session key associated with the RPC connection. */
sessionKey := session key associated with security context of hDrs,
as specified by [MS-RPCE] section 3.3.1.5.2 and [MS-KILE]
section 3.1.1.2

for j := 0 to (attrBlock.attrCount - 1)
    attr := attrBlock.pAttr[j]
    localAttid = LocalAttidFromRemoteAttid(prefixTable, attr.attrTyp)
    if IsSecretAttribute(localAttid) then
        /* Decrypt all values of this attribute. */
        for i := 0 to (attr.AttrVal.valCount - 1)
            pPayload := attr.AttrVal.pAVal[i].pVal
            salt := pPayload^.Salt
            /* Compute encryption key. */
            MD5Init(md5Context)
            MD5Update(md5Context, sessionKey, sessionKey.length)
            MD5Update(md5Context, salt, 16)
            MD5Final(md5Context)

            Decrypt (attr.AttrVal.pAVal[i].valLen - 16) bytes starting at
            the address of pPayload^.Checksum using the RC4 stream cipher
            algorithm [RC4] with encryption key md5Context.digest. At the
            end of this operation pPayload^.EncryptedData field contains
            decrypted attribute value.

            /* Calculate checksum of the clear value. */
            crcComputed :=
                CRC32 [ISO/IEC 13239] of the
                (attr.AttrVal.pAVal[i].valLen - 20)
                bytes starting at pPayload^.EncryptedData
            crcReceived := pPayload^.Checksum
            if (not crcComputed = crcReceived) then
                /* Checksums don't match. Stop processing the reply message.
                */
```

```

        return SEC_E_ALGORITHM_MISMATCH
    endif

    /* Modify ATTRVAL structure to have decrypted data. */
    attr.AttrVal.pAVal[i].valLen :=
        attr.AttrVal.pAVal[i].valLen - 20
    attr.AttrVal.pAVal[i].pVal := ADR(pPayload^.EncryptedData)
endfor
endif
endfor
return 0

```

4.1.10.6.14 DecompressReplyMessage

```

procedure DecompressReplyMessage(
    msgOut: DRS_MSG_GETCHREPLY,
    dwOutVersion: DWORD): DRS_MSG_GETCHG_REPLY_V6

```

Informative summary of behavior: Compression subdivides a data stream into sequences of bytes called chunks. The DecompressReplyMessage procedure decompresses the data stream.

The following table identifies the chunk size for each algorithm type:

Algorithm	Chunk size
COMP_ALG_NONE	not applicable
COMP_ALG_MSZIP	32768
COMP_ALG_W2K3	65536

Each chunk in the compressed byte sequence is represented by means of a [COMPRESSED_DATA](#) structure.

```

pInBuffer: sequence of BYTE
pOutBuffer: sequence of BYTE
pInBlock: ADDRESS OF COMPRESSED_DATA
cbInBufferCompress: DWORD
cbInBufferDeCompress: DWORD
cbInputProcessed: DWORD
cbDecompressedData: DWORD

if (dwOutVersion = 2) or
(dwOutVersion = 7) then
    /* decompress data that is compressed.
    */
    if (dwOutVersion = 2) then
        pInBuffer := msgOut.V2.CompressedV1.pbCompressedData
        cbInBufferCompress := msgOut.CompressedV1.cbCompressedSize
        cbInBufferDeCompress := msgOut.CompressedV1.cbUncompressedSize
    else if (dwOutVersion = 7) then
        pInBuffer := msgOut.V7.CompressedAny.pbCompressedData
        cbInBufferCompress := msgOut.V7.CompressedAny.cbCompressedSize
        cbInBufferDeCompress :=

```

```

        msgOut.V7.CompressedAny.cbUncompressedSize
    endif

    if (cbInBufferCompress = cbInBufferDecompress) then
        /* No decompression required here. */
        pOutBuffer := cbInBuffer
        cbOutBuffer := cbInBufferDeCompress
    else
        cbInputProcessed := 0
        while (cbInputProcessed ≤ cbInBufferCompress)
            pInBlock := ADR(pInputBuffer[cbInputProcessed])
            if (pInBlock^.cbDecompressedSize =
                pInBlock^.cbCompressedSize) then
                pDecompressedData := pInBlock^.data
                cbDecompressedData := pInBlock^.cbDecompressedSize
            else
                if (dwOutVersion = 2) or
                    (msgOut.V7.CompressionAlg = DRS_COMP_ALG_MSZIP) then
                    pDecompressedData :=
                        Decompress pInBlock^.data in accordance
                        with [RFC1951].
                else
                    pDecompressedData := new sequence of BYTE of length
                        pInBlock^.cbDecompressedSize
                    DecompressWin2k3(pInBlock^.data, pDecompressedData,
                        pInBlock^.cbDecompressedSize, cbDecompressedData)
                endif
                cbDecompressedData := pInBlock^.cbDecompressedSize
            endif
            pOutputBuffer := Append sequence of BYTE pDecompressedData of
                size cbDecompressedData to sequence of BYTE
                pOutputBuffer
            cbOutputBuffer :=
                cbOutputBuffer + pInBlock^.cbDecompressedSize
            cbInputProcessed := cbInputProcessed +
                pInBlock^.cbCompressedSize
            Round up value in cbInputProcessed such that
                ADR(pInBlock[cbInputProcessed]) align on double word
                boundary.
        endwhile
    endif

    /* pOutputBuffer now has the uncompressed data that was derived by
     * serializing a DRS_GETCHGREPLY structure at the server.
     * Convert the serialized data back to DRS_GETCHGREPLY structure.*/
    if dwOutVersion = 2 then
        dwOutVersion := 1
    else
        dwOutVersion := 6
    endif
    msgOut := Unpickling of data in pOutBuffer of length cbOutBuffer,
        as specified by [C311] Part 2, "IDL/NDR Pickles", and
        [MS-RPCE] sections 2.2.6 and 2.2.7

    return GetNCChangesV6Reply(msgOut, dwOutVersion)
endif

```


4.1.10.6.15 DecompressWin2k3

```
procedure DecompressWin2k3(  
    inputBuffer: sequence of BYTE,  
    ref outputBuffer: sequence of BYTE,  
    outputSize: DWORD,  
    ref inputConsumed: DWORD)
```

Informative summary of behavior: The DecompressWin2k3 procedure decompresses data that was compressed on the server. The procedure has the following parameters:

- **inputBuffer:** Sequence of [BYTE](#) containing compressed data.
- **outputBuffer:** Sequence of [BYTE](#) that is an empty buffer. Decompressed data will be filled into this buffer.
- **outputSize:** The [DWORD](#) value that indicates the size of data in bytes that can be accommodated in outputBuffer.
- **inputConsumed:** [DWORD](#) reference. At the end of the procedure, this parameter will have the number of bytes processed in inputBuffer.

```
outputIndex: DWORD  
inputIndex: DWORD  
indicator: DWORD  
indicatorBit: DWORD  
length: DWORD  
offset: DWORD  
nibbleIndex: DWORD  
  
outputIndex := 0  
inputIndex := 0  
indicator := 0  
indicatorBit := 0  
length := 0  
offset := 0  
nibbleIndex := 0  
  
while (outputIndex <= outputSize)  
    if (indicatorBit = 0) then  
        indicatorBit := copy inputBuffer[inputIndex] as 32-bit integer in  
            little-endian format  
        inputIndex := inputIndex + 4  
        indicatorBit := 32  
    endif  
  
    indicatorBit := indicatorBit - 1  
    /* check whether the bit specified by indicatorBit is set or not  
     * set in indicator. For example, if indicatorBit has value 4  
     * check whether the 4th bit of the value in indicator is set */  
    if indicatorBit bit in indicator is not set then  
        inputBuffer[inputIndex] := outputBuffer[outputIndex]  
        inputIndex := inputIndex + 1  
        outputIndex := outputIndex + 1  
    else  
        length := copy inputBuffer[inputIndex] as 16-bit integer in
```

```

        little-endian format
inputIndex := inputIndex + 2
offset := length / 8
length := length mod 8
if (length = 7) then
    if (nibbleIndex = 0) then
        nibbleIndex := inputIndex
        length := inputBuffer[inputIndex] mod 16
        inputIndex := inputIndex + 1
    else
        length := inputBuffer[inputIndex] / 16
        nibbleIndex := 0
    endif
endif
if (length = 15) then
    length := inputBuffer[inputIndex]
    inputIndex := inputIndex + 1
    if (length = 255) then
        length := copy inputBuffer[inputIndex] as 16-bit integer in
            little-endian format
        inputIndex := inputIndex + 2
        length := length - (15 + 7)
    endif
    length := length + 15
endif
length := length + 7
endif
length := length + 3
while (not length = 0)
    outputBuffer[outputIndex] :=
        outputBuffer[outputIndex - offset - 1]
    outputIndex := outputIndex + 1
    length := length - 1
endwhile
endif
endwhile
return

```

4.1.10.6.16 UndeleteObject

```

procedure UndeleteObject(
    obj: DSNAME,
    attributesAndStamps: set of AttributeAndStamp)

```

For each attStamp in attributesAndStamps, the UndeleteObject procedure performs an originating update to obj such that the value(s) of attStamp.attribute do not change, but AttrStamp(obj, attStamp.attribute).dwVersion > attStamp.stamp.dwVersion. The effect of this update to obj is such that this DC's values for these attributes replicate out to other DCs and overwrite the updates with stamps in attributesAndStamps.

4.1.10.7 Examples of the IDL_DRSGetNCChanges Method - Add User

4.1.10.7.1 Initial State

User "Kim Akers" is created on DC1 with the [sAMAccountName](#) "KimAkers"

```
ldap_add_s("CN=Kim Akers,CN=Users,DC=contoso,DC=com", [sAMAccountName])
```

Added {CN=Kim Akers,CN=Users,DC=contoso,DC=com }.

Querying the [nTDSDSA](#) objects for the root domain NC DC=CONTOSO, DC=COM for DC1:

- `ldap_search_s("CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com", baseObject, "(objectClass=*)", [objectClass, cn ... objectGUID])`
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=NTDS Settings,CN=DC1,CN=Servers, CN=Default-First-Site-Name,CN=Sites, CN=Configuration,DC=contoso,DC=com
 - 3> objectClass: top; applicationSettings; nTDSDSA;
 - 1> cn: NTDS Settings;
 - 1> distinguishedName: CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com;
 - 1> objectGUID: c20bc312-4d35-4cc0-9903-b1073368af4a;

Querying the [user](#) object "CN=Kim Akers, CN=Users, DC=CONTOSO, DC=COM" on DC1:

- `ldap_search_s("CN=Kim Akers,CN=Users,DC=contoso,DC=com", baseObject, "(objectClass=*)", [objectClass, cn ... objectCategory])`
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=Kim Akers,CN=Users,DC=contoso,DC=com
 - 4> objectClass: top; person; organizationalPerson; user;
 - 1> cn: Kim Akers;
 - 1> sn: Dow;
 - 1> givenName: Kim;
 - 1> distinguishedName: CN=Kim Akers,CN=Users,DC=contoso,DC=com;
 - 1> instanceType: 0x4 = (IT_WRITE);
 - 1> whenCreated: 07/17/2006 13:50:32 Pacific Standard Pacific Daylight Time;
 - 1> whenChanged: 07/17/2006 13:50:33 Pacific Standard Pacific Daylight Time;
 - 1> displayName: Kim Akers;

- 1> uSNCreated: 29345;
- 1> uSNChanged: 29350;
- 1> name: Kim Akers;
- 1> objectGUID: 39ab8618-d3fd-410c-b627-64b65104384d;
- 1> userAccountControl: 0x200 = (UF_NORMAL_ACCOUNT);
- 1> badPwdCount: 0;
- 1> codePage: 0;
- 1> countryCode: 0;
- 1> badPasswordTime: 01/01/1601 00:00:00 UNC ;
- 1> lastLogoff: 01/01/1601 00:00:00 UNC ;
- 1> lastLogon: 01/01/1601 00:00:00 UNC ;
- 1> pwdLastSet: 07/17/2006 13:50:33 Pacific Standard Time Pacific Daylight Time;
- 1> primaryGroupID: 513;
- 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1129;
- 1> accountExpires: 09/14/30828 02:48:05 UNC ;
- 1> logonCount: 0;
- 1> sAMAccountName: KimAkers;
- 1> sAMAccountType: SAM_NORMAL_USER_ACCOUNT;
- 1> userPrincipalName: KimAkers@contoso.com;
- 1> objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=contoso,DC=com;

Querying the [repsFrom](#) attribute on the NC root object for domain DC=CONTOSO, DC=COM on DC2:

- ldap_search_s("DC=contoso,DC=com", *baseObject*, "(objectclass=*)",)
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: DC=contoso,DC=com
 - 1> repsFrom: dwVersion = 1, V1.cb: 276, V1.cConsecutiveFailures: 4
 - V1.timeLastSuccess: 12797642625 V1.timeLastAttempt: 12797643058
 - V1.ulResultLastAttempt: 0x2108 V1.cbOtherDraOffset: 216
 - V1.cbOtherDra: 60 V1.ulReplicaFlags: 0x70

- V1.rtSchedule: <ldap:skipped> V1.usnvec.usnHighObjUpdate: 29322
V1.usnvec.usnHighPropUpdate: 29322
- V1.uuidDsaObj: c20bc312-4d35-4cc0-9903-b1073368af4a
V1.uuidInvocId: c20bc312-4d35-4cc0-9903-b1073368af4a
V1.uuidTransportObj: 00000000-0000-0000-0000-000000000000
- V1.mtx_address: c20bc312-4d35-4cc0-9903-b1073368af4a._msdcs.contoso.com
- V1.cbPASDataOffset: 0 V1.PasData: version = -1, size = -1, flag = -1 ;

Querying the [user](#) object "CN=Kim Akers, CN=Users, DC=CONTOSO, DC=COM" on DC2 returns no entries because the object is not present on DC2:

- `ldap_search_s("CN=Kim Akers,CN=Users,DC=contoso,DC=com", singleLevel, "(objectclass=*)", null)`
- Error: Search: No Such Object.
- Matched DNs: CN=Users,DC=contoso,DC=com
- Getting 0 entries:

4.1.10.7.2 Client Request

DC2 invokes the [IDL DRSGetNCChanges](#) method against DC1, with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 8
- pmsgIn = DRS_MSG_GETCHGREQ_V8
 - Destination DSA objGuid: _GUID {6aad8f5a-07cc-403a-9696-9102fe1c320b}
 - Source DSA Invocation ID: _GUID {c20bc312-4d35-4cc0-9903-b1073368af4a}
 - usnvecFrom: USN_VECTOR
 - usnHighObjUpdate : 29322
 - usnHighPropUpdate : 29322
 - pUpToDateVecDest : UPTODATE_VECTOR_V1_EXT
 - DSA Invoc ID: 9876730c-5844-4c94-b0bd-28458be39333, USN: 27359
 - DSA Invoc ID: c20bc312-4d35-4cc0-9903-b1073368af4a, USN: 29335
 - ulFlags:
 - DRS_ASYNC_OP
 - DRS_WRIT_REP
 - DRS_INIT_SYNC

- DRS_PER_SYNC
- Max objects to return: 535
- Max bytes to return: 5357731
- Extended operation: none
- Fsmo Info: 0
- PrefixTableDest : SCHEMA_PREFIX_TABLE

4.1.10.7.3 Server Response

Return code of 0 with the following values:

- pdwOutVersion= 6
- pmsgOut = DRS_MSG_GETCHGREPLY_V6
 - uuidDsaObjSrc: _GUID {c20bc312-4d35-4cc0-9903-b1073368af4a}
 - uuidInvocIdSrc: _GUID {c20bc312-4d35-4cc0-9903-b1073368af4a}
 - pNC: DSNAME DC=CONTOSO,DC=COM
 - usnvecFrom : USN_VECTOR
 - usnHighObjUpdate : 29322
 - usnHighPropUpdate : 29322
 - usnvecTo: USN_VECTOR
 - usnHighObjUpdate : 29379
 - usnHighPropUpdate : 29379
 - pUpToDateVecSrc : UPTODATE_VECTOR_V2_EXT
 - DSA Invoc ID: c20bc312-4d35-4cc0-9903-b1073368af4a,
 - usnHighPropUpdate : 29379, timeLastSyncSuccess : 12797643933
 - PrefixTableSrc : SCHEMA_PREFIX_TABLE
 - pObjects: REPLENTINFLIST
 - objectClass: top; person; organizationalPerson; user;
 - sn: Akers;
 - givenName: Kim;
 - instanceType: 0x4 = (IT_WRITE);
 - whenCreated: 07/17/2006 13:50:32 Pacific Standard Daylight Time;
 - whenChanged: 07/17/2006 14:05:21 Pacific Standard Daylight Time;

- displayName: Kim Akers;
- nTSecurityDescriptor: *binary data*
- objectGUID: 39ab8618-d3fd-410c-b627-64b65104384d;
- codePage: 0;
- countryCode: 0;
- dBCSPwd: *binary data*
- logonHours: 0
- unicodePwd: *binary data*
- ntPwdHistory: *binary data*
- pwdLastSet: 07/17/2006 13:50:33 Pacific Standard Daylight Time;
- sAMAccountName: KimAkers;
- sAMAccountType: SAM_NORMAL_USER_ACCOUNT;
- userPrincipalName: KimAkers@contoso.com;
- objectCategory: CN=Person, CN=Schema, CN=Configuration, DC=contoso, DC=com;
- rgValues: (null)

4.1.10.7.4 Final State

Querying the [repsFrom](#) attribute on the NC root object for the domain DC=CONTOSO, DC=COM on DC2:

- ldap_search_s("DC=contoso,DC=com", *baseObject*, "(objectclass=*)", *repsFrom*)
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: DC=contoso,DC=com
 - 1> repsFrom: dwVersion = 1, V1.cb: 276, V1.cConsecutiveFailures: 0
 - V1.timeLastSuccess: 12797643933 V1.timeLastAttempt: 12797643933
V1.ulResultLastAttempt: 0x0 V1.cbOtherDraOffset: 216
 - V1.cbOtherDra: 60 V1.ulReplicaFlags: 0x70
 - V1.rtSchedule: <ldp:skipped> V1.usnvec.usnHighObjUpdate: 29379
V1.usnvec.usnHighPropUpdate: 29379
 - V1.uuidDsaObj: c20bc312-4d35-4cc0-9903-b1073368af4a

- V1.uuidInvocId: c20bc312-4d35-4cc0-9903-b1073368af4a V1.uuidTransportObj: 00000000-0000-0000-0000-000000000000 V1.mtx_address: c20bc312-4d35-4cc0-9903-b1073368af4a._msdcs.contoso.com
- V1.cbPASDataOffset: 0 V1.PasData: version = -1, size = -1, flag = -1

Querying the [user](#) object "CN=Kim Akers, CN=Users, DC=CONTOSO,DC=COM" on DC2, which is now present:

- ldap_search_s("CN=Kim Akers,CN=Users,DC=contoso,DC=com", *baseObject*, "(objectclass=*)", [*objectClass*, *cn ... objectCategory*])
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=Kim Akers,CN=Users,DC=contoso,DC=com
 - 4> objectClass: top; person; organizationalPerson; user;
 - 1> cn: Kim Akers;
 - 1> sn: Akers;
 - 1> givenName: Kim;
 - 1> distinguishedName: CN=Kim Akers,CN=Users,DC=contoso,DC=com;
 - 1> instanceType: 0x4 = (IT_WRITE);
 - 1> whenCreated: 07/17/2006 13:50:32 Pacific Standard Daylight Time;
 - 1> whenChanged: 07/17/2006 14:05:21 Pacific Standard Daylight Time;
 - 1> displayName: Kim Akers;
 - 1> uSNCreated: 38197;
 - 1> uSNChanged: 38197;
 - 1> name: Kim Akers;
 - 1> objectGUID: 39ab8618-d3fd-410c-b627-64b65104384d;
 - 1> userAccountControl: 0x200 = (UF_NORMAL_ACCOUNT);
 - 1> codePage: 0;
 - 1> countryCode: 0;
 - 1> pwdLastSet: 07/17/2006 13:50:33 Pacific Standard Daylight Time;
 - 1> primaryGroupID: 513;
 - 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1129;
 - 1> accountExpires: 09/14/30828 02:48:05 UNC ;

- 1> sAMAccountName: KimAkers;
- 1> sAMAccountType: SAM_NORMAL_USER_ACCOUNT;
- 1> userPrincipalName: KimAkers@contoso.com;
- 1> objectCategory: CN=Person, CN=Schema, CN=Configuration, DC=contoso, DC=com;

4.1.10.8 Examples of the IDL_DRSGetNCChanges Method - Add User to a Group

4.1.10.8.1 Initial State

User "Kim Akers" is added to the group "GroupA" on DC1.

Querying the [repsFrom](#) attribute on the NC root object for the domain DC=CONTOSO, DC=COM on DC2:

- ldap_search_s("DC=contoso,DC=com", *baseObject*, "(objectclass=*)", *repsFrom*)
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: DC=contoso,DC=com
 - 1> repsFrom: dwVersion = 1,
 V1.cb: 276, V1.cConsecutiveFailures: 3 V1.timeLastSuccess: 12797643933
 V1.timeLastAttempt: 12797645671 V1.ulResultLastAttempt: 0x2108 V1.cbOtherDraOffset: 216 V1.cbOtherDra: 60 V1.ulReplicaFlags: 0x70
 V1.rtSchedule: <ldap:skipped> V1.usnvec.usnHighObjUpdate: 29379
 V1.usnvec.usnHighPropUpdate: 29379
 V1.uuidDsaObj: c20bc312-4d35-4cc0-9903-b1073368af4a V1.uuidInvocId: c20bc312-4d35-4cc0-9903-b1073368af4a V1.uuidTransportObj: 00000000-0000-0000-0000-000000000000
 V1.mtx_address: c20bc312-4d35-4cc0-9903-b1073368af4a._msdcs.contoso.com
 V1.cbPASDataOffset: 0 V1.PasData: version = -1, size = -1, flag = -1 ;

Querying the [group](#) object "CN=GroupA, CN=Users, DC=CONTOSO, DC=COM" on DC1:

- ldap_search_s("CN=GroupA, CN=Users, DC=contoso, DC=com", *baseObject*, "(objectclass=*)", [*objectClass*, *cn* ... *objectCategory*])
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=GroupA,CN=Users,DC=contoso,DC=com
 - 2> objectClass: top; group;
 - 1> cn: GroupA;

- 2> member: CN=Kim Akers,CN=Users,DC=contoso,DC=com;
- 1> distinguishedName: CN=GroupA,CN=Users,DC=contoso,DC=com;
- 1> instanceType: 0x4 = (IT_WRITE);
- 1> whenCreated: 07/13/2006 12:25:35 Pacific Standard Daylight Time;
- 1> whenChanged: 07/17/2006 14:34:12 Pacific Standard Daylight Time;
- 1> uSNCreated: 16023;
- 1> uSNChanged: 29387;
- 1> name: GroupA;
- 1> objectGUID: 328ab893-b884-4e31-a73c-71740e261715;
- 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1114;
- 1> sAMAccountName: GroupA;
- 1> sAMAccountType: 536870912;
- 1> groupType: 0x80000004 = (GROUP_TYPE_RESOURCE_GROUP | GROUP_TYPE_SECURITY_ENABLED);
- 1> objectCategory: CN=Group,CN=Schema,CN=Configuration,DC=contoso,DC=com;

Querying the [group](#) object "CN=GroupA, CN=Users, DC=CONTOSO, DC=COM" on DC2, the [member](#) attribute value is not returned, as it is currently empty because this group has no members on DC2:

- ldap_search_s("CN=GroupA, CN=Users, DC=contoso, DC=com", baseObject, "(objectclass=*)", [objectClass, cn ... objectCategory])
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
 - >> Dn: CN=GroupA,CN=Users,DC=contoso,DC=com
 - 2> objectClass: top; group;
 - 1> cn: GroupA;
 - 1> distinguishedName: CN=GroupA,CN=Users,DC=contoso,DC=com;
 - 1> instanceType: 0x4 = (IT_WRITE);
 - 1> whenCreated: 07/13/2006 12:25:35 Pacific Standard Daylight Time;
 - 1> whenChanged: 07/13/2006 12:36:03 Pacific Standard Daylight Time;
 - 1> uSNCreated: 26457;
 - 1> uSNChanged: 26543;
 - 1> name: GroupA;

- 1> objectGUID: 328ab893-b884-4e31-a73c-71740e261715;
- 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1114;
- 1> sAMAccountName: GroupA;
- 1> sAMAccountType: 536870912;
- 1> groupType: 0x80000004 = (GROUP_TYPE_RESOURCE_GROUP | GROUP_TYPE_SECURITY_ENABLED);
- 1> objectCategory: CN=Group, CN=Schema, CN=Configuration, DC=contoso, DC=com;

4.1.10.8.2 Client Request

DC2 invokes the method [IDL DRSGetNCChanges](#) against DC1, with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 8
- pmsgIn = DRS_MSG_GETCHGREQ_V8
 - Destination DSA objGuid: _GUID {6aad8f5a-07cc-403a-9696-9102fe1c320b}
 - Source DSA Invocation ID: _GUID {c20bc312-4d35-4cc0-9903-b1073368af4a}
 - usnvecFrom: USN_VECTOR
 - usnHighObjUpdate : 29379
 - usnHighPropUpdate : 29379
 - pUpToDateVecDest : UPTODATE_VECTOR_V1_EXT
 - DSA Invoc ID: c20bc312-4d35-4cc0-9903-b1073368af4a, USN: 29379
 - Flags:
 - DRS_ASYNC_OP
 - DRS_WRIT_REP
 - DRS_INIT_SYNC
 - DRS_PER_SYNC
 - Max objects to return: 535
 - Max bytes to return: 5357731
 - Extended operation: none
 - Fsmo Info: 0
 - PrefixTableDest : SCHEMA_PREFIX_TABLE

4.1.10.8.3 Server Response

Return code of 0 with the following values:

- pdwOutVersion= 6
- pmsgOut = DRS_MSG_GETCHGREPLY_V6
 - uuidDsaObjSrc: _GUID {c20bc312-4d35-4cc0-9903-b1073368af4a}
 - uuidInvocIdSrc: _GUID {c20bc312-4d35-4cc0-9903-b1073368af4a}
 - pNC: DSNAME DC=CONTOSO,DC=COM
 - usnvecFrom : USN_VECTOR
 - usnHighObjUpdate : 29379
 - usnHighPropUpdate : 29379
 - usnvecTo: USN_VECTOR
 - usnHighObjUpdate : 29389
 - usnHighPropUpdate : 29389
 - pUpToDateVecSrc : UPTODATE_VECTOR_V2_EXT
 - DSA Invoc ID: c20bc312-4d35-4cc0-9903-b1073368af4a
 - usnHighPropUpdate : 29389, timeLastSyncSuccess : 12797646597
 - PrefixTableSrc : SCHEMA_PREFIX_TABLE
 - pObjects: (null)
 - rgValues: REPLVALINF
 - pObject: CN=Kim Akers,CN=Users,DC=contoso,DC=com
 - attrTyp: "member"

4.1.10.8.4 Final State

Querying the [repsFrom](#) attribute on the NC root object for the domain DC=CONTOSO, DC=COM on DC2:

- ldap_search_s("DC=contoso,DC=com", *baseObject*, "(objectclass=*)", *repsFrom*)
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: DC=contoso,DC=com
 - 1> repsFrom: dwVersion = 1, V1.cb: 276, V1.cConsecutiveFailures: 0
 - V1.timeLastSuccess: 12797646597 V1.timeLastAttempt: 12797646597
V1.ulResultLastAttempt: 0x0 V1.cbOtherDraOffset: 216
 - V1.cbOtherDra: 60 V1.ulReplicaFlags: 0x70

- V1.rtSchedule: <ldap:skipped> V1.usnvec.usnHighObjUpdate: 29389
- V1.usnvec.usnHighPropUpdate: 29389
- V1.uuidDsaObj: c20bc312-4d35-4cc0-9903-b1073368af4a
- V1.uuidInvocId: c20bc312-4d35-4cc0-9903-b1073368af4a
- V1.uuidTransportObj: 00000000-0000-0000-0000-000000000000 V1.mtx_address: c20bc312-4d35-4cc0-9903-b1073368af4a._msdcs.contoso.com
- V1.cbPASDataOffset: 0 V1.PasData: version = -1, size = -1, flag = -1 ;

Querying the [group](#) object "CN=GroupA, CN=Users, DC=CONTOSO, DC=COM" on DC2:

- ldap_search_s("CN=GroupA,CN=Users,DC=contoso,DC=com", baseObject, "(objectclass=*)", [objectClass, cn ... objectCategory])
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=GroupA,CN=Users,DC=contoso,DC=com
 - 2> objectClass: top; group;
 - 1> cn: GroupA;
 - 1> member: CN=Kim Akers,CN=Users,DC=contoso,DC=com
 - 1> distinguishedName: CN=GroupA,CN=Users,DC=contoso,DC=com;
 - 1> instanceType: 0x4 = (IT_WRITE);
 - 1> whenCreated: 07/13/2006 12:25:35 Pacific Standard Daylight Time;
 - 1> whenChanged: 07/17/2006 14:49:46 Pacific Standard Daylight Time;
 - 1> uSNCreated: 26457;
 - 1> uSNChanged: 38218;
 - 1> name: GroupA;
 - 1> objectGUID: 328ab893-b884-4e31-a73c-71740e261715;
 - 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1114;
 - 1> sAMAccountName: GroupA;
 - 1> sAMAccountType: 536870912;
 - 1> groupType: 0x80000004 = (GROUP_TYPE_RESOURCE_GROUP | GROUP_TYPE_SECURITY_ENABLED);
 - 1> objectCategory: CN=Group, CN=Schema, CN=Configuration, DC=contoso, DC=com;

4.1.10.9 Examples of the IDL_DRSGetNCChanges Method - Change User Password

4.1.10.9.1 Initial State

User Kim Akers changes the password by pressing CTRL+ALT+DELETE, and the password change is processed by DC1.

Querying the [repsFrom](#) attribute on the NC root object for domain DC=CONTOSO, DC=COM on DC2:

- `ldap_search_s("DC=contoso,DC=com", baseObject, "(objectclass=*)", repsFrom)`
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: DC=contoso,DC=com
 - 1> repsFrom: dwVersion = 1, V1.cb: 276, V1.cConsecutiveFailures: 4
 - V1.timeLastSuccess: 12797646597 V1.timeLastAttempt: 12797646597
 - V1.ulResultLastAttempt: 0x2108 V1.cbOtherDraOffset: 216 V1.cbOtherDra: 60
 - V1.ulReplicaFlags: 0x70
 - V1.rtSchedule: <ldp:skipped> V1.usnvec.usnHighObjUpdate: 29389
 - V1.usnvec.usnHighPropUpdate: 29389
 - V1.uuidDsaObj: c20bc312-4d35-4cc0-9903-b1073368af4a
 - V1.uuidInvocId: c20bc312-4d35-4cc0-9903-b1073368af4a
 - V1.uuidTransportObj: 00000000-0000-0000-0000-000000000000 V1.mtx_address: c20bc312-4d35-4cc0-9903-b1073368af4a._msdcs.contoso.com
 - V1.cbPASDataOffset: 0 V1.PasData: version = -1, size = -1, flag = -1;

Querying [user](#) object "CN=Kim Akers, CN=Users, DC=CONTOSO,DC=COM" on DC1:

- `ldap_search_s("CN=Kim Akers,CN=Users,DC=contoso,DC=com", baseObject, "(objectClass=*)", [objectClass, cn ... objectCategory])`
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=Kim Akers,CN=Users,DC=contoso,DC=com
 - 4> objectClass: top; person; organizationalPerson; user;
 - 1> cn: Kim Akers;
 - 1> sn: Akers;
 - 1> givenName: Kim;

- 1> distinguishedName: CN=Kim Akers,CN=Users,DC=contoso,DC=com;
- 1> instanceType: 0x4 = (IT_WRITE);
- 1> whenCreated: 07/17/2006 13:50:32 Pacific Standard Daylight Time;
- 1> whenChanged: 07/17/2006 14:58:36 Pacific Standard Daylight Time;
- 1> displayName: Kim Akers;
- 1> uSNCreated: 29345;
- 1> memberOf: CN=GroupA,CN=Users,DC=contoso,DC=com;
- 1> uSNChanged: 29408;
- 1> name: Kim Akers;
- 1> objectGUID: 39ab8618-d3fd-410c-b627-64b65104384d;
- 1> userAccountControl: 0x200 = (UF_NORMAL_ACCOUNT);
- 1> badPwdCount: 0;
- 1> codePage: 0;
- 1> countryCode: 0;
- 1> badPasswordTime: 01/01/1601 00:00:00 UNC ;
- 1> lastLogoff: 01/01/1601 00:00:00 UNC ;
- 1> lastLogon: 01/01/1601 00:00:00 UNC ;
- 1> pwdLastSet: 07/17/2006 14:58:36 Pacific Standard Daylight Time;
- 1> primaryGroupID: 513;
- 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1129;
- 1> accountExpires: 09/14/30828 02:48:05 UNC ;
- 1> logonCount: 0;
- 1> sAMAccountName: KimAkers;
- 1> sAMAccountType: 805306368;
- 1> userPrincipalName: KimAkers@contoso.com;
- 1> objectCategory: CN=Person, CN=Schema, CN=Configuration, DC=contoso, DC=com;

Querying [user](#) object "CN=Kim Akers, CN=Users, DC=CONTOSO,DC=COM" on DC2:

- ldap_search_s("CN=Kim Akers,CN=Users,DC=contoso,DC=com", *baseObject*, "(objectclass=*)", [*objectClass*, *cn* ... *objectCategory*])
- Result <0>: (null)
- Matched DNs:

- Getting 1 entries:
- >> Dn: CN=Kim Akers,CN=Users,DC=contoso,DC=com
 - 4> objectClass: top; person; organizationalPerson; user;
 - 1> cn: Kim Akers;
 - 1> sn: Akers;
 - 1> givenName: Kim;
 - 1> distinguishedName: CN=Kim Akers,CN=Users,DC=contoso,DC=com;
 - 1> instanceType: 0x4 = (IT_WRITE);
 - 1> whenCreated: 07/17/2006 13:50:32 Pacific Standard Daylight Time;
 - 1> whenChanged: 07/17/2006 14:05:21 Pacific Standard Daylight Time;
 - 1> displayName: Kim Akers;
 - 1> uSNCreated: 38197;
 - 1> memberOf: CN=GroupA,CN=Users,DC=contoso,DC=com;
 - 1> uSNChanged: 38197;
 - 1> name: Kim Akers;
 - 1> objectGUID: 39ab8618-d3fd-410c-b627-64b65104384d;
 - 1> userAccountControl: 0x200 = (UF_NORMAL_ACCOUNT);
 - 1> codePage: 0;
 - 1> countryCode: 0;
 - 1> pwdLastSet: 07/17/2006 13:50:33 Pacific Standard Daylight Time;
 - 1> primaryGroupID: 513;
 - 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1129;
 - 1> accountExpires: 09/14/30828 02:48:05 UNC ;
 - 1> sAMAccountName: KimAkers;
 - 1> sAMAccountType: 805306368;
 - 1> userPrincipalName: KimAkers@contoso.com;
 - 1> objectCategory: CN=Person, CN=Schema, CN=Configuration, DC=contoso,DC=com;

4.1.10.9.2 Client Request

DC2 invokes the method [IDL DRSGetNCChanges](#) against DC1, with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 8
- pmsgIn = DRS_MSG_GETCHGREQ_V8
 - Destination DSA objGuid: _GUID {6aad8f5a-07cc-403a-9696-9102fe1c320b}
 - Source DSA Invocation ID: _GUID {c20bc312-4d35-4cc0-9903-b1073368af4a}
 - usnvecFrom: USN_VECTOR
 - usnHighPropUpdate : 29389
 - usnHighObjUpdate : 29389
 - pUpToDateVecDest : UPTODATE_VECTOR_V1_EXT
 - DSA Invoc ID: c20bc312-4d35-4cc0-9903-b1073368af4a, USN: 29389
 - Flags:
 - DRS_ASYNC_OP
 - DRS_WRIT_REP
 - DRS_INIT_SYNC
 - DRS_PER_SYNC
 - Max objects to return: 535
 - Max bytes to return: 5357731
 - Extended operation: none
 - Fsmo Info: 0
 - PrefixTableDest : SCHEMA_PREFIX_TABLE

4.1.10.9.3 Server Response

A return code of 0 with the following values:

- pdwOutVersion= 6
- pmsgOut = DRS_MSG_GETCHGREPLY_V6
 - uuidDsaObjSrc: _GUID {c20bc312-4d35-4cc0-9903-b1073368af4a}
 - uuidInvocIdSrc: _GUID {c20bc312-4d35-4cc0-9903-b1073368af4a}
 - pNC: DSNAME DC=CONTOSO,DC=COM
 - usnvecFrom : USN_VECTOR
 - usnHighObjUpdate : 29389
 - usnHighPropUpdate : 29389
 - usnvecTo : USN_VECTOR

- usnHighObjUpdate : 29438
- usnHighPropUpdate : 29438
- pUpToDateVecSrc : UPTODATE_VECTOR_V2_EXT
 - DSA Invoc ID: c20bc312-4d35-4cc0-9903-b1073368af4a,
 - usnHighPropUpdate : 29438, timeLastSyncSuccess : 12797647962
- PrefixTableSrc : SCHEMA_PREFIX_TABLE
- pObjects: REPLENTINFLIST
 - instanceType: IT_WRITE
 - dBCSPwd: *binary data*
 - unicodePwd: *binary data*
 - ntPwdHistory: *binary data*
 - pwdLastSet: 07/17/2006 14:58:36 Pacific Standard Daylight Time
 - supplementalCredentials: *binary data*
 - lmPwdHistory: *binary data*

4.1.10.9.4 Final State

Querying the [repsFrom](#) attribute on the NC root object for domain DC=CONTOSO, DC=COM on DC2:

- ldap_search_s("DC=contoso,DC=com", *baseObject*, "(objectclass=*)", *repsFrom*)
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >>Dn: DC=contoso,DC=com
 - 1> repsFrom: dwVersion = 1, V1.cb: 276, V1.cConsecutiveFailures: 0
 - V1.timeLastSuccess: 12797647962 V1.timeLastAttempt: 12797647962
V1.ulResultLastAttempt: 0x70 V1.cbOtherDraOffset: 216 V1.cbOtherDra: 60
V1.ulReplicaFlags: 0x70 V1.rtSchedule: <ldap:skipped> V1.usnvec.usnHighObjUpdate: 29438 V1.usnvec.usnHighPropUpdate: 29438 V1.uuidDsaObj: c20bc312-4d35-4cc0-9903-b1073368af4a
 - V1.uuidInvocId: c20bc312-4d35-4cc0-9903-b1073368af4a
 - V1.uuidTransportObj: 00000000-0000-0000-0000-000000000000 V1.mtx_address: c20bc312-4d35-4cc0-9903-b1073368af4a._msdcs.contoso.com
 - V1.cbPASDataOffset: 0 V1.PasData: version = -1, size = -1, flag = -1;

Querying [user](#) object "CN=Kim Akers, CN=Users, DC=CONTOSO,DC=COM" on DC2, which has now been updated:

- `ldap_search_s("CN=Kim Akers,CN=Users,DC=contoso,DC=com", baseObject, "(objectclass=*)", [objectClass, cn ... objectCategory])`
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=Kim Akers,CN=Users,DC=contoso,DC=com
 - 4> objectClass: top; person; organizationalPerson; user;
 - 1> cn: Kim Akers;
 - 1> sn: Akers;
 - 1> givenName: Kim;
 - 1> distinguishedName: CN=Kim Akers,CN=Users,DC=contoso,DC=com;
 - 1> instanceType: 0x4 = (IT_WRITE);
 - 1> whenCreated: 07/17/2006 13:50:32 Pacific Standard Daylight Time;
 - 1> whenChanged: 07/17/2006 15:12:35 Pacific Standard Daylight Time;
 - 1> displayName: Kim Akers;
 - 1> uSNCreated: 38197;
 - 1> memberOf: CN=GroupA,CN=Users,DC=contoso,DC=com;
 - 1> uSNChanged: 38270;
 - 1> name: Kim Akers;
 - 1> objectGUID: 39ab8618-d3fd-410c-b627-64b65104384d;
 - 1> userAccountControl: 0x200 = (UF_NORMAL_ACCOUNT);
 - 1> codePage: 0;
 - 1> countryCode: 0;
 - 1> pwdLastSet: 07/17/2006 14:58:36 Pacific Standard Daylight Time;
 - 1> primaryGroupID: 513;
 - 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1129;
 - 1> accountExpires: 09/14/30828 02:48:05 UNC ;
 - 1> sAMAccountName: KimAkers;
 - 1> sAMAccountType: 805306368;
 - 1> userPrincipalName: KimAkers@contoso.com;
 - 1> objectCategory: CN=Person, CN=Schema, CN=Configuration, DC=contoso, DC=com;

4.1.11 IDL_DRSGetNT4ChangeLog (Opnum 11)

If the server is the **PDC emulator** FSMO role owner, the **IDL_DRSGetNT4ChangeLog** method returns either a sequence of PDC change log entries or the NT4 replication state, or both, as requested by the client.

```
ULONG IDL_DRSGetNT4ChangeLog(  
    [in, ref] DRS_HANDLE hDrs,  
    [in] DWORD dwInVersion,  
    [in, ref, switch_is(dwInVersion)]  
        DRS_MSG_NT4_CHGLOG_REQ* pmsgIn,  
    [out, ref] DWORD* pdwOutVersion,  
    [out, ref, switch_is(*pdwOutVersion)]  
        DRS_MSG_NT4_CHGLOG_REPLY* pmsgOut  
);
```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 or ERROR_MORE_DATA if successful; another Windows error code if a failure occurred.

4.1.11.1 Method-Specific Concrete Types

4.1.11.1.1 DRS_MSG_NT4_CHGLOG_REQ

The **DRS_MSG_NT4_CHGLOG_REQ** union defines the request messages sent to the [IDL_DRSGetNT4ChangeLog](#) method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef  
[switch_type(DWORD)]  
union {  
    [case(1)]  
        DRS_MSG_NT4_CHGLOG_REQ_V1 V1;  
} DRS_MSG_NT4_CHGLOG_REQ;
```

V1: The version 1 request.

4.1.11.1.2 DRS_MSG_NT4_CHGLOG_REQ_V1

The **DRS_MSG_NT4_CHGLOG_REQ_V1** structure defines the request message sent to the [IDL_DRSGetNT4ChangeLog](#) method.

```
typedef struct {
```

```

DWORD dwFlags;
DWORD PreferredMaximumLength;
[range(0,10485760)] DWORD cbRestart;
[size_is(cbRestart)] BYTE* pRestart;
} DRS_MSG_NT4_CHGLOG_REQ_V1;

```

dwFlags: Zero or more of the following bit flags:

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
C	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	X	X	X	X	X	X
L	N																														

X: Unused. MUST be zero and ignored.

CL (DRS_NT4_CHGLOG_GET_CHANGE_LOG): If set, the server returns the PDC change log.

SN (DRS_NT4_CHGLOG_GET_SERIAL_NUMBERS): If set, the server returns the NT4 replication state.

PreferredMaximumLength: The maximum size, in bytes, of the change log data that should be retrieved in a single operation.

cbRestart: Zero if pRestart = null. Otherwise, the size, in bytes, of pRestart^.

pRestart: Null to request the change log from the beginning. Otherwise, a pointer to an opaque value, returned in some previous call to **IDL_DRSGetNT4ChangeLog**, identifying the last change log entry returned in that previous call.

4.1.11.1.3 DRS_MSG_NT4_CHGLOG_REPLY

The **DRS_MSG_NT4_CHGLOG_REPLY** union defines the response messages received from the [IDL_DRSGetNT4ChangeLog](#) method. Only one version, identified by dwVersion = 1, is currently defined.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_NT4_CHGLOG_REPLY_V1 V1;
} DRS_MSG_NT4_CHGLOG_REPLY;

```

V1: The version 1 reply.

4.1.11.1.4 DRS_MSG_NT4_CHGLOG_REPLY_V1

The **DRS_MSG_NT4_CHGLOG_REPLY_V1** structure defines the response message received from the [IDL_DRSGetNT4ChangeLog](#) method.

```
typedef struct {
    [range(0,10485760)] DWORD cbRestart;
    [range(0,10485760)] DWORD cbLog;
    NT4_REPLICATION_STATE ReplicationState;
    DWORD ActualNtStatus;
    [size_is(cbRestart)] BYTE* pRestart;
    [size_is(cbLog)] BYTE* pLog;
} DRS_MSG_NT4_CHGLOG_REPLY_V1;
```

cbRestart: Zero if pRestart = null. Otherwise, the size, in bytes, of pRestart^.

cbLog: Zero if pRestart = null. Otherwise, the size, in bytes, of pLog^.

ReplicationState: The replication state for Windows NT 4.0 DCs.

ActualNtStatus: A **STATUS code**. See the pseudo-code for interpretation.

pRestart: Null if no entries were returned. Otherwise, a pointer to an opaque value identifying the last entry returned in pLog.

pLog: The buffer containing the next entries from the change log.

4.1.11.1.5 NT4_REPLICATION_STATE

The **NT4_REPLICATION_STATE** structure defines the replication state for Windows NT 4.0 DCs, whose interpretation is specified in [\[MS-ADTS\]](#) section 3.1.1.7.1.

```
typedef struct {
    LARGE_INTEGER SamSerialNumber;
    LARGE_INTEGER SamCreationTime;
    LARGE_INTEGER BuiltinSerialNumber;
    LARGE_INTEGER BuiltinCreationTime;
    LARGE_INTEGER LsaSerialNumber;
    LARGE_INTEGER LsaCreationTime;
} NT4_REPLICATION_STATE;
```

SamSerialNumber: The Windows NT 4.0 replication update sequence number for the SAM database.

SamCreationTime: The time at which the Windows NT 4.0 replication update sequence number for the SAM database was set to 1.

BuiltinSerialNumber: The Windows NT 4.0 replication update sequence number for the built-in database.

BuiltinCreationTime: The time at which the Windows NT 4.0 replication update sequence number for the built-in database was set to 1.

LsaSerialNumber: The Windows NT 4.0 replication update sequence number for the local security authority (LSA) database.

LsaCreationTime: The time at which the Windows NT 4.0 replication update sequence number for the LSA database was set to 1.

4.1.11.2 Method-Specific Abstract Types and Procedures

4.1.11.2.1 IsPDC

```
procedure IsPDC(): boolean
```

Returns true if the DC owns the PDC role for this domain, otherwise, false.

4.1.11.2.2 GetWindowsErrorCode

```
procedure GetWindowsErrorCode(ntStatus: DWORD): DWORD
```

Returns the Windows error code corresponding to the specified STATUS code.

4.1.11.3 Server Behavior of the IDL_DRSGetNT4ChangeLog Method

Informative summary of behavior: If the server is the PDC emulatorFSMO role owner, it returns either a sequence of PDC change log entries or the NT4 replication state, or both, as requested by the client.

Multiple calls of this method may be required to retrieve the entire PDC change log. The client passes pRestart = null on the first call in a series of calls; the server returns a sequence of change log entries, including the first, a pointer to an opaque cookie, and a result code. If the server returns no change log entries, it returns null instead of a pointer to a cookie. If the server returns the result code zero, the sequence of change log entries in the response includes the final entry in the log.

The cookie encodes the serial number of the last change log entry returned. If the server returns ERROR_MORE_DATA, the final change log entry in the response was not the final entry in the change log. The client can make another call, with pRestart pointing to the cookie. The server processes this call identically to a call with pRestart = null, except that it returns change log entries starting with the entry following the last previously returned entry, as indicated by the cookie. By making enough calls the client can retrieve the entire change log.

If the client includes a cookie that is either corrupted or identifies a nonexistent change log entry (possibly because the cookie is too old), the server returns ERROR_INVALID_PARAMETER. If there are change log entries to return, but the client specifies a bound on the size of the returned change log entries that is too small to hold even a single entry, the server returns ERROR_INSUFFICIENT_BUFFER.

The NT4 replication state is a small, fixed-size structure and the server simply copies it into the response.

When the client requests both the PDC change log and the NT4 replication state, the server processes the PDC change log request first. If an error occurs during this processing the server does not process the request for NT4 replication state. If an error occurs while processing the NT4 replication state request, the server returns no indication to the client that the PDC change log request succeeded.

```
ULONG
IDL_DRSGetNT4ChangeLog(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_NT4_CHGLOG_REQ *pmsgIn,
```

```

[out, ref] DWORD *pdwOutVersion,
[out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_NT4_CHGLOG_REPLY *pmsgOut)

msgIn: DRS_MSG_NT4_CHGLOG_REQ_V1
readStatus, ntStatus: DWORD
sequenceNumber: integer
nextIndexToBeReturned, lastIndexToBeReturned: integer
lastReturnedSerialNumber: LONGLONG
lastReturnedIndex: integer
pChangeLog: ADDRESS OF CHANGE_LOG_ENTRIES

/* Validate the request version */
if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1

/* Access check */
if not AccessCheckCAR(DefaultNC(), DS-Replication-Get-Changes) then
    return ERROR_ACCESS_DENIED
endif

/* The DC must own the PDC role */
if not IsPDC() then
    return ERROR_INVALID_DOMAIN_ROLE
endif

ntStatus := 0
readStatus := 0
if DRS_NT4_CHGLOG_GET_CHANGE_LOG in msgIn.dwFlags then
    /* Return NT4 change log entries. */

    /* Determine the position of the first entry in the change log that
     * needs to be returned. If pRestart = null, this is the first
     * entry of the change log, otherwise it is the entry following the
     * entry identified in the cookie pRestart^. */
    if msgIn.pRestart = null then
        sequenceNumber := 1
        nextIndexToBeReturned := 0
    else
        sequenceNumber :=
            (Sequence number extracted from msgIn.pRestart^) + 1
        lastReturnedSerialNumber :=
            Serial number extracted from msgIn.pRestart^
        lastReturnedIndex := select one i in dc.pdcChangeLog where
            dc.pdcChangeLog[i].SerialNumber = lastReturnedSerialNumber
        if lastReturnedIndex = null then
            /* Cookie is old or corrupted.
             * The STATUS code STATUS_INVALID_PARAMETER corresponds to
             * the Windows error code ERROR_INVALID_PARAMETER. */
            ntStatus := STATUS_INVALID_PARAMETER
        else
            nextIndexToBeReturned := lastReturnedIndex + 1
        endif
    endif
endif

if ntStatus = 0 and nextIndexToBeReturned ≥ dc.pdcChangeLog.length

```



```

        then
        /* No entries to be returned, complete the response message */
        pmsgOut^.Vl.pLog := null
        pmsgOut^.Vl.cbLog := 0
        pmsgOut^.Vl.pRestart := null
        pmsgOut^.Vl.cbRestart := 0
    endif

    if ntStatus = 0 and
        nextIndexToBeReturned < dc.pdcChangeLog.length then
        /* Entries to be returned. First, determine how many entries fit
        * into the response message */
        lastIndexToBeReturned := the largest integer q such that
            q < dc.pdcChangeLog.length and
            the size in bytes of
                dc.pdcChangeLog[nextIndexToBeReturned .. q]
            is <= msgIn.PreferredMaximumLength
        if lastIndexToBeReturned < nextIndexToBeReturned then
        /* Client's PreferredMaximumLength is too small for a single
        * entry, so return no entries.
        * The STATUS code STATUS_BUFFER_TOO_SMALL corresponds to
        * the Windows error code ERROR_INSUFFICIENT_BUFFER. */
            ntStatus := STATUS_BUFFER_TOO_SMALL
        else
        /* Client's PreferredMaximumLength is large enough for one or
        * more entries. Fill in pChangeLog^ from dc.pdcChangeLog */
            pChangeLog^.Size := 0x00000010
            pChangeLog^.Version := 0x00000001
            pChangeLog^.SequenceNumber := sequenceNumber
            pChangeLog^.Flags := 0x00000000
            pChangeLog^.ChangeLogEntries :=
                dc.pdcChangeLog[nextIndexToBeReturned ..
                    lastIndexToBeReturned]
            if a fatal error occurred while retrieving dc.pdcChangeLog then
                ntStatus :=
                    STATUS code of error that occurred, high-order bit set
            end
        endif
    endif
    if ntStatus = 0 then
        /* No errors, complete the response message */
        pmsgOut^.Vl.pLog := pChangeLog
        pmsgOut^.Vl.cbLog := size in bytes of pmsgOut^.Vl.pLog^
        /* Construct a new cookie */
        lastReturnedSerialNumber :=
            dc.pdcChangeLog[lastIndexToBeReturned].SerialNumber
        pmsgOut^.Vl.pRestart :=
            ADDRESS OF implementation-specific struct
            encapsulating lastReturnedSerialNumber and sequenceNumber
        pmsgOut^.Vl.cbRestart := size in bytes of pmsgOut^.Vl.pRestart^
        if lastIndexToBeReturned < dc.pdcChangeLog.length - 1 then
        /* There are more entries to be returned.
        * The STATUS code STATUS_MORE_ENTRIES corresponds to
        * the Windows error code ERROR_MORE_DATA. */
            ntStatus := STATUS_MORE_ENTRIES
        endif
    endif /* Response complete */
endif /* Entries returned */
endif /* Processed change log request */

```

```

/* Save the status code from the previous operation */
readStatus := ntStatus

if ntStatus < 0x80000000 and
    DRS_NT4_CHGLOG_GET_SERIAL_NUMBERS in msgIn.dwFlags then
    /* Return NT4 replication state. */
    pmsgOut^.V1.ReplicationState.SamSerialNumber :=
        dc.nt4ReplicationState.SamNT4ReplicationUSN
    pmsgOut^.V1.ReplicationState.SamCreationTime :=
        dc.nt4ReplicationState.SamCreationTime
    pmsgOut^.V1.ReplicationState.BuiltinSerialNumber :=
        dc.nt4ReplicationState.BuiltinNT4ReplicationUSN
    pmsgOut^.V1.ReplicationState.BuiltinCreationTime :=
        dc.nt4ReplicationState.BuiltinCreationTime
    pmsgOut^.V1.ReplicationState.LsaSerialNumber := 1
    pmsgOut^.V1.ReplicationState.LsaCreationTime :=
        current time on the DC
    if a fatal error occurred while retrieving NT4 replication state
    then
        ntStatus :=
            STATUS code of error that occurred, high-order bit set
    end
endif

if ntStatus < 0x80000000 then
    pmsgOut^.V1.ActualStatus := readStatus
else
    pmsgOut^.V1.ActualStatus := ntStatus
endif

return GetWindowsErrorCode(ntStatus)

```

4.1.11.4 Examples of the IDL_DRSGetNT4ChangeLog Method

4.1.11.4.1 Initial State

Domain functional level is Windows 2000 mixed. PDC role is held by DC2.

- `ldap_search_s("DC=contoso,DC=com", baseObject, "(objectClass=*)")`
- `>> Dn: DC=contoso,DC=com`
 - `1> fSMORoleOwner: CN=NTDS Settings, CN=DC2, CN=Servers,`
`CN=Default-First-Site-Name, CN=Sites, CN=Configuration,`
`DC=contoso, DC=com;`

4.1.11.4.2 Client Request

The PDC role is transferred to DC1, which results in DC1 invoking the [IDL_DRSGetNT4ChangeLog](#) method against DC2 with the following parameters ([DRS_HANDLE](#) to DC2 omitted):

```

dwMsgVersion = 1
pmsgIn =

```

```

dwFlags: DRS_NT4_CHGLOG_GET_CHANGE_LOG |
          DRS_NT4_CHGLOG_GET_SERIAL_NUMBERS
PreferredMaximumLength: 0x4000
cbRestart: 0
pRestart: NULL

```

4.1.11.4.3 Server Response

Return code of 0 with the following values:

```

pmsgOut = DRS_MSG_NT4_CHGLOG_REPLY_V1
cbRestart = 0x10
cbLog = 0x2d00
ReplicationState = _NT4_REPLICATION_STATE
    SamSerialNumber = 0x30`00000097
    SamCreationTime = 0x1c6a7a9`792f51f6
    BuiltinSerialNumber = 0x30`00000054
    BuiltinCreationTime = 0x1c6a7a9`792f51f6
    LsaSerialNumber = 0x1
    LsaCreationTime = 0x1c6a832`0a495151
ActualNtStatus = 0
pRestart = "LMEM"
pLog = pointer to actual log (log data omitted)

```

4.1.11.4.4 Final State

The PDC change log entries on DC1 are synchronized with the change log entries on DC2; there is no other change in state.

4.1.12 IDL_DRSGetObjectExistence (Opnum 23)

The **IDL_DRSGetObjectExistence** method helps the client check the consistency of object existence between its replica of an NC and the server's replica of the same NC. Checking the consistency of object existence means identifying objects that have replicated to both replicas and that exist in one replica but not in the other. For the purposes of this method, an object *exists* within a NC replica if it is either an object or a tombstone.

See [IDL_DRSReplicaVerifyObjects](#) for a use of this method.

```

ULONG IDL_DRSGetObjectExistence(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_EXISTREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_EXISTREPLY* pmsgOut
);

```

hDrs: The **RPC** context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.12.1 Method-Specific Concrete Types

4.1.12.1.1 DRS_MSG_EXISTREQ

The **DRS_MSG_EXISTREQ** union defines request messages sent to the [IDL DRSGetObjectExistence](#) method. Only one version, identified by dwVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_EXISTREQ_V1 V1;
} DRS_MSG_EXISTREQ;
```

V1: The version 1 request.

4.1.12.1.2 DRS_MSG_EXISTREQ_V1

The **DRS_MSG_EXISTREQ_V1** structure defines the request message sent to the [IDL DRSGetObjectExistence](#) method.

```
typedef struct {
    UUID guidStart;
    DWORD cGuids;
    DSNAME* pNC;
    UPTODATE_VECTOR_V1_EXT* pUpToDateVecCommonV1;
    UCHAR Md5Digest[16];
} DRS_MSG_EXISTREQ_V1;
```

guidStart: The [objectGUID](#) of the first object in the client's object sequence.

cGuids: The number of objects in the client's object sequence.

pNC: The NC containing the objects in the sequence.

pUpToDateVecCommonV1: The filter excluding objects from the client's object sequence.

Md5Digest: The digest of the [objectGUID](#) values of the objects in the client's object sequence.

4.1.12.1.3 DRS_MSG_EXISTREPLY

The **DRS_MSG_EXISTREPLY** union defines the response message versions received from the [IDL DRSGetObjectExistence](#) method. Only one version, identified by `pdwOutVersion^ = 1`, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_EXISTREPLY_V1 V1;
} DRS_MSG_EXISTREPLY;
```

V1: The version 1 response.

4.1.12.1.4 DRS_MSG_EXISTREPLY_V1

The **DRS_MSG_EXISTREPLY_V1** structure defines the response message received from the [IDL DRSGetObjectExistence](#) method.

```
typedef struct {
    DWORD dwStatusFlags;
    [range(0,10485760)] DWORD cNumGuids;
    [size_is(cNumGuids)] UUID* rgGuids;
} DRS_MSG_EXISTREPLY_V1;
```

dwStatusFlags: 0 if the digests of the object sequences on the client and server are the same, 1 if they are different.

cNumGuids: The number of items in the `rgGuids` array. Zero if `dwStatusFlags = 0`.

rgGuids: The [objectGUIDs](#) of the objects in the server's object sequence.

4.1.12.2 Method-Specific Abstract Types and Procedures

The following procedure is used in specifying both the client and server behavior of [IDL DRSGetObjectExistence](#).

4.1.12.2.1 GuidSequence

Informative summary of behavior: The *candidate set* of objects is the set of all objects and tombstones in the local replica of NC nc, excluding objects that have not yet replicated to both the client and server replicas of NC nc. The exclusion of objects created too recently is performed using the client-supplied **up-to-date vector** `utd`.

A *cluster* is any subset of the candidate set such that no object in the candidate set outside the cluster has an [objectGUID](#) lying between the [objectGUIDs](#) of any two members of the cluster.

The cluster constructed by `GuidSequence` contains the object in the candidate set with the smallest [objectGUID](#) greater than or equal to `startGUID`. The cluster contains as many objects as possible, but no more than `count`.

Both the client and the server use GuidSequence to compute a cluster, create a sorted sequence of [objectGUIDs](#) of objects in the cluster, and compute a digest of that sequence.

```
procedure GuidSequence(  
    startGUID: GUID,  
    count: ULONG,  
    nc: DSName,  
    utd: UPTODATE_VECTOR_V1_EXT,  
    var s: sequence of DSName,  
    var digest: sequence [0..15] of byte)
```

The procedure GuidSequence returns the following:

1. A sequence *s* of [objectGUIDs](#) from the server's state.
2. An MD5 digest value (contained in the *digest* field) that is derived from the sequence *s*.

The first four parameters determine the result sequence *s*, as follows:

1. Construct the following set of [DSNames](#):

```
select all o subtree-ts-included nc where  
    StampLessThanOrEqualUTD(AttrStamp(o, whenCreated), utd)
```

2. Construct the [GUID](#) sequence *S* that contains the [objectGUIDs](#) of members of the set, sorted into ascending order by [GUID](#) value.
3. Find the smallest integer *i* such that *S*[*i*] >= startGUID. If there is no such *i*, the result sequence *s* is empty, otherwise the result sequence *s* is as follows:

S[*i* .. min(*i*+count, *S*.length)-1]

The result digest is the value of ComputeDigest applied to the result sequence *s*, where ComputeDigest is specified as follows:

```
procedure ComputeDigest(s: sequence of GUID): sequence [0..15] of byte  
md5Context : MD5_CTX  
MD5Init(md5Context)  
    for i := 0 to s.length-1  
        MD5Update(md5Context, s[i], 16)  
    endfor  
MD5Final(md5Context)  
return md5Context.digest
```

4.1.12.3 Client Behavior when Sending the IDL_DRSGetObjectExistence Request

Informative summary of behavior: The client uses [IDL_DRSGetObjectExistence](#) to check the consistency of object existence between its replica of an NC and another replica of the same NC. Checking the consistency of object existence means identifying objects that have replicated to both replicas, and that exist in one replica but not in the other. For the purposes of this method, an object *exists* within an NC replica if it is either an object or a tombstone.

IDL_DRSGetObjectExistence allows the client to perform this checking in *clusters*, as defined in [GuidSequence \(section 4.1.12.2.1\)](#).

The inputs to this checking process on the client are as follows:

```
nc: DSName
utdClient, utdServer: UPTODATE_VECTOR_V1_EXT
guidStart: GUID
count: ULONG
```

nc: The NC containing the cluster that the client will check.

utdClient, utdServer: The up-to-date vectors of the client and server for the NC nc, respectively. The client can obtain utdServer using [IDL_DRSGetReplInfo](#).

guidStart: Lower bound on the smallest [objectGUID](#) in the cluster that the client will check.

count: Upper bound on the number of objects in the cluster that the client will check.

Given these inputs, the client creates the request message to **IDL_DRSGetObjectExistence** as follows: [<20>](#)

```
msgIn: DRS_MSG_EXISTREQ_V1
s: sequence
digest: sequence [0..15] of byte
msgIn.pNC := nc
msgIn.pUpToDateVecCommonV1 := MergeUTD(utdClient, utdServer)
GuidSequence(
    guidStart, count, nc, msgIn.pUpToDateVecCommonV1^, s, digest)
msgIn.guidStart := s[0]
msgIn.length := s.length
msgIn.Md5Digest := digest
pmsgIn^.V1 := msgIn
```

4.1.12.4 Server Behavior of the IDL_DRSGetObjectExistence Method

Informative summary of behavior: The server computes a cluster, an [objectGUID](#) sequence, and a digest in the same manner as the client, but uses the server's NC replica. If the digest computed by the server equals the digest in the client's request, the server returns dwStatusFlags = 1, otherwise the server returns dwStatusFlags = 0 and the [objectGUID](#) sequence.

```
ULONG IDL_DRSGetObjectExistence (
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_EXISTREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_EXISTREPLY *pmsgOut)
msgIn: DRS_MSG_EXISTREQ_V1
nc: DSName
s: sequence of GUID
digest: sequence [0..15] of byte
msgOut: DRS_MSG_EXISTREPLY_V1
if dwInVersion ≠ 0x1 then
    return ERROR_INVALID_PARAMETER
```

```

endif
msgIn := pmsgIn^.V1
nc := msgIn.pNC^
if not MasterReplicaExists(nc) then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
if msgIn.guidStart = NULLGUID then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
if not AccessCheckCAR(nc, DS-Replication-Get-Changes) then
    return ERROR_DS_DRA_ACCESS_DENIED
endif
GuidSequence(msgIn.guidStart, msgIn.cGuids, nc,
    msgIn.pUpToDateVecCommonV1^, s, digest)
if msgIn.Md5Digest = digest then
    msgOut.dwStatusFlags := 1
    msgOut.cNumGuids := 0
    msgOut.rgGuids := null
else if
    msgOut.dwStatusFlags := 0
    msgOut.cNumGuids := s.length
    for i := 0 to s.length - 1
        msgOut.rgGuids[i] := s[i]
    endfor
endif
pmsgOut^.V1 := msgOut
return 0

```

4.1.12.5 Client Behavior when Receiving the IDL_DRSGetObjectExistence Response

Informative summary of behavior: If the server response contains dwStatusFlags = 0, then the client computes the difference between the client and the server sequences and takes whatever action is required.

4.1.13 IDL_DRSGetReplInfo (Opnum 19)

The **IDL_DRSGetReplInfo** method retrieves the replication state of the server.

```

ULONG IDL_DRSGetReplInfo(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_GETREPLINFO_REQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_GETREPLINFO_REPLY* pmsgOut
);

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: A pointer to the request message.

pdwOutVersion: A pointer to the version of the response message.

pmsgOut: A pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.13.1 Method-Specific Concrete Types

4.1.13.1.1 DRS_MSG_GETREPLINFO_REQ

The **DRS_MSG_GETREPLINFO_REQ** union defines the request message versions sent to the [IDL DRSGetReplInfo](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_GETREPLINFO_REQ_V1 V1;
    [case(2)]
        DRS_MSG_GETREPLINFO_REQ_V2 V2;
} DRS_MSG_GETREPLINFO_REQ;
```

V1: Version 1 request (Windows 2000 and later).

V2: Version 2 request (Windows Server 2003 and later). The V2 request structure is a superset of the V1 request structure.

4.1.13.1.2 DRS_MSG_GETREPLINFO_REQ_V1

The **DRS_MSG_GETREPLINFO_REQ_V1** structure defines a version 1 request message sent to the [IDL DRSGetReplInfo](#) method. This request version is supported by Windows 2000 and later releases of Windows.

```
typedef struct {
    DWORD InfoType;
    [string] LPWSTR pszObjectDN;
    UUID uuidSourceDsaObjGuid;
} DRS_MSG_GETREPLINFO_REQ_V1;
```

InfoType: MUST be a [DS_REPL_INFO](#) code.

pszObjectDN: DN of the object on which the operation should be performed. The meaning of this parameter depends on the value of the *InfoType* parameter.

uuidSourceDsaObjGuid: NULL GUID or the DSA GUID of a DC.

4.1.13.1.3 DRS_MSG_GETREPLINFO_REQ_V2

The **DRS_MSG_GETREPLINFO_REQ_V2** structure defines a version 2 request message sent to the [IDL DRSGetReplInfo](#) method. The V2 request structure is a superset of the V1 request structure and is supported by Windows Server 2003 and later versions of Windows.

```
typedef struct {
    DWORD InfoType;
    [string] LPWSTR pszObjectDN;
    UUID uuidSourceDsaObjGuid;
    DWORD ulFlags;
    [string] LPWSTR pszAttributeName;
    [string] LPWSTR pszValueDN;
    DWORD dwEnumerationContext;
} DRS_MSG_GETREPLINFO_REQ_V2;
```

InfoType: MUST be a [DS_REPL_INFO](#) code.

pszObjectDN: DN of the object on which the operation should be performed. The meaning of this parameter depends on the value of the *InfoType* parameter.

uuidSourceDsaObjGuid: NULL GUID or the DSA GUID of a DC.

ulFlags: Zero or more of the following bit flags:

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
M 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	X	X	X	X	X	X	X

X: Unused. MUST be zero and ignored.

MT (DS_REPL_INFO_FLAG_IMPROVE_LINKED_ATTRS): Return attribute stamps for linked values.

pszAttributeName: Null, or the [LDAPDisplayName](#) of a link attribute.

pszValueDN: Null, or the DN of the link value for which to retrieve a stamp.

dwEnumerationContext: The range bound of values to be returned by the server.

4.1.13.1.4 DS_REPL_INFO Codes

DS_REPL_INFO codes indicate the type of replication state information being requested.

Symbolic name	Value
DS_REPL_INFO_NEIGHBORS	0x00000000
DS_REPL_INFO_CURSORS_FOR_NC	0x00000001
DS_REPL_INFO_METADATA_FOR_OBJ	0x00000002
DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES	0x00000003
DS_REPL_INFO_KCC_DSA_LINK_FAILURES	0x00000004
DS_REPL_INFO_PENDING_OPS	0x00000005

Symbolic name	Value
DS_REPL_INFO_METADATA_FOR_ATTR_VALUE	0x00000006
DS_REPL_INFO_CURSORS_2_FOR_NC	0x00000007
DS_REPL_INFO_CURSORS_3_FOR_NC	0x00000008
DS_REPL_INFO_METADATA_2_FOR_OBJ	0x00000009
DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE	0x0000000A
DS_REPL_INFO_SERVER_OUTGOING_CALLS	0xFFFFFFFFFA
DS_REPL_INFO_UPTODATE_VECTOR_V1	0xFFFFFFFFFB
DS_REPL_INFO_CLIENT_CONTEXTS	0xFFFFFFFFFC
DS_REPL_INFO_REPSTO	0xFFFFFFFFFE

4.1.13.1.5 DRS_MSG_GETREPLINFO_REPLY

The **DRS_MSG_GETREPLINFO_REPLY** union defines response messages received from the [IDL DRSGetReplInfo](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(0)]
        DS_REPL_NEIGHBORSW* pNeighbors;
    [case(1)]
        DS_REPL_CURSORS* pCursors;
    [case(2)]
        DS_REPL_OBJ_META_DATA* pObjMeta-data;
    [case(3)]
        DS_REPL_KCC_DSA_FAILURESW* pConnectFailures;
    [case(4)]
        DS_REPL_KCC_DSA_FAILURESW* pLinkFailures;
    [case(5)]
        DS_REPL_PENDING_OPSW* pPendingOps;
    [case(6)]
        DS_REPL_ATTR_VALUE_META_DATA* pAttrValueMeta-data;
    [case(7)]
        DS_REPL_CURSORS_2* pCursors2;
    [case(8)]
        DS_REPL_CURSORS_3W* pCursors3;
    [case(9)]
        DS_REPL_OBJ_META_DATA_2* pObjMeta-data2;
    [case(10)]
        DS_REPL_ATTR_VALUE_META_DATA_2* pAttrValueMeta-data2;
    [case(0xFFFFFFFFFA)]
        DS_REPL_SERVER_OUTGOING_CALLS* pServerOutgoingCalls;
    [case(0xFFFFFFFFFB)]
        UPTODATE_VECTOR_V1_EXT* pUpToDateVec;
    [case(0xFFFFFFFFFC)]
        DS_REPL_CLIENT_CONTEXTS* pClientContexts;
    [case(0xFFFFFFFFFE)]
        DS_REPL_NEIGHBORSW* pRepsTo;
```

```
} DRS_MSG_GETREPLINFO_REPLY;
```

pNeighbors: The neighbor information.

pCursors: The cursors for an NC replica.

pObjMetaData: The attribute stamps.

pConnectFailures: The connection failure data.

pLinkFailures: The link failure data.

pPendingOps: The pending operations in the replication queue.

pAttrValueMetaData: The link value stamps.

pCursors2: The cursors for an NC replica.

pCursors3: The cursors for an NC replica.

pObjMetaData2: The attribute stamps.

pAttrValueMetaData2: The link value stamps.

pServerOutgoingCalls: The outstanding requests from this DC to other DCs.

pUpToDateVec: The cursors for an NC replica.

pClientContexts: The active RPC client connections.

pRepsTo: The neighbor information.

4.1.13.1.6 DS_REPL_NEIGHBORSW

The **DS_REPL_NEIGHBORSW** structure defines a set of replication neighbors. This structure is a concrete representation of a sequence of [RepsFrom](#) or [RepsTo](#) values.

```
typedef struct {
    DWORD cNumNeighbors;
    DWORD dwReserved;
    [size_is(cNumNeighbors)] DS_REPL_NEIGHBORW rgNeighbor[];
} DS_REPL_NEIGHBORSW;
```

cNumNeighbors: The count of items in the rgNeighbor array.

dwReserved: MUST be 0.

rgNeighbor: The set of replication neighbors.

4.1.13.1.7 DS_REPL_NEIGHBORW

The **DS_REPL_NEIGHBORW** structure defines a replication neighbor. This structure is a concrete representation of a [RepsFrom](#) or [RepsTo](#) value.

```
typedef struct {
    [string] LPWSTR pszNamingContext;
    [string] LPWSTR pszSourceDsaDN;
    [string] LPWSTR pszSourceDsaAddress;
    [string] LPWSTR pszAsyncIntersiteTransportDN;
    DWORD dwReplicaFlags;
    DWORD dwReserved;
    UUID uuidNamingContextObjGuid;
    UUID uuidSourceDsaObjGuid;
    UUID uuidSourceDsaInvocationID;
    UUID uuidAsyncIntersiteTransportObjGuid;
    USN usnLastObjChangeSynced;
    USN usnAttributeFilter;
    FILETIME ftimeLastSyncSuccess;
    FILETIME ftimeLastSyncAttempt;
    DWORD dwLastSyncResult;
    DWORD cNumConsecutiveSyncFailures;
} DS_REPL_NEIGHBOR;
```

pszNamingContext: The NC root of the NC replica.

pszSourceDsaDN: The DN of the server DC [nTDSDSA](#) object.

pszSourceDsaAddress: The [NetworkAddress](#) of the server DC.

pszAsyncIntersiteTransportDN: The DN of the [interSiteTransport](#) object corresponding to the transport used to communicate with the server DC.

dwReplicaFlags: The [DRS_OPTIONS](#) flags.

dwReserved: MUST be 0.

uuidNamingContextObjGuid: The [objectGUID](#) of the NC root.

uuidSourceDsaObjGuid: The DSA GUID of the server DC.

uuidSourceDsaInvocationID: The invocation ID associated with the server DC.

uuidAsyncIntersiteTransportObjGuid: The [objectGUID](#) of the [interSiteTransport](#) object corresponding to the transport used to communicate with the server DC.

usnLastObjChangeSynced: An implementation-specific value.

usnAttributeFilter: An implementation-specific value.

ftimeLastSyncSuccess: The time of the last successful replication from the server DC.

ftimeLastSyncAttempt: The time of the last attempt to replicate from the server DC.

dwLastSyncResult: 0, or the Windows error code resulting from the last sync attempt.

cNumConsecutiveSyncFailures: The number of consecutive failures to replicate from the server DC.

4.1.13.1.8 DS_REPL_CURSORS

The **DS_REPL_CURSORS** structure defines a set of replication cursors for a given NC replica. This structure is a concrete representation of a sequence of [ReplUpToDateVector](#) values.

```
typedef struct {
    DWORD cNumCursors;
    DWORD dwReserved;
    [size_is(cNumCursors)] DS_REPL_CURSOR rgCursor[];
} DS_REPL_CURSORS;
```

cNumCursors: The count of items in the rgCursor array.

dwReserved: MUST be 0.

rgCursor: The set of replication cursors.

4.1.13.1.9 DS_REPL_CURSOR

The **DS_REPL_CURSOR** structure defines a replication cursor for a given NC replica. This structure is a concrete representation of a [ReplUpToDateVector](#) value.

```
typedef struct {
    UUID uuidSourceDsaInvocationID;
    USN usnAttributeFilter;
} DS_REPL_CURSOR;
```

uuidSourceDsaInvocationID: The invocation ID of a DC.

usnAttributeFilter: The USN at which an update was applied on the DC.

4.1.13.1.10 DS_REPL_CURSORS_2

The **DS_REPL_CURSORS_2** structure defines a set of replication cursors for a given NC replica. This structure is a concrete representation of a sequence of [ReplUpToDateVector](#) values; it is a superset of [DS_REPL_CURSORS](#).

```
typedef struct {
    DWORD cNumCursors;
    DWORD dwEnumerationContext;
    [size_is(cNumCursors)] DS_REPL_CURSOR_2 rgCursor[];
} DS_REPL_CURSORS_2;
```

cNumCursors: The count of items in the rgCursor array.

dwEnumerationContext: 0xFFFFFFFF, or the range bound of the results.

rgCursor: A set of replication cursors.

4.1.13.1.11 DS_REPL_CURSOR_2

The **DS_REPL_CURSOR_2** structure defines a replication cursor for a given NC replica. This structure is a concrete representation of a [ReplUpToDateVector](#) value; it is a superset of [DS_REPL_CURSOR](#).

```
typedef struct {
    UUID uuidSourceDsaInvocationID;
    USN usnAttributeFilter;
    FILETIME ftimeLastSyncSuccess;
} DS_REPL_CURSOR_2;
```

uuidSourceDsaInvocationID: The invocation ID of a DC.

usnAttributeFilter: The USN at which an update was applied on the DC.

ftimeLastSyncSuccess: The time at which the last successful replication occurred from the DC identified by uuidDsa. Used for **replication latency** reporting only.

4.1.13.1.12 DS_REPL_CURSORS_3W

The **DS_REPL_CURSORS_3W** structure defines a replication cursor for a given NC replica. This structure is a concrete representation of a sequence of [ReplUpToDateVector](#) values; it is a superset of [DS_REPL_CURSORS_2](#).

```
typedef struct {
    DWORD cNumCursors;
    DWORD dwEnumerationContext;
    [size_is(cNumCursors)] DS_REPL_CURSOR_3W rgCursor[];
} DS_REPL_CURSORS_3W;
```

cNumCursors: The count of items in the rgCursor array.

dwEnumerationContext: 0xFFFFFFFF, or the range bound of the results.

rgCursor: A set of replication cursors.

4.1.13.1.13 DS_REPL_CURSOR_3W

The **DS_REPL_CURSOR_3W** structure defines a replication cursor for a given NC replica. This structure is a concrete representation of a [ReplUpToDateVector](#) value; it is a superset of [DS_REPL_CURSOR_2](#).

```
typedef struct {
    UUID uuidSourceDsaInvocationID;
    USN usnAttributeFilter;
    FILETIME ftimeLastSyncSuccess;
    [string] LPWSTR pszSourceDsaDN;
} DS_REPL_CURSOR_3W;
```

uuidSourceDsaInvocationID: The invocation ID of a DC.

usnAttributeFilter: The USN at which an update was applied on the DC.

ftimeLastSyncSuccess: The time at which the last successful replication occurred from the DC identified by uuidDsa. Used for replication latency reporting only.

pszSourceDsaDN: The DN of the [nTDSDSA](#) object with an [invocationId](#) of uuidSourceDsaInvocationID.

4.1.13.1.14 DS_REPL_OBJ_META_DATA

The **DS_REPL_OBJ_META_DATA** structure defines a set of attribute stamps for a given object. This structure is a concrete representation of the sequence of [AttributeStamp](#) values for all attributes of a given object.

```
typedef struct {
    DWORD cNumEntries;
    DWORD dwReserved;
    [size_is(cNumEntries)] DS_REPL_ATTR_META_DATA rgMetaData[];
} DS_REPL_OBJ_META_DATA;
```

cNumEntries: The count of items in the rgMetaData array.

dwReserved: MUST be 0.

rgMetaData: A set of attribute stamps.

4.1.13.1.15 DS_REPL_ATTR_META_DATA

The **DS_REPL_ATTR_META_DATA** structure defines an attribute stamp for a given object. This structure is a concrete representation of an [AttributeStamp](#).

```
typedef struct {
    [string] LPWSTR pszAttributeName;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
} DS_REPL_ATTR_META_DATA;
```

pszAttributeName: The [LDAPDisplayName](#) of the attribute to which the stamp corresponds.

dwVersion: The stamp version.

ftimeLastOriginatingChange: The date and time at which the last originating update was made.

uuidLastOriginatingDsaInvocationID: The invocation ID of the DC that performed the last originating update.

usnOriginatingChange: The USN assigned to the last originating update by the DC that performed it.

usnLocalChange: The implementation-specific value.

4.1.13.1.16 DS_REPL_OBJ_META_DATA_2

The **DS_REPL_OBJ_META_DATA_2** structure defines a set of attribute stamps for a given object. This structure is a concrete representation of the sequence of [AttributeStamp](#) values for all attributes of a given object; it is a superset of [DS_REPL_OBJ_META_DATA](#).

```
typedef struct {
    DWORD cNumEntries;
    DWORD dwReserved;
    [size_is(cNumEntries)] DS_REPL_ATTR_META_DATA_2 rgMetaData[];
} DS_REPL_OBJ_META_DATA_2;
```

cNumEntries: The count of items in the rgMetaData array.

dwReserved: MUST be 0.

rgMetaData: A set of attribute stamps.

4.1.13.1.17 DS_REPL_ATTR_META_DATA_2

The **DS_REPL_ATTR_META_DATA_2** structure defines an attribute stamp for a given object. This structure is a concrete representation of an [AttributeStamp](#); it is a superset of [DS_REPL_ATTR_META_DATA](#).

```
typedef struct {
    [string] LPWSTR pszAttributeName;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
    [string] LPWSTR pszLastOriginatingDsaDN;
} DS_REPL_ATTR_META_DATA_2;
```

pszAttributeName: The [LDAPDisplayName](#) of the attribute to which the stamp corresponds.

dwVersion: The stamp version.

ftimeLastOriginatingChange: The date and time at which the last originating update was made.

uuidLastOriginatingDsaInvocationID: The invocation ID of the DC that performed the last originating update.

usnOriginatingChange: The USN assigned to the last originating update by the DC that performed it.

usnLocalChange: The implementation-specific value.

pszLastOriginatingDsaDN: The DN of the [nTDSDSA](#) object with an [invocationId](#) of `uuidLastOriginatingDsaInvocationID`.

4.1.13.1.18 DS_REPL_KCC_DSA_FAILURESW

The **DS_REPL_KCC_DSA_FAILURESW** structure defines a set of DCs that are in an error state with respect to replication. This structure is a concrete representation of [KCCFailedConnections](#) and [KCCFailedLinks](#).

```
typedef struct {
    DWORD cNumEntries;
    DWORD dwReserved;
    [size_is(cNumEntries)] DS_REPL_KCC_DSA_FAILUREW rgDsaFailure[];
} DS_REPL_KCC_DSA_FAILURESW;
```

cNumEntries: The count of items in the `rgDsaFailure` array.

dwReserved: MUST be 0.

rgDsaFailure: An array of [DS_REPL_KCC_DSA_FAILUREW](#).

4.1.13.1.19 DS_REPL_KCC_DSA_FAILUREW

The **DS_REPL_KCC_DSA_FAILUREW** structure defines a DC that is in a replication error state. This structure is a concrete representation of a tuple in a [KCCFailedConnections](#) or [KCCFailedLinks](#) sequence.

```
typedef struct {
    [string] LPWSTR pszDsaDN;
    UUID uuidDsaObjGuid;
    FILETIME ftimeFirstFailure;
    DWORD cNumFailures;
    DWORD dwLastResult;
} DS_REPL_KCC_DSA_FAILUREW;
```

pszDsaDN: The DN of the [nTDSDSA](#) object corresponding to the DC.

uuidDsaObjGuid: The DSA GUID of the DC.

ftimeFirstFailure: The date and/or time at which the DC entered an error state.

cNumFailures: The number of errors that have occurred.

dwLastResult: The Windows error code for the last error.

4.1.13.1.20 DS_REPL_PENDING_OPSW

The **DS_REPL_PENDING_OPSW** structure defines a sequence of replication operations to be processed by a DC. This structure is a concrete representation of [ReplicationQueue](#).

```
typedef struct {
    FILETIME ftimeCurrentOpStarted;
    DWORD cNumPendingOps;
    [size_is(cNumPendingOps)] DS_REPL_OPW rgPendingOp[];
} DS_REPL_PENDING_OPSW;
```

ftimeCurrentOpStarted: The time when the current operation started.

cNumPendingOps: The count of items in the rgPendingOp array.

rgPendingOp: The sequence of replication operations to be performed.

4.1.13.1.21 DS_REPL_OPW

The **DS_REPL_OPW** structure defines a replication operation to be processed by a DC. This structure is a concrete representation of a tuple in a [ReplicationQueue](#) sequence.

```
typedef struct {
    FILETIME ftimeEnqueued;
    ULONG ulSerialNumber;
    ULONG ulPriority;
    DS_REPL_OP_TYPE OpType;
    ULONG ulOptions;
    [string] LPWSTR pszNamingContext;
    [string] LPWSTR pszDsaDN;
    [string] LPWSTR pszDsaAddress;
    UUID uuidNamingContextObjGuid;
    UUID uuidDsaObjGuid;
} DS_REPL_OPW;
```

ftimeEnqueued: The date and/or time at which the operation was requested.

ulSerialNumber: The unique ID associated with the operation.

ulPriority: The priority of the operation.

OpType: The type of operation.

ulOptions: The [DRS_OPTIONS](#) flags.

pszNamingContext: The NC root of the relevant NC replica.

pszDsaDN: The DN of the relevant DC's [nTDSDSA](#) object.

pszDsaAddress: The [NetworkAddress](#) of the relevant DC.

uuidNamingContextObjGuid: The [objectGUID](#) of the NC root of the relevant NC replica.

uuidDsaObjGuid: The DSA GUID of the DC.

4.1.13.1.22 DS_REPL_ATTR_VALUE_META_DATA

The **DS_REPL_ATTR_VALUE_META_DATA** structure defines a sequence of link value stamps. This structure is a concrete representation of a sequence of [LinkValueStamp](#) values.

```
typedef struct {
    DWORD cNumEntries;
    DWORD dwEnumerationContext;
    [size_is(cNumEntries)] DS_REPL_VALUE_META_DATA rgMetaData[];
} DS_REPL_ATTR_VALUE_META_DATA;
```

cNumEntries: The count of items in rgMetaData array.

dwEnumerationContext: 0xFFFFFFFF, or the range bound of the results.

rgMetaData: The sequence of link value stamps.

4.1.13.1.23 DS_REPL_VALUE_META_DATA

The **DS_REPL_VALUE_META_DATA** structure defines a link value stamp. This structure is a concrete representation of a [LinkValueStamp](#).

```
typedef struct {
    [string] LPWSTR pszAttributeName;
    [string] LPWSTR pszObjectDn;
    DWORD cbData;
    [size_is(cbData), ptr] BYTE* pbData;
    FILETIME ftimeDeleted;
    FILETIME ftimeCreated;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
} DS_REPL_VALUE_META_DATA;
```

pszAttributeName: The [LDAPDisplayName](#) of the attribute.

pszObjectDn: The DN of the object.

cbData: The size, in bytes, of the pbData array.

pbData: Null, or data associated with the attribute value if the attribute is of syntax Object(DN-Binary) or Object(DN-String).

ftimeDeleted: The date and time at which the last replicated update was made that deleted the value, or 0 if the value is not currently deleted.

ftimeCreated: The date and time at which the first originating update was made.

dwVersion: The stamp version.

ftimeLastOriginatingChange: The date and time at which the last originating update was made.

uuidLastOriginatingDsaInvocationID: The invocation ID of the DC that performed the last originating update.

usnOriginatingChange: The USN assigned to the last originating update by the DC that performed the update.

usnLocalChange: An implementation-specific value.

4.1.13.1.24 DS_REPL_ATTR_VALUE_META_DATA_2

The **DS_REPL_ATTR_VALUE_META_DATA_2** structure defines a sequence of link value stamps. This structure is a concrete representation of a sequence of [LinkValueStamp](#) values; it is a superset of [DS_REPL_ATTR_VALUE_META_DATA](#).

```
typedef struct {
    DWORD cNumEntries;
    DWORD dwEnumerationContext;
    [size_is(cNumEntries)] DS_REPL_VALUE_META_DATA_2 rgMetaData[];
} DS_REPL_ATTR_VALUE_META_DATA_2;
```

cNumEntries: The count of items in the rgMetaData array.

dwEnumerationContext: 0xFFFFFFFF, or the range bound of the results.

rgMetaData: The sequence of link value stamps.

4.1.13.1.25 DS_REPL_VALUE_META_DATA_2

The **DS_REPL_VALUE_META_DATA_2** structure defines a link value stamp. This structure is a concrete representation of [LinkValueStamp](#); it is a superset of [DS_REPL_VALUE_META_DATA](#).

```
typedef struct {
    [string] LPWSTR pszAttributeName;
    [string] LPWSTR pszObjectDn;
    DWORD cbData;
    [size_is(cbData), ptr] BYTE* pbData;
    FILETIME ftimeDeleted;
    FILETIME ftimeCreated;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
    [string] LPWSTR pszLastOriginatingDsaDN;
} DS_REPL_VALUE_META_DATA_2;
```

pszAttributeName: The [LDAPDisplayName](#) of the attribute.

pszObjectDn: The DN of the object.

cbData: The size, in bytes, of the pbData array.

pbData: Null, or data associated with the attribute value if the attribute is of syntax Object(DN-Binary) or Object(DN-String).

ftimeDeleted: The date and/or time at which the last replicated update was made that deleted the value, or 0 if the value is not currently deleted.

ftimeCreated: The date and/or time at which the first originating update was made.

dwVersion: The stamp version.

ftimeLastOriginatingChange: The date and/or time at which the last originating update was made.

uuidLastOriginatingDsaInvocationID: The invocation ID of the DC that performed the last originating update.

usnOriginatingChange: The USN assigned to the last originating update by the DC that performed the update.

usnLocalChange: An implementation-specific value.

pszLastOriginatingDsaDN: The DN of the [nTDSDSA](#) object with an [invocationId](#) of `uuidLastOriginatingDsaInvocationID`.

4.1.13.1.26 DS_REPL_CLIENT_CONTEXTS

The **DS_REPL_CLIENT_CONTEXTS** structure defines a set of active RPC client connections. This structure is a concrete representation of [RPCClientContexts](#).

```
typedef struct {
    [range(0,10000)] DWORD cNumContexts;
    DWORD dwReserved;
    [size_is(cNumContexts)] DS_REPL_CLIENT_CONTEXT rgContext[];
} DS_REPL_CLIENT_CONTEXTS;
```

cNumContexts: The count of items in the rgContext array.

dwReserved: MUST be 0.

rgContext: The set of active RPC client connections.

4.1.13.1.27 DS_REPL_CLIENT_CONTEXT

The **DS_REPL_CLIENT_CONTEXT** structure defines an active RPC client connection. This structure is a concrete representation of a tuple in an [RPCClientContexts](#) sequence.

```
typedef struct {
    ULONGLONG hCtx;
    LONG lReferenceCount;
    BOOL fIsBound;
    UUID uuidClient;
    DSTIME timeLastUsed;
    ULONG IPAddr;
```

```

    int pid;
} DS_REPL_CLIENT_CONTEXT;

```

hCtx: The unique ID of the client context.

lReferenceCount: The number of references to the context.

fIsBound: True if, and only if, the context has not yet been closed by the [IDL_DRSUnbind](#) method.

uuidClient: Zeros, or the value pointed to by the *puuidClientDsa* parameter to [IDL_DRSBind](#).

timeLastUsed: The date and/or time at which this context was last used in an RPC method call.

IPAddr: The IPv4 address of the client. If the client is connected with IPv6, this field contains zero.

pid: 0, or the process ID passed specified by the client in the *pextClient* parameter to [IDL_DRSBind](#).

4.1.13.1.28 DS_REPL_SERVER_OUTGOING_CALLS

The **DS_REPL_SERVER_OUTGOING_CALLS** structure defines a set of outstanding requests from this DC to other DCs. This structure is a concrete representation of [RPCOutgoingContexts](#).

```

typedef struct {
    [range(0,256)] DWORD cNumCalls;
    DWORD dwReserved;
    [size_is(cNumCalls)] DS_REPL_SERVER_OUTGOING_CALL rgCall[];
} DS_REPL_SERVER_OUTGOING_CALLS;

```

cNumCalls: The count of items in the *rgCall* array.

dwReserved: MUST be 0.

rgCall: The set of outstanding requests from this DC to other DCs.

4.1.13.1.29 DS_REPL_SERVER_OUTGOING_CALL

The **DS_REPL_SERVER_OUTGOING_CALL** structure defines an outstanding request from this DC to another DC. This structure is a concrete representation of a tuple from an [RPCOutgoingContexts](#) sequence.

```

typedef struct {
    [string] LPWSTR pszServerName;
    BOOL fIsHandleBound;
    BOOL fIsHandleFromCache;
    BOOL fIsHandleInCache;
    DWORD dwThreadId;
    DWORD dwBindingTimeoutMins;
    DSTIME dstimeCreated;
    DWORD dwCallType;
} DS_REPL_SERVER_OUTGOING_CALL;

```

```
} DS_REPL_SERVER_OUTGOING_CALL;
```

pszServerName: The [NetworkAddress](#) of the server.

fIsHandleBound: True if, and only if, the [IDL_DRSBind](#) method has completed and the [IDL_DRSUnbind](#) method has not yet been called.

fIsHandleFromCache: True if, and only if, the context handle used was retrieved from the cache.

fIsHandleInCache: True if, and only if, the context handle is still in the cache.

dwThreadId: The ID of the thread that is using the context.

dwBindingTimeoutMins: If the context is set to be canceled, then the time-out, in minutes.

dstimeCreated: The date and/or time when the context was created.

dwCallType: The call that the client is waiting on. MUST be one of the values in the following table.

Value	Meaning
2	IDL_DRSBind
3	IDL_DRSUnbind
4	IDL_DRSReplicaSync
5	IDL_DRSGetNCChanges
6	IDL_DRSUpdateRefs
7	IDL_DRSReplicaAdd
8	IDL_DRSReplicaDel
9	IDL_DRSVerifyNames
10	IDL_DRSGetMemberships
11	IDL_DRSInterDomainMove
12	IDL_DRSGetNT4ChangeLog
13	IDL_DRSCrackNames
14	IDL_DRSAddEntry
15	IDL_DRSGetMemberships2
16	IDL_DRSGetObjectExistence
17	IDL_DRSGetReplInfo

4.1.13.2 Method-Specific Abstract Types and Procedures

4.1.13.2.1 GetDNFromInvocationID

```
procedure GetDNFromInvocationID(invocationID: GUID): DN
```

Returns the DN of the [nTDSDSA](#) object that has the specified invocation ID.

4.1.13.2.2 GetDNFromObjectGuid

```
procedure GetDNFromObjectGuid(guid: GUID): DN
```

Returns the DN of the object with the specified object GUID. This is represented by the following expression:

```
obj := select one o from all where (o!objectGUID = guid)
return obj.dn
```

4.1.13.2.3 GetNCs

```
procedure GetNCs(): set of DSName
```

Returns a set containing the [DSNames](#) of all NCs hosted by this server.

4.1.13.3 Server Behavior of the IDL_DRSGetReplInfo Method

Informative summary of behavior: This method retrieves the replication state information of a DC. Based on the value of the **InfoType** field in the request message, different information is returned, which is summarized as follows:

- DS_REPL_INFO_NEIGHBORS: The replication state data for each NC and source server pair, for all NC replicas hosted by this DC.
- DS_REPL_INFO_REPSTO: The replication state data for each NC and destination server (which is notified of changes) pair, for all NC replicas hosted by this DC.
- DS_REPL_INFO_CURSORS_FOR_NC
- DS_REPL_INFO_CURSORS_2_FOR_NC
- DS_REPL_INFO_CURSORS_3_FOR_NC: The replication state for the NC replica of a given NC.
- DS_REPL_INFO_METADATA_FOR_OBJ
- DS_REPL_INFO_METADATA_2_FOR_OBJ: Stamps for all the replicated attributes of the given object.
- DS_REPL_INFO_METADATA_FOR_ATTR_VALUE
- DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE: Stamps for a specific link attribute of the given object.

- DS_REPL_INFO_KCC_DSA_FAILURES: Replication state data regarding connection failures with inbound replication partners.
- DS_REPL_INFO_KCC_LINK_FAILURES: Replication state data regarding link failures with inbound replication partners.
- DS_REPL_INFO_PENDING_OPS: Replication tasks that are currently executing or that are queued to execute.
- DS_REPL_INFO_CLIENT_CONTEXTS: A list of all outstanding RPC client contexts.
- DS_REPL_INFO_SERVER_OUTGOING_CALLS: A list of all outstanding RPC server call contexts.

```

ULONG
IDL_DRSGetReplInfo(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_GETREPLINFO_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_GETREPLINFO_REPLY *pmsgOut)

msgIn: DRS_MSG_GETREPLINFO_REQ_V2
infoType: DWORD
accessGranted: boolean
infoTypeValid: boolean
defaultNC: DSName
object: DSName
enumerationContext: DWORD
baseIndex: DWORD
endIndex: DWORD
ncs: set of DSName
nc: DSName
i, j: DWORD
r: RepsFrom
s: RepsTo
pNeighbor: ADDRESS OF DS_REPL_NEIGHBORW
utd: sequence of ReplUpToDateVector
pCursor: ADDRESS OF DS_REPL_CURSOR
pCursor2: ADDRESS OF DS_REPL_CURSOR_2
pCursor3: ADDRESS OF DS_REPL_CURSOR_3W
a: ATTRTYP
attr: ATTRTYP
attrs: set of ATTRTYP
attrsSeq: sequence of ATTRTYP
s: AttributeStamp
pObjMetaData: ADDRESS OF DS_REPL_OBJ_META_DATA
pObjMetaData2: ADDRESS OF DS_REPL_OBJ_META_DATA_2
values: set of attribute value
valuesSeq: sequence of attribute value
ls: LinkValueStamp
pAttrValueMetaData: ADDRESS OF DS_REPL_ATTR_VALUE_META_DATA
pAttrValueMetaData2: ADDRESS OF DS_REPL_ATTR_VALUE_META_DATA_2
dc: DC
pFailedConnection: ADDRESS OF DS_REPL_KCC_DSA_FAILUREW
pFailedLink: ADDRESS OF DS_REPL_KCC_DSA_FAILUREW
pPendingOp: ADDRESS OF DS_REPL_OPW

```

```

pClientContext: ADDRESS OF DS_REPL_CLIENT_CONTEXT
pOutgoingContext: ADDRESS OF DS_REPL_SERVER_OUTGOING_CALL

/* Validate the version of the request message */
if (dwInVersion # 1 and dwInVersion # 2) then
    return ERROR_REVISION_MISMATCH
endif

if dwInVersion = 1 then
    msgIn := pmsgIn^.V1
else
    msgIn := pmsgIn^.V2
endif

/* For some of the request types, paging is supported. For these
 * cases, we need a starting index into the result set based on
 * what has already been returned in a previous call. Only version 2
 * request messages provide a mechanism for the client to supply the
 * context information from a previous call. */
if dwInVersion = 1 then
    baseIndex := 0
else
    baseIndex := msgIn.dwEnumerationContext
endif

/* Perform the necessary access checks. */
defaultNC := DefaultNC()
fAccessGranted := false
if (infoType = DS_REPL_INFO_NEIGHBORS and object # null) then
    infoTypeValid := true
    fAccessGranted :=
        AccessCheckAttr(object, repsFrom, RIGHT_DS_READ_PROPERTY) or
        AccessCheckCAR(object, DS-Replication-Manage-Topology) or
        AccessCheckCAR(object, DS-Replication-Monitor-Topology)
endif
if (infoType = DS_REPL_INFO_NEIGHBORS and object = null) then
    infoTypeValid := true
    fAccessGranted :=
        AccessCheckAttr(defaultNC, repsFrom, RIGHT_DS_READ_PROPERTY) or
        AccessCheckCAR(defaultNC, DS-Replication-Manage-Topology) or
        AccessCheckCAR(defaultNC, DS-Replication-Monitor-Topology)
endif
if (infoType = DS_REPL_INFO_REPSTO and object # null) then
    infoTypeValid := true
    fAccessGranted :=
        AccessCheckAttr(object, repsTo, RIGHT_DS_READ_PROPERTY) or
        AccessCheckCAR(object, DS-Replication-Manage-Topology) or
        AccessCheckCAR(object, DS-Replication-Monitor-Topology)
endif
if (infoType = DS_REPL_INFO_REPSTO and object = null) then
    infoTypeValid := true
    fAccessGranted :=
        AccessCheckAttr(defaultNC, repsTo, RIGHT_DS_READ_PROPERTY) or
        AccessCheckCAR(defaultNC, DS-Replication-Manage-Topology) or
        AccessCheckCAR(defaultNC, DS-Replication-Monitor-Topology)
endif
if infoType in {DS_REPL_INFO_CURSORS_FOR_NC,
                DS_REPL_INFO_CURSORS_2_FOR_NC,

```

```

                DS_REPL_INFO_CURSORS_3_FOR_NC,
                DS_REPL_INFO_UPTODATE_VECTOR_V1} then
infoTypeValid := true
fAccessGranted :=
    AccessCheckAttr(
        object, replUpToDateVector, RIGHT_DS_READ_PROPERTY) or
    AccessCheckCAR(object, DS-Replication-Manage-Topology) or
    AccessCheckCAR(object, DS-Replication-Monitor-Topology)
endif
if infoType in {DS_REPL_INFO_METADATA_FOR_OBJ,
                DS_REPL_INFO_METADATA_2_FOR_OBJ,
                DS_REPL_INFO_METADATA_FOR_ATTR_VALUE,
                DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE} then
infoTypeValid := true
fAccessGranted :=
    AccessCheckAttr(object,
                    replPropertyMetaData,
                    RIGHT_DS_READ_PROPERTY) or
    AccessCheckCAR(object, DS-Replication-Manage-Topology) or
    AccessCheckCAR(object, DS-Replication-Monitor-Topology)
endif
if infoType in {DS_REPL_INFO_PENDING_OPS,
                DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES,
                DS_REPL_INFO_KCC_DSA_LINK_FAILURES,
                DS_REPL_INFO_CLIENT_CONTEXTS,
                DS_REPL_INFO_SERVER_OUTGOING_CALLS} then
infoTypeValid := true
fAccessGranted :=
    AccessCheckCAR(defaultNC, DS-Replication-Manage-Topology) or
    AccessCheckCAR(defaultNC, DS-Replication-Monitor-Topology)
endif

if not infoTypeValid then
    return ERROR_INVALID_PARAMETER
endif

if not accessGranted then
    return ERROR_DS_DRA_ACCESS_DENIED
endif

/* Based on the type of information requested, the corresponding
 * information is retrieved and the response message constructed */

/* DS_REPL_INFO_NEIGHBORS/DS_REPL_INFO_REPSTO */
if infoType in {DS_REPL_INFO_NEIGHBORS, DS_REPL_INFO_REPSTO}
/* If an object is specified, it must be an NC root. */
nc := object

if nc ≠ null and not FullReplicaExists(nc) and not
    PartialGCReplicaExists(nc) then
    return ERROR_DS_DRA_BAD_NC
endif

if baseIndex = 0xffffffff then
/* No more data is available. */
    return ERROR_NO_MORE_ITEMS
endif

```

```

if nc ≠ null then
    ncs := {nc}
else
    ncs := GetNCs()
endif

if infoType = DS_REPL_INFO_NEIGHBORS then
    i := 0
    j := 0
    foreach nc in ncs
        foreach r in nc!repsFrom
            /* The ordering of ncs hosted by the server and the values of
             * repsFrom for each nc is arbitrary but consistent from call
             * to call on a server. */

            /* If a source server GUID is specified, only information for
             * that server is returned. */
            If (msgIn.uuidSourceDsaGuid = NULL GUID or
                msgIn.uuidSourceDsaGuid = r.uuidDsa) then
                if i >= baseIndex then
                    pNeighbor := ADR(pmsgOut^.pNeighbors^.rgNeighbor[j])
                    pNeighbor^.pszSourceDsaAddress := r.naDsa
                    pNeighbor^.dwReplicaFlags := r.options
                    pNeighbor^.timeLastAttempt := r.timeLastAttempt
                    pNeighbor^.uuidSourceDsaObjGuid := r.uuidDsa
                    pNeighbor^.pszSourceDsaDN :=
                        GetDNFromObjectGuid(r.uuidDsa)
                    pNeighbor^.pszNamingContext := nc!distinguishedName
                    pNeighbor^.uuidNamingContextObjGuid := nc!objectGUID
                    pNeighbor^.pszAsyncIntersiteTransportDN :=
                        GetDNFromObjectGuid(r.uuidTransportObj)
                    pNeighbor^.uuidSourceDsaInvocationID := r.uuidInvocId
                    pNeighbor^.uuidAsyncIntersiteTransportObjGuid :=
                        r.uuidTransportObj
                    pNeighbor^.usnLastObjChangeSynced :=
                        r.usnVec.usnHighObjUpdate
                    pNeighbor^.usnAttributeFilter :=
                        r.usnVec.usnHighPropUpdate
                    pNeighbor^.fTimeLastSyncSuccess := r.timeLastSuccess
                    pNeighbor^.dwLastSyncResult := r.ulResultLastAttempt
                    pNeighbor^.cNumConsecutiveSyncFailures :=
                        r.cbConsecutiveFailures
                    j := j + 1
                endif
                i := i + 1
            endif
        endfor
    endfor
    pmsgOut^.pNeighbors^.cNumNeighbors := j
else
    /* DS_REPL_INFO_REPSTO case. */
    i := 0
    j := 0
    foreach nc in ncs
        foreach r in nc!repsTo
            /* The ordering of ncs hosted by the server and the values of
             * repsTo for each nc is arbitrary but consistent from call
             * to call on a server. */

```

```

        if i >= baseIndex then
            pNeighbor := ADR(pmsgOut^.pNeighbors^.rgNeighbor[j])
            pNeighbor^.pszSourceDsaAddress := r.naDsa
            pNeighbor^.dwReplicaFlags := r.options
            pNeighbor^.timeLastAttempt := r.timeLastAttempt
            pNeighbor^.uuidSourceDsaObjGuid := r.uuidDsa
            pNeighbor^.pszSourceDsaDN := GetDNFromObjectGuid(r.uuidDsa)
            pNeighbor^.pszNamingContext := nc!distinguishedName
            pNeighbor^.uuidNamingContextObjGuid := nc!objectGUID
            j := j + 1
        endif
        i := i + 1
    endfor
endfor
pmsgOut^.pRepsTo^.cNumNeighbors := j
endif
endif

/* DS_REPL_INFO_METADATA_FOR_OBJ/DS_REPL_INFO_METADATA_2_FOR_OBJ */
if infoType in {DS_REPL_INFO_METADATA_FOR_OBJ,
    DS_REPL_INFO_METADATA_2_FOR_OBJ} then
    /* Basic parameter validation */
    if object = null or not ObjExists(object) then
        return ERROR_INVALID_PARAMETER
    endif

    if baseIndex = 0xffffffff then
        /* No more data is available. */
        return ERROR_NO_MORE_ITEMS
    endif

    /* Enumerate all the replicated attributes of the object */
    attrSeq := select all a from Replicated Attributes of object
    i := 0
    j := 0
    while (i < attrSeq.length)
        attr := attrSeq[i]
        s := AttrStamp(object, attr)

        if (attr in Link Attributes of object and
            dwInVersion = 2 and
            DS_REPL_INFO_FLAG_IMPROVE_LINKED_ATTRS in msgIn.ulFlags)
            then
                ls := LinkValueStamp of the most recent
                    value change in object!attr
                if ls is more recent than s (based on order in which
                    the change was applied on server) then
                    if s = null then
                        s := 0 /* An AttributeStamp with 0 for all fields. */
                    endif

                    /* Improve the stamp with the link value stamp. */
                    s.dwVersion := ls.dwVersion
                    s.timeChanged := ls.timeChanged
                    s.uuidOriginating := NULL GUID
                    s.usnOriginating := ls.usnOriginating
                endif
            endif
        endif
    endwhile
endif

```

```

if s ≠ null then
  if i ≥ baseIndex
    if infoType = DS_REPL_INFO_METADATA_FOR_OBJ then
      pObjMetaData := ADR(pmsgOut^.pObjMetaData1->rgMetaData[j])
      pObjMetaData^.pszAttributeName := attr
      pObjMetaData^.dwVersion := s.dwVersion
      pObjMetaData^.timeChanged := s.timeChanged
      pObjMetaData^.uuidLastOriginatingDsaInvocationID :=
        s.uuidOriginating
      pObjMetaData^.usnOriginatingChange := s.usnOriginating
      pObjMetaData^.usnLocalChange :=
        An implementation specific value that the server
        maintains for replicated attributes
    else
      pObjMetaData2 := ADR(pmsgOut^.pObjMetaData2->rgMetaData[j])
      pObjMetaData2^.pszAttributeName := attr
      pObjMetaData2^.dwVersion := s.dwVersion
      pObjMetaData2^.timeChanged := s.timeChanged
      pObjMetaData2^.uuidLastOriginatingDsaInvocationID :=
        s.uuidOriginating
      pObjMetaData2^.usnOriginatingChange := s.usnOriginating
      pObjMetaData2^.usnLocalChange :=
        An implementation specific value that the server
        maintains for replicated attributes
      pObjMetaData2^.pszLastOriginatingDsaDN :=
        GetDNFromInvocationID(s.uuidOriginating)
    endif
    j := j + 1
  endif
  i := i + 1
endif
endwhile
pmsgOut^.pObjMetaData2^.cNumEntries = j
endif

/* DS_REPL_INFO_CURSORS_FOR_NC */
if infoType = DS_REPL_INFO_CURSORS_FOR_NC then
  /* Parameter validation */
  /* The NC root object must be specified */
  nc := object

  if nc = null then
    return ERROR_INVALID_PARAMETER
  endif

  if not FullReplicaExists(nc) and
    not PartialGCReplicaExists(nc) then
    return ERROR_DS_DRA_BAD_NC
  endif

  if baseIndex = 0xffffffff then
    /* No more data is available. */
    return ERROR_NO_MORE_ITEMS
  endif

  utd := nc!replUpToDateVector
  i := baseIndex

```

```

j := 0
while i < utd.length
    pCursor := pmsgOut^.pCursors^.rgCursors[j]
    pCursor^.uuidSourceDsaInvocationID := utd[i].uuidDsa
    pCursor^.usnAttributeFilter := utd[i].usnHighPropUpdate
    i := i + 1
    j := j + 1
endwhile
pmsgOut^.pCursors^.cNumCursors := j
endif

/* DS_REPL_INFO_CURSORS_2_FOR_NC/ DS_REPL_INFO_CURSORS_3_FOR_NC */
if infoType in {DS_REPL_INFO_CURSORS_2_FOR_NC,
    DS_REPL_INFO_CURSORS_3_FOR_NC} then

    /* Parameter validation. */
    /* The NC root object must be specified. */
    nc := object

    if (nc = null) then
        return ERROR_INVALID_PARAMETER
    endif

    if not FullReplicaExists(nc) and
        not PartialGCReplicaExists(nc) then
        return ERROR_DS_DRA_BAD_NC
    endif

    if baseIndex = 0xffffffff then
        /* No more data is available. */
        return ERROR_NO_MORE_ITEMS
    endif

    i := baseIndex
    j := 0
    utd := nc!replUpToDateVector

    /* A maximum of 1000 items will be sent in each call. */
    if utd.length - baseIndex - 1 > 1000 then
        endIndex = baseIndex + 1000
    else
        endIndex = utd.length
    endif

    while i < endIndex
        if infoType = DS_REPL_INFO_CURSORS_2_FOR_NC then
            pCursor2 := ADR(pmsgOut^.pCursors2^.rgCursors[j])
            pCursor2^.uuidSourceDsaInvocationID := utd[i].uuidDsa
            pCursor2^.usnAttributeFilter := utd[i].usnHighPropUpdate
            pCursor2^.ftimeLastSyncSuccess := utd[i].timeLastSyncSuccess
        else
            pCursor3 := ADR(pmsgOut^.pCursor3^.rgCursors[j])
            pCursor3^.uuidSourceDsaInvocationID := utd[i].uuidDsa
            pCursor3^.usnAttributeFilter := utd[i].usnHighPropUpdate
            pCursor3^.ftimeLastSyncSuccess := utd[i].timeLastSyncSuccess
            pCursor3^.pszSourceDsaDN :=
                GetDNFromInvocationID(utd[i].uuidDsa)
        endif
    endwhile
endwhile

```



```

        j := j + 1
        i := i + 1
    endwhile
    pmsgOut^.pCursors3^.cNumCursors := j
    if i < utd.length - 1 then
        /* Not all items could be sent back in this call, so save the
         * index of the first item to be sent in the next call. */
        pmsgOut^.pCursors3^.dwEnumerationContext := i
    else
        /* No more data is available. */
        pmsgOut^.pCursors3^.dwEnumerationContext := 0xffffffff
    endif
endif

/* DS_REPL_INFO_UPTODATE_VECTOR_V1 */
if infoType = DS_REPL_INFO_UPTODATE_VECTOR_V1 then
    /* Parameter validation. */
    /* The NC root object must be specified. */
    nc := object

    if (nc = null) then
        return ERROR_INVALID_PARAMETER
    endif

    if not FullReplicaExists(nc) and
        not PartialGCReplicaExists(nc) then
        return ERROR_DS_DRA_BAD_NC
    endif

    utd := nc!replUpToDateVector
    for i := 0 to utd.length - 1
        pCursor := ADR(pmsgOut^.pUpToDateVec^.rgCursors[i])
        pCursor^.uuidDsa := utd[i].uuidDsa
        pCursor^.usnHighPropUpdate := utd[i].usnHighPropUpdate
    endfor
    pmsgOut^.pUpToDateVec^.cNumCursors := utd.length
endif

/* DS_REPL_INFO_METADATA_FOR_ATTR_VALUE/
 * DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE */
if infoType in {DS_REPL_INFO_METADATA_FOR_ATTR_VALUE,
    DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE} then

    /* Parameter validation. */
    if (object = null or not ObjExists(object)) then
        return ERROR_INVALID_PARAMETER
    endif

    if baseIndex = 0xffffffff then
        /* No more data is available. */
        return ERROR_NO_MORE_ITEMS
    endif

    /* If the attribute name is specified it must be a link
     * attribute. */
    attrs := select all a in Link Attributes of object
    if (pmsgIn^.V2.pszAttributeNameValue ≠ null and
        pmsgIn^.V2.pszAttributeNameValue not in attrs) then

```

```

    return ERROR_DS_WRONG_LINKED_ATT_SYNTAX
endif

/* If the attribute name is not specified, replication state for a
 * link attribute of the object which has a value is returned. */
if (pmsgIn^.V2.pszAttributeNameValue ≠ null) then
    attr := pmsgIn^.V2.pszAttributeNameValue
else
    attrSeq := select all a in attrs where
        GetAttrVals(object, a, true) ≠ null
    attr := attrSeq[0]
endif

if attr ≠ null then
    valuesSeq := GetAttrVals(object, attr, true)

    /* If a start value has been specified, then start at the first
     * occurrence of that value in the sequence of values, otherwise
     * start at the index determined from the enumeration context
     * which specifies the index of the next value to be returned. */
    if (pmsgIn^.V2.pszValueDN ≠ null and
        Syntax(attr) = Object(DS-DN)) then
        i := index of pmsgIn^.V2.pszValueDN in valuesSeq
    else
        i := baseIndex
    endif

    j := 0
    while (i < valuesSeq.length and j < 1000)
        ls := LinkStamp(object, attr, valuesSeq[i])
        if infoType = DS_REPL_INFO_METADATA_FOR_ATTR_VALUE then
            pAttrValueMetaData :=
                ADR(pmsgOut^.pAttrValueMetaData^.rgMetadata[j])
            pAttrValueMetaData^.pszAttributeName := attr
            pAttrValueMetaData^.pszObjectDN := object!distinguishedName
            if (Syntax(attr) = Object(DN-Binary) or
                Syntax(attr) = Object(DN-String)) then
                pAttrValueMetaData^.cbData :=
                    length of data associated with valuesSeq[i]
                pAttrValueMetaData^.pbData := data associated with
                    valuesSeq[i]
            endif
            pAttrValueMetaData^.ftimeCreated := ls.timeCreated
            pAttrValueMetaData^.ftimeDeleted := ls.timeDeleted
            pAttrValueMetaData^.dwVersion := ls.dwVersion
            pAttrValueMetaData^.ftimeLastOriginatingChange :=
                ls.timeChanged
            pAttrValueMetaData^.uuidLastOriginatingDsaInvocationID :=
                ls.uuidOriginating
            pAttrValueMetaData^.usnOriginatingChange := ls.usnOriginating
            pAttrValueMetaData^.usnLocalChange :=
                implementation-specific value maintained for each link
                attribute value
        else
            pAttrValueMetaData2 :=
                pmsgOut^.pAttrValueMetaData2^.rgMetadata[j]
            pAttrValueMetaData2^.pszAttributeName := attr
            pAttrValueMetaData2^.pszObjectDN := object!distinguishedName

```

```

        if (Syntax(attr) = Object(DN-Binary) or
            Syntax(attr) = Object(DN-String)) then
            pAttrValueMetaData2^.cbData :=
                length of data associated with valuesSeq[i]
            pAttrValueMetaData2^.pbData :=
                data associated with valuesSeq[i]
        endif
        pAttrValueMetaData2^.ftimeCreated := ls.timeCreated
        pAttrValueMetaData2^.ftimeDeleted := ls.timeDeleted
        pAttrValueMetaData2^.dwVersion := ls.dwVersion
        pAttrValueMetaData2^.ftimeLastOriginatingChange :=
            ls.timeChanged
        pAttrValueMetaData2^.uuidLastOriginatingDsaInvocationID :=
            ls.uuidOriginating
        pAttrValueMetaData2^.usnOriginatingChange :=
            ls.usnOriginating
        pAttrValueMetaData2^.usnLocalChange :=
            implementation-specific value maintained for each
            link attribute value
        pAttrValueMetaData2^.pszLastOriginatingDsaDN :=
            GetDNFromInvocationID(ls.uuidOriginating)
    endif

    i := i + 1
    j := j + 1
endwhile

if i < valuesSeq.length - 1 then
    /* Since there are more entries to be returned, save the index
       * of the first value to be returned in the next call. */
    pmsgOut^.pAttrValueMetaData2^.dwEnumerationContext := i
else
    /* No more data is available. */
    pmsgOut^.pAttrValueMetaData2^.dwEnumerationContext :=
        0xffffffff
endif

pmsgOut^.pAttrValueMetaData2^.cNumEntries = j
endif
endif

/* DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES */
if infoType = DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES then
    i := 0
    foreach t in dc.kccFailedConnections
        pConnectionFailure :=
            ADR(pmsgOut^.pConnectionFailures^.rgDsaFailure[i])
        pConnectionFailure^.pszDsaDN := t.DsaDN
        pConnectionFailure^.uuidDsaObjGuid := t.UUIDDsa
        pConnectionFailure^.ftimeFirstFailure := t.TimeFirstFailure
        pConnectionFailure^.cNumFailures := t.FailureCount
        pConnectionFailure^.dwLastResult := t.LastResult
        i := i + 1
    endfor
    pmsgOut^.pConnectionFailures^.cNumEntries := i
endif

/* DS_REPL_INFO_KCC_LINK_FAILURES */

```

```

if infoType = DS_REPL_INFO_KCC_LINK_FAILURES then
    i := 0
    foreach t in dc.kccFailedLinks
        pConnectionLink := ADR(pmsgOut^.pLinkFailures^.rgDsaFailure[i])
        pConnectionLink^.pszDsaDN := t.DsaDN
        pConnectionLink^.uuidDsaObjGuid := t.UUIDDsa
        pConnectionLink^.fTimeFirstFailure := t.TimeFirstFailure
        pConnectionLink^.cNumFailures := t.FailureCount
        pConnectionLink^.dwLastResult := t.LastResult
        i := i + 1
    endfor
    pmsgOut^.pConnectionLinks^.cNumEntries := i
endif

/* DS_REPL_INFO_PENDING_OPS */
if infoType = DS_REPL_INFO_PENDING_OPS then
    i := 0
    foreach t in dc.replicationQueue
        pPendingOp := ADR(pmsgOut^.pPendingOps^.rgPendingOp[i])
        pPendingOp^.fTimeEnqueued := t.TimeEnqueued
        pPendingOp^.ulSerialNumber := t.SerialNumber
        pPendingOp^.ulPriority := t.Priority
        pPendingOp^.OpType := t.OperationType
        pPendingOp^.ulOptions := t.Options
        pPendingOp^.pszNamingContext := t.NamingContext
        pPendingOp^.pszDsaDN := t.DsaDN
        pPendingOp^.pszDsaAddress := t.DsaAddress
        pPendingOp^.uuidNamingContextObjGuid := t.UUIDNC
        pPendingOp^.uuidDsaObjGuid := t.UUIDDsa
        i := i + 1
    endfor
    pmsgOut^.pPendingOps^.cNumPendingOps := i
    pmsgOut^.pPendingOps^.fTimeCurrentOpStarted := time when current
        operation was started
endif

/* DS_REPL_INFO_CLIENT_CONTEXTS */
if infoType = DS_REPL_INFO_CLIENT_CONTEXTS then
    i := 0
    foreach t in dc.rpcClientContexts
        pClientContext := ADR(pmsgOut^.pClientContexts^.rgContext[i])
        pClientContext^.hCtx := t.BindingContext
        pClientContext^.lReferenceCount := t.RefCount
        pClientContext^.fIsBound := t.IsBound
        pClientContext^.uuidClient := t.UUIDClient
        pClientContext^.timeLastUsed := t.TimeLastUsed
        pClientContext^.IPAddr := t.IPAddress
        pClientContext^.pid := t.PID
        i := i + 1
    endfor
    pmsgOut^.pClientContexts^.cNumContexts := i
endif

/* DS_REPL_INFO_SERVER_OUTGOING_CALLS */
if infoType = DS_REPL_INFO_SERVER_OUTGOING_CALLS then
    i := 0
    foreach t in dc.rpcOutgoingContexts
        pOutgoingContext =

```

```

        ADR(pmsgOut^.pServerOutgoingCalls^.rgContext[i])
pOutgoingContext^.pszServerName := t.ServerName
pOutgoingContext^.fIsHandleBound := t.IsBound
pOutgoingContext^.fIsHandleFromCache := t.HandleFromCache
pOutgoingContext^.fIsHandleInCache := t.HandleInCache
pOutgoingContext^.dwThreadId := t.ThreadId
pOutgoingContext^.dwBindingTimeoutMins := t.BindingTimeout
pOutgoingContext^.dstimeCreated := t.CreateTime
pOutgoingContext^.dwCallType := t.CallType
i := i + 1
endfor
pmsgOut^.pServerOutgoingCalls^.cNumCalls := i
endif
return 0

```

4.1.14 IDL_DRSInitDemotion (Opnum 25)

The **IDL_DRSInitDemotion** method performs the first phase of the removal of a DC from an **AD/LDS** forest. This method is supported only by AD/LDS.

```

ULONG IDL_DRSInitDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_INIT_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_INIT_DEMOTIONREPLY* pmsgOut
);

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.14.1 Method-Specific Concrete Types

4.1.14.1.1 DRS_MSG_INIT_DEMOTIONREQ

The **DRS_MSG_INIT_DEMOTIONREQ** union defines request messages sent to the [IDL_DRSInitDemotion](#) method. Only one version, identified by `dwInVersion = 1`, is currently defined.

```

typedef
[switch_type(DWORD)]
union {

```

```

    [case(1)]
        DRS_MSG_INIT_DEMOTIONREQ_V1 V1;
    } DRS_MSG_INIT_DEMOTIONREQ;

```

V1: Version 1 request. Currently, only one version is defined.

4.1.14.1.2 DRS_MSG_INIT_DEMOTIONREQ_V1

The **DRS_MSG_INIT_DEMOTIONREQ_V1** structure defines a request message sent to the [IDL DRSInitDemotion](#) method.

```

typedef struct {
    DWORD dwReserved;
} DRS_MSG_INIT_DEMOTIONREQ_V1;

```

dwReserved: MUST be 0.

4.1.14.1.3 DRS_MSG_INIT_DEMOTIONREPLY

The **DRS_MSG_INIT_DEMOTIONREPLY** union defines the response messages received from the [IDL DRSInitDemotion](#) method. Only one version, identified by `pdwOutVersion^ = 1`, is currently defined.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_INIT_DEMOTIONREPLY_V1 V1;
    } DRS_MSG_INIT_DEMOTIONREPLY;

```

V1: Version 1 reply.

4.1.14.1.4 DRS_MSG_INIT_DEMOTIONREPLY_V1

The **DRS_MSG_INIT_DEMOTIONREPLY_V1** structure defines a response message received from the [IDL DRSInitDemotion](#) method.

```

typedef struct {
    DWORD dwOpError;
} DRS_MSG_INIT_DEMOTIONREPLY_V1;

```

dwOpError: A Win32 error code, as specified in [\[MS-ERREF\]](#) section 3.0.

4.1.14.2 Server Behavior of the IDL_DRSInitDemotion Method

Informative summary of behavior: Performs the first phase of the removal of a DC from an AD/LDS forest. This phase consists of the following:

1. Disabling both originating and replicated updates to the AD/LDS DC.
2. Marking the database as read-only.

Both of these effects are outside the state model.

```
ULONG
IDL_DRSInitDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_INIT_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_INIT_DEMOTIONREPLY* pmsgOut
)
msgIn: DRS_MSG_INIT_DEMOTIONREQ_V1
ret: DWORD
if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
if not IsMemberOfBuiltinAdminGroup() then
    /* only BA is allowed to demote an AD/LDS service */
    return ERROR_ACCESS_DENIED
endif
Disable
ret := Disable originating and replicated updates
if ret = ERROR_SUCCESS then
    ret := Mark database as read-only
endif
pmsgOut^.dwOpError := ret
pdwMsgOut^ := 1
return ERROR_SUCCESS
```

4.1.15 IDL_DRSInterDomainMove (Opnum 10)

The **IDL_DRSInterDomainMove** method is a helper method used in a cross-NC move LDAP operation.

```
ULONG IDL_DRSInterDomainMove(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_MOVEREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_MOVEREPLY* pmsgOut
);
```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.15.1 Method-Specific Concrete Types

4.1.15.1.1 DRS_MSG_MOVEREQ

The **DRS_MSG_MOVEREQ** union defines the request messages sent to the [IDL_DRSInterDomainMove](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_MOVEREQ_V1 V1;
    [case(2)]
        DRS_MSG_MOVEREQ_V2 V2;
} DRS_MSG_MOVEREQ;
```

V1: The version 1 request (obsolete).

V2: The version 2 request (Windows 2000 and later).

4.1.15.1.2 DRS_MSG_MOVEREQ_V1

The **DRS_MSG_MOVEREQ_V1** structure defines a request message sent to the [IDL_DRSInterDomainMove](#) method. This request version is obsolete. [<21>](#)

```
typedef struct {
    char* pSourceDSA;
    ENTINF* pObject;
    UUID* pParentUUID;
    SCHEMA_PREFIX_TABLE PrefixTable;
    ULONG ulFlags;
} DRS_MSG_MOVEREQ_V1;
```

pSourceDSA: The [NetworkAddress](#) of the client DC.

pObject: The object to be moved.

pParentUUID: The [objectGUID](#) of the new parent object.

PrefixTable: The prefix table with which to translate the [ATTRTYP](#) values in pObject to OIDs.

ulFlags: MUST be 0.

4.1.15.1.3 DRS_MSG_MOVEREQ_V2

The **DRS_MSG_MOVEREQ_V2** structure defines a request message sent to the [IDL DRSInterDomainMove](#) method. This request version is supported by Windows 2000 and later releases of Windows.

```
typedef struct {
    DSNAME* pSrcDSA;
    ENTINF* pSrcObject;
    DSNAME* pDstName;
    DSNAME* pExpectedTargetNC;
    DRS_SecBufferDesc* pClientCreds;
    SCHEMA_PREFIX_TABLE PrefixTable;
    ULONG ulFlags;
} DRS_MSG_MOVEREQ_V2;
```

pSrcDSA: The client DC [nTDSDSA](#) object.

pSrcObject: The object to be moved.

pDstName: The name the object will have in the destination domain.

pExpectedTargetNC: The NC to which pSrcObject is being moved.

pClientCreds: The credentials of the user initiating the call.

PrefixTable: The prefix table with which to translate the [ATTRTYP](#) values in pSrcObject to OIDs.

ulFlags: MUST be 0.

4.1.15.1.4 DRS_MSG_MOVEREPLY

The **DRS_MSG_MOVEREPLY** union defines the response messages received from the [IDL DRSInterDomainMove](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_MOVEREPLY_V1 V1;
    [case(2)]
        DRS_MSG_MOVEREPLY_V2 V2;
} DRS_MSG_MOVEREPLY;
```

V1: The version 1 response (obsolete).

V2: The version 2 response (Windows 2000 and later).

4.1.15.1.5 DRS_MSG_MOVE_REPLY_V1

The **DRS_MSG_MOVE_REPLY_V1** structure defines a response message received from the [IDL DRSTransferDomainMove](#) method. This response version is obsolete. <22>

```
typedef struct {
    ENTINF** ppResult;
    SCHEMA_PREFIX_TABLE PrefixTable;
    ULONG* pError;
} DRS_MSG_MOVE_REPLY_V1;
```

ppResult: The object as it appears following the move operation.

PrefixTable: The prefix table with which to translate the [ATTRTYP](#) values in ppResult to OIDs.

pError: 0 if successful, or not 0 if a fatal error occurred.

4.1.15.1.6 DRS_MSG_MOVE_REPLY_V2

The **DRS_MSG_MOVE_REPLY_V2** structure defines a response message received from the [IDL DRSTransferDomainMove](#) method. This response version is supported by Windows 2000 and later.

```
typedef struct {
    ULONG win32Error;
    [unique] DSNAME* pAddedName;
} DRS_MSG_MOVE_REPLY_V2;
```

win32Error: 0 if successful, or not 0 if a fatal error occurred.

pAddedName: The name of the object in its new domain.

4.1.15.2 Method-Specific Abstract Types and Procedures

4.1.15.2.1 AttrIsBacklink

```
procedure AttrIsBacklink(attr: ATTRTYP): boolean
```

Returns true if the attribute attr is a back link, and returns false otherwise.

```
return SchemaObj(attr)!linkID mod 2 = 1
```

4.1.15.2.2 AttrIsConstructed

```
procedure AttrIsConstructed(attr: ATTRTYP): boolean
```

Returns true if the attribute attr is a constructed attribute, and returns false otherwise.

```
return FLAG_ATTR_IS_CONSTRUCTED in SchemaObj(attr)!systemFlags
```

4.1.15.2.3 AttrIsNonReplicated

```
procedure AttrIsNonReplicated(attr: ATTRTYP): boolean
```

Returns true if the attribute attr is a nonreplicated attribute, and returns false otherwise.

```
return FLAG_ATTR_NOT_REPLICATED in SchemaObj(attr)!systemFlags
```

4.1.15.2.4 AuthorizationInfoFromClientCredentials

```
procedure AuthorizationInfoFromClientCredentials(  
  credBuffer: DRS_SecBufferDesc,  
  var token: ClientAuthorizationInfo): DWORD
```

Generates a [ClientAuthorizationInfo](#) token (which is a security token) from client credentials credBuffer. See [\[MS-DTYP\]](#) section 2.5.2 for more details. Returns 0 if it succeeds, or a Windows error code if it fails.

4.1.15.2.5 ImpersonateAuthorizationInfo

```
procedure ImpersonateAuthorizationInfo(token: ClientAuthorizationInfo)
```

Impersonates a set of client credentials. This affects the outcome of all subsequent [AccessCheckAttr](#), [AccessCheckCAR](#), [AccessCheckObject](#), [AccessCheckWriteToSpnAttribute](#), and related calls, until [RevertToSelf](#) is called.

4.1.15.2.6 IsApplicationNC

```
procedure IsApplicationNC(nc: DSName): boolean
```

Returns true if, and only if, nc is an application NC.

4.1.15.2.7 RevertToSelf

```
procedure RevertToSelf()
```

Undoes the effect of [ImpersonateAuthorizationInfo](#). After the RevertToSelf procedure is called, the security context is restored to what it was before ImpersonateAuthorizationInfo was called.

4.1.15.3 Server Behavior of the IDL_DRSInterDomainMove Method

Informative summary of behavior: [IDL_DRSInterDomainMove](#) is used during a cross-NC move operation. This is a special object move operation because it involves moving an object from one DC into another. A normal move operation moves the object within one NC on one DC; a cross-NC move involves two DCs. **IDL_DRSInterDomainMove** is an intermediate step in the cross-NC move

operation, which is initiated by an LDAP call. The **IDL_DRSInterDomainMove** call is performed by the source DC on the target DC in order to move the object with all of its data from one NC replica into another. **Note** **IDL_DRSInterDomainMove** transfers data that is normally not readable by the end user (such as password hashes and other secrets). During the move, the **ENTINF** structure that contains the object data is constructed by the source DC and passed to the target DC. The target DC enforces certain constraints, transforms the data according to the processing rules, and then either creates the object in its NC replica or updates the existing object. For more information on cross-NC move operations, see [\[MS-ADTS\]](#) section 3.1.1.5.4.2.

```

ULONG
IDL_DRSInterDomainMove(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_MOVEREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_MOVEREPLY *pmsgOut)

msgIn: DRS_MSG_MOVEREQ_V2
lastPrefixTableEntry: PrefixTableEntry
prefixTable: PrefixTable
dwErr: DWORD
clientCreds: ClientAuthorizationInfo
callerCreds: ClientAuthorizationInfo
O: ENTINF
existingObj: DSName
attribute: ATTRTYP
proxyEpoch: DWORD

pdwOutVersion^ := 2
msgOut^.V2.win32error := ERROR_DS_GENERIC_ERROR
msgOut^.V2.pAddedName := null
if dwInVersion # 2 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V2
if msgIn.pExpectedTargetNC # DefaultNC() then
    return ERROR_DS_DST_NC_MISMATCH
endif
if msgIn.PrefixTable.PrefixCount < 1 then
    return ERROR_SCHEMA_MISMATCH
endif

/* Originating updates are blocked on an RODC */
if AmIRODC() then

    return ERROR_DS_DRA_INTERNAL_ERROR
endif

/* Remember last prefix table entry, and remove it from the prefix
 * table.*/
lastPrefixTableEntry :=
    msgIn.PrefixTable.pPrefixEntry[msgIn.PrefixTable.PrefixCount]
msgIn.PrefixTable.PrefixCount := msgIn.PrefixTable.PrefixCount-1

/* Perform a binary comparison of the OID value from the last

```

```

    * prefixTable entry with the schemaInfo attribute on the
    * schema NC.*/
    if lastPrefixTableEntry.oid ≠ SchemaNC()!schemaInfo then
        return ERROR_SCHEMA_MISMATCH
    endif

    prefixTable := AbstractPTFromConcretePT(msgIn.PrefixTable)

    /* Convert client creds into ClientAuthorizationInfo format. */
    dwErr := AuthorizationInfoFromClientCredentials(msgIn.pClientCreds,
        clientCreds)
    if dwErr ≠ ERROR_SUCCESS then
        return dwErr
    endif

    /* Check that the caller (the "source" DC) is actually a DC by
    * checking Enterprise Domain Controllers SID in its token. */
    callerCreds := GetCallerAuthorizationInfo()
    if not CheckGroupMembership(callerCreds, SidFromStringSid("S-1-5-9"))
        then
        return ERROR_DS_DRA_ACCESS_DENIED
    endif

    /* Validate input ENTINF. */
    O := msgIn.pSrcObject^

    if ADS_UF_SERVER_TRUST_ACCOUNT in
        ENTINF_GetValue(
            O, userAccountControl, prefixTable) or
        ADS_UF_INTERDOMAIN_TRUST_ACCOUNT in
        ENTINF_GetValue(
            O, userAccountControl, prefixTable) then
        /* Disallowed to move DC accounts and trust objects. */
        return ERROR_DS_ILLEGAL_XDOM_MOVE_OPERATION
    endif
    existingObj := select one obj from all where
        (obj!distinguishedName = ENTINF_GetValue(O, distinguishedName,
            prefixTable))
    if existingObj ≠ NULL and
        existingObj!objectGUID ≠
        ENTINF_GetValue(O, objectGUID, prefixTable) then
        /* There's already an object with the same DN but different GUID.*/
        return ERROR_DS_SRC_GUID_MISMATCH
    endif

    existingObj := select one obj from all where
        (obj!objectGUID = ENTINF_GetValue(O, objectGUID, prefixTable))
    if existingObj ≠ NULL and existingObj!proxiedObjectName ≠
        ENTINF_GetValue(O, proxiedObjectName, prefixTable) then
        /* There's already an object with the same guid,
        * but proxiedObjectName is different - not allowed. */
        return ERROR_DS_EPOCH_MISMATCH
    endif

    if IsApplicationNC(GetObjectNC(O.pName^)) then
        return ERROR_DS_INTERNAL_FAILURE
    endif

```

```

/* Scan through the ENTINF and throw away any attributes that are not
 * supposed to be moved. */
foreach attribute in ENTINF_EnumerateAttributes(0, prefixTable)
  if AttrIsBacklink(attribute) or AttrIsNonReplicated(attribute) or
    AttrIsConstructed(attribute) then
    ENTINF_SetValue(0, attribute, null, prefixTable)
  endif
  if attribute in {
    adminCount, badPasswordTime, badPwdCount, creationTime,
    distinguishedName, domainReplica, instanceType,
    isCriticalSystemObject, isDeleted, lastLogoff, lastLogon,
    lastLogonTimestamp, lockoutTime, logonCount, modifiedCount,
    modifiedCountAtLastProm, msDS-Cached-Membership,
    msDS-Cached-Membership-Time-Stamp, msDS-Site-Affinity, nextRid,
    nTSecurityDescriptor, objectCategory, operatorCount,
    primaryGroupID, proxiedObjectName, replPropertyMetaData,
    revision, rid, sAMAccountType, serverState, subRefs,
    systemFlags, uASCompat, uSNChanged, uSNCreated,
    uSNSALastObjRemoved, uSNLastObjRem, whenChanged, whenCreated}
    then
      ENTINF_SetValue(0, attribute, null, prefixTable)
    endif
  endif
endfor

if ENTINF_GetValue(0, userAccountControl, prefixTable) ≠ null then
  /* Reset lockout bit. */
  ENTINF_SetValue(0, userAccountControl,
    ENTINF_GetValue(0, userAccountControl) - {ADS_UF_LOCKOUT},
    prefixTable)
endif

if ENTINF_GetValue(0, pwdLastSet, prefixTable) ≠ null and
  ENTINF_GetValue(0, pwdLastSet, prefixTable) ≠ 0 then
  /* If pwdLastSet is set to non-zero, then change it to -1. */
  ENTINF_SetValue(0, pwdLastSet, (LONGLONG)-1, prefixTable)
endif

/* Append objectSid to sidHistory. */
ENTINF_SetValue(0, sidHistory,
  ENTINF_GetValue(0, sidHistory, prefixTable)
  + {ENTINF_GetValue(0, objectSid, prefixTable)})

/* Compute the new proxiedObjectName value. */
if ENTINF_GetValue(0, proxiedObjectName, prefixTable) ≠ null and
  GetProxyType(ENTINF_GetValue(0, proxiedObjectName)) =
    PROXY_TYPE_MOVED_OBJECT then
  /* There's already a valid proxiedObjectName on the object,
   * so just increment the epoch value. */
  proxyEpoch := GetProxyEpoch(ENTINF_GetValue(0, proxiedObjectName,
    prefixTable))+1
else
  /* No valid proxiedObjectName, so start a new one. */
  proxyEpoch := 1
endif

/* Stamp the new proxiedObjectName value into ENTINF. */
ENTINF_SetValue(0,
  proxiedObjectName,
  MakeProxyValue(msgIn.pSrcNC^,

```

```

                                PROXY_TYPE_MOVED_OBJECT,
                                proxyEpoch),
    prefixTable)

if existingObj ≠ null then
    /* Purge existing object, we are about to overwrite it. */
    Expunge(existingObj)
endif

ImpersonateAuthorizationInfo(clientCreds)
dwErr :=
    PerformAddOperation(
        0,
        msgOut^.V2.pAddedName^,
        AbstractPTFromConcretePT(msgIn.PrefixTable))
RevertToSelf()

msgOut^.V2.win32error = dwErr
return dwErr

```

4.1.15.4 Examples of the IDL_DRSInterDomainMove Method

In this example, a user is moved from the domain NC ASIA.CONTOSO.COM to the domain NC CONTOSO.COM.

4.1.15.4.1 Initial State

Querying the [user](#) object for Aaron Con in the domain NC ASIA.CONTOSO.COM on DCA1, prior to the move:

- `ldap_search_s("CN=Aaron Con,CN=Users,DC=asia,DC=contoso,DC=com", singleLevel, "(objectclass=*)", [objectClass, cn, ... objectCategory])`
- Getting 1 entries:
- `>> Dn: CN=Aaron Con,CN=Users,DC=asia,DC=contoso,DC=com`
 - `4> objectClass: top; person; organizationalPerson; user;`
 - `1> cn: Aaron Con;`
 - `1> sn: Con;`
 - `1> givenName: Aaron;`
 - `1> distinguishedName: CN=Aaron Con, CN=Users, DC=asia, DC=contoso, DC=com;`
 - `1> instanceType: 0x4 = (IT_WRITE);`
 - `1> whenCreated: 07/12/2006 17:25:53 Pacific Std Daylight Time;`
 - `1> whenChanged: 07/12/2006 17:25:54 Pacific Std Daylight Time;`
 - `1> displayName: Aaron Con;`
 - `1> uSNCreated: 13798;`

- 1> uSNChanged: 13803;
- 1> name: Aaron Con;
- 1> objectGUID: 45a6999f-31eb-40ab-a2e5-906ccd86d5eb;
- 1> userAccountControl: 0x200 = (UF_NORMAL_ACCOUNT);
- 1> badPwdCount: 0;
- 1> codePage: 0;
- 1> countryCode: 0;
- 1> badPasswordTime: 01/01/1601 00:00:00 UNC ;
- 1> lastLogoff: 01/01/1601 00:00:00 UNC ;
- 1> lastLogon: 01/01/1601 00:00:00 UNC ;
- 1> pwdLastSet: 07/12/2006 17:25:53 Pacific Std Pacific Daylight Time;
- 1> primaryGroupID: 513;
- 1> objectSid: S-1-5-21-1880045291-2375173688-894673254-1109;
- 1> accountExpires: 09/14/30828 02:48:05 UNC ;
- 1> logonCount: 0;
- 1> sAMAccountName: aaroncon;
- 1> sAMAccountType: 805306368;
- 1> userPrincipalName: aaroncon@asia.contoso.com;
- 1> objectCategory: CN=Person, CN=Schema, CN=Configuration, DC=contoso, DC=com;

Querying the [user](#) object for Aaron Con in the domain NC CONTOSO.COM on DC1 prior to the move yields no results, as follows:

- ldap_search_s("CN=Aaron Con,CN=Users,DC=contoso,DC=com", *oneLevel*, "(objectclass=*)", [*objectClass*, *cn*, ... *objectCategory*])
- Error: Search: No Such Object.
- Matched DNs: CN=Users,DC=contoso,DC=com
- Getting 0 entries:

4.1.15.4.2 Client Request

An LDAP client invokes the [IDL DRSInterDomainMove](#) method against a DC named DCA1.ASIA.CONTOSO.COM with the following parameters ([DRS_HANDLE](#) to DCA1 omitted):

- dwInVersion = 2
- pmsgIn = DRS_MSG_MOVEREQ_V2

- pSrcDSA: CN=NTDS Settings,CN=DCA1,CN=Servers, CN=Default-First-Site-Name,CN=Sites, CN=Configuration,DC=contoso,DC=com
- pSrcObject: ENTINF
 - objectClass: top; person; organizationalPerson; user
 - cn: Aaron Con
 - sn: Con
 - givenName: Aaron
 - instanceType: IT_WRITE
 - whenCreated: 07/12/2006 17:25:53 Pacific Std Time Pacific Daylight Time;
 - whenChanged: 07/12/2006 17:25:54 Pacific Std Time Pacific Daylight Time;
 - displayName: Aaron Con;
 - uSNCreated: 13798;
 - uSNChanged: 13803;
 - objectGUID: 45a6999f-31eb-40ab-a2e5-906ccd86d5eb;
 - userAccountControl: 0x200 = (UF_NORMAL_ACCOUNT);
 - badPwdCount: 0;
 - countryCode: 0;
 - badPasswordTime: 01/01/1601 00:00:00 UNC ;
 - lastLogoff: 01/01/1601 00:00:00 UNC ;
 - lastLogon: 01/01/1601 00:00:00 UNC ;
 - dBCSPwd: *Binary data*
 - unicodePwd: *Binary data*
 - supplementalCredentials: *none*
 - pwdLastSet: 07/12/2006 17:25:53 Pacific Std Time Pacific Daylight Time;
 - primaryGroupID: 513;
 - objectSid: S-1-5-21-1880045291-2375173688-894673254-1109;
 - accountExpires: 09/14/30828 02:48:05 UNC ;
 - logonCount: 0;
 - sAMAccountName: aaroncon;
 - sAMAccountType: SAM_NORMAL_USER_ACCOUNT;
 - userPrincipalName: aaroncon@asia.contoso.com;

- objectCategory: CN=Person, CN=Schema, CN=Configuration, DC=contoso, DC=com;
- nTSecurityDescriptor: *Binary data*
- replPropertyMetaData: *omitted*
- pDstName: CN=Aaron Con,CN=Users,DC=contoso,DC=com
- pExpectedTargetNC: DC=contoso,DC=com
- pClientCreds: DRS_SecBufferDesc
- PrefixTable: SCHEMA_PREFIX_TABLE
- ulFlags: None

4.1.15.4.3 Server Response

Return code of 0 with the following values:

- pdwOutVersion = 2
- pmsgOut = DRS_MSG_MOVEREPLY_V2
 - pAddedName: CN=Aaron Con,CN=Users,DC=contoso,DC=com

4.1.15.4.4 Final State

After the move, the [user](#) object for Aaron Con is not present on domain NC ASIA.CONTOSO.COM, querying DCA1, as follows:

- ldap_search_s("CN=Aaron Con,CN=Users,DC=asia,DC=contoso,DC=com", *singleLevel*, "(objectclass=*)", [*distinguishedName*, *objectGUID*, *userAccountControl*, *objectSid*, *sAMAccountName*, *userPrincipalName*])
- Error: Search: No Such Object.
- Matched DNs: CN=Users,DC=asia,DC=contoso,DC=com
- Getting 0 entries:

After the move, the [user](#) object for Aaron Con is now present on domain NC CONTOSO.COM, querying DC1:

- ldap_search_s("CN=Aaron Con,CN=Users, DC=contoso,DC=com", *oneLevel*, "(objectclass=*)", [*cn*, *distinguishedName* ... *proxiedObjectName*, *dSCorePropagationData*])
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=Aaron Con,CN=Users,DC=contoso,DC=com
 - 4> objectClass: top; person; organizationalPerson; user;
 - 1> cn: Aaron Con;
 - 1> sn: Con;

- 1> givenName: Aaron;
- 1> distinguishedName: CN=Aaron Con,CN=Users,DC=contoso,DC=com;
- 1> instanceType: 0x4 = (IT_WRITE);
- 1> whenCreated: 07/12/2006 17:32:04 Pacific Standard Daylight Time;
- 1> whenChanged: 07/12/2006 17:32:04 Pacific Standard Daylight Time;
- 1> displayName: Aaron Con;
- 1> uSNCreated: 15366;
- 1> uSNChanged: 15369;
- 1> name: Aaron Con;
- 1> objectGUID: 45a6999f-31eb-40ab-a2e5-906ccd86d5eb;
- 1> userAccountControl: 0x200 = (UF_NORMAL_ACCOUNT);
- 1> badPwdCount: 0;
- 1> codePage: 0;
- 1> countryCode: 0;
- 1> badPasswordTime: 01/01/1601 00:00:00 UNC ;
- 1> lastLogoff: 01/01/1601 00:00:00 UNC ;
- 1> lastLogon: 01/01/1601 00:00:00 UNC ;
- 1> pwdLastSet: 07/12/2006 17:32:04 Pacific Standard Daylight Time;
- 1> primaryGroupID: 513;
- 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1111;
- 1> accountExpires: 09/14/30828 02:48:05 UNC ;
- 1> logonCount: 0;
- 1> sAMAccountName: aaroncon;
- 1> sAMAccountType: 805306368;
- 1> sIDHistory: S-1-5-21-1880045291-2375173688-894673254-1109;
- 1> userPrincipalName: aaroncon@asia.contoso.com;
- 1> objectCategory: CN=Person, CN=Schema, CN=Configuration, DC=contoso, DC=com;
- 1> proxiedObjectName: B:16:0000000000000001:DC=asia, DC=contoso, DC=com;
- 1> dScorePropagationData: 07/12/2006 17:32:04 Pacific Standard Daylight Time;
07/12/2006 17:32:04 Pacific Standard Time Pacific Daylight Time; 01/01/1601 01:08:16 UNC
;

4.1.16 IDL_DRSQuerySitesByCost (Opnum 24)

The **IDL_DRSQuerySitesByCost** method determines the communication cost from a "from" site to one or more "to" sites.

```
ULONG IDL_DRSQuerySitesByCost(  
    [in, ref] DRS_HANDLE hDrs,  
    [in] DWORD dwInVersion,  
    [in, ref, switch_is(dwInVersion)]  
        DRS_MSG_QUERY_SITESREQ* pmsgIn,  
    [out, ref] DWORD* pdwOutVersion,  
    [out, ref, switch_is(*pdwOutVersion)]  
        DRS_MSG_QUERY_SITESREPLY* pmsgOut  
);
```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, or a Windows error code if a failure occurs.

4.1.16.1 Method-Specific Concrete Types

4.1.16.1.1 DRS_MSG_QUERY_SITESREQ

The **DRS_MSG_QUERY_SITESREQ** union defines the request message versions sent to the [IDL_DRSQuerySitesByCost](#) method. Only one version, identified by dwVersion = 1, is currently defined.

```
typedef  
[switch_type(DWORD)]  
union {  
    [case(1)]  
        DRS_MSG_QUERY_SITESREQ_V1 V1;  
} DRS_MSG_QUERY_SITESREQ;
```

V1: The version 1 request.

4.1.16.1.2 DRS_MSG_QUERY_SITESREQ_V1

The **DRS_MSG_QUERY_SITESREQ_V1** structure defines a request message sent to the [IDL_DRSQuerySitesByCost](#) method.

```
typedef struct {  
    [string] const WCHAR* pwszFromSite;  
    [range(1,10000)] DWORD cToSites;  
    [string, size_is(cToSites)] WCHAR** rgpszToSites;
```

```

    DWORD dwFlags;
} DRS_MSG_QUERY_SITESREQ_V1;

```

pwszFromSite: The RDN of the [site](#) object of the "from" site.

cToSites: The count of items in the rgszToSites array (the count of "to" sites).

rgszToSites: The RDNs of the [site](#) objects of the "to" sites.

dwFlags: MUST be 0.

4.1.16.1.3 DRS_MSG_QUERY_SITESREPLY

The **DRS_MSG_QUERY_SITESREPLY** union defines the response messages received from the [IDL DRSQuerySitesByCost](#) method.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_QUERY_SITESREPLY_V1 V1;
} DRS_MSG_QUERY_SITESREPLY;

```

V1: The version 1 response.

4.1.16.1.4 DRS_MSG_QUERY_SITESREPLY_V1

The **DRS_MSG_QUERY_SITESREPLY_V1** structure defines a response message received from the [IDL DRSQuerySitesByCost](#) method.

```

typedef struct {
    [range(0,10000)] DWORD cToSites;
    [size_is(cToSites)] DRS_MSG_QUERY_SITESREPLYELEMENT_V1* rgCostInfo;
    DWORD dwFlags;
} DRS_MSG_QUERY_SITESREPLY_V1;

```

cToSites: The count of items in the rgCostInfo array.

rgCostInfo: The sequence of computed site costs, in the same order as the rgszToSites field in the request message.

dwFlags: MUST be 0.

4.1.16.1.5 DRS_MSG_QUERY_SITESREPLYELEMENT_V1

The **DRS_MSG_QUERY_SITESREPLYELEMENT_V1** structure defines the computed cost of communication between two sites.

```

typedef struct {

```

```

    DWORD dwErrorCode;
    DWORD dwCost;
} DRS_MSG_QUERY_SITES_REPLY_ELEMENT_V1;

```

dwErrorCode: 0 if this "from-to" computation was successful, or ERROR_DS_OBJ_NOT_FOUND if the "to" [site](#) does not exist.

dwCost: The communication cost between the "from" [site](#) and this "to" [site](#), or 0xFFFFFFFF if the sites are not connected.

4.1.16.2 Method-Specific Abstract Types and Procedures

4.1.16.2.1 ValidateSiteRDN

```

procedure ValidateSiteRDN(s: uncodestring): boolean

```

Returns true if s is a valid RDN for a [site](#) object. A valid RDN has the following characteristics:

- Is not null.
- Does not have a 0 length.
- Does not have a length greater than 256.
- Contains no occurrences of the equal sign (=) and comma (,).

4.1.16.2.2 WeightedArc and WeightedArcSet

```

type WeightedArc = [initial: DSName, final: DSName, cost: integer]
type WeightedArcSet = set of WeightedArc

```

The cost field of a WeightedArc is positive.

4.1.16.2.3 MinWeightPath

```

procedure MinWeightPath(
    vSet: set of DSName,
    aSet: WeightedArcSet): WeightedArcSet

```

Returns a [WeightedArcSet](#) where for each WeightedArc a:

- a.initial and a.final are vertices in vSet
- a.final is reachable from a.initial in the graph G = (vSet, aSet)
- a.cost is the cost of the minimum-cost path in G from a.initial to a.final.

4.1.16.3 Server Behavior of the IDL_DRSQuerySitesByCost Method

Informative summary of behavior: Given a site *fromSite* and an array of sites *toSites*, the server returns an array that contains the cost from *fromSite* to each element of *toSite*, where the cost is defined as follows.

The server computes a weighted graph $G = (V, A)$. Each vertex in V corresponds to a [site](#) object. Each arc in A corresponds to a [siteLink](#) object that connects two vertices in V ; the weight of an arc is the value of attribute [cost](#) on the arc's [siteLink](#) object. The cost of a path in the graph is the sum of the arc weights on the path. The cost from one site to another is the minimum-cost path between the two sites.

The model just described corresponds to fully transitive communications between sites: If site *a* communicates with site *b* and site *b* communicates with site *c*, then site *a* communicates with site *c* by routing through *b*. Replication can be configured to restrict transitive communication to sites specified in the same [siteLinkBridge](#) object. Suppose there is a [siteLink](#) object for site *a* and site *b*, and a [siteLink](#) object for site *b* and site *c*, but no [siteLink](#) object for site *a* and site *c*. If both of the [siteLink](#) objects are specified on the same [siteLinkBridge](#) object, site *a* can communicate with site *c* by routing through *b*. If no such [siteLinkBridge](#) object exists, site *a* cannot communicate with site *c*.

To calculate the cost when [siteLinkBridge](#) objects are used, let *nBridges* be the number of [siteLinkBridge](#) objects. For each *k* in the subrange $[0 .. nBridges-1]$, construct a weighted graph $G[k] = (V, A[k])$ using [siteLinkBridge](#) object *b[k]*. Graph $G[k]$ has the same vertex set as G , but its arc set $A[k]$ is a subset of A , including only the arcs listed in attribute [siteLinkList](#) on [siteLinkBridge](#) object *b[k]*. Then the cost from site *a* to site *c* is the minimum of the following costs:

1. The cost of the arc, if any, from *a* to *c* in G .
2. For each *k* in the subrange $[0 .. nBridges-1]$, the cost of the minimum cost path, if any, from *a* to *c* in $G[k]$.

Any authenticated user can perform this operation; no access checking is performed. [<23>](#)

```
ULONG
IDL_DRSQuerySitesByCost(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_QUERYSITESREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_QUERYSITESREPLY *pmsgOut)
msgIn: DRS_MSG_QUERYSITESREQ_V1
vSet, slSet, sbSet : set of DSName
aSet, aSetB, aSetC, aSetD: WeightedArcSet
siteContainer, ipObject, fromSite, toSite: DSName
u, v, sl, sb: DSName
i, c: integer
min: WeightedArc
/* Perform input validation,
 * initialize siteContainer, ipObject, fromSite. */
if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
if not ValidateSiteRDN(msgIn.pwszFromSite) then
    return ERROR_INVALID_PARAMETER
```

```

endif
if msgIn.cToSites > 0 and msgIn.rgszToSites = null then
    return ERROR_INVALID_PARAMETER
endif
for i := 0 to msgIn.cToSites - 1
    if not ValidateSiteRDN(msgIn.rgszToSites[i]) then
        return ERROR_INVALID_PARAMETER
    endif
endfor
siteContainer := DescendantObject(ConfigNC(), "CN=Sites,")
ipObject := DescendantObject(ConfigNC(),
    "CN=IP,CN=Inter-Site Transports,CN=Sites,")
fromSite := select one v from children siteContainer where
    site in v!objectClass and v!name = msgIn.pswzFromSite
if fromSite = null then
    return ERROR_DS_OBJ_NOT_FOUND
endif
/* Construct the vertex set vSet. */
vSet := select all v from children siteContainer where
    site in v!objectClass
if vSet = {} then
    return ERROR_DS_OBJ_NOT_FOUND
endif
/* Construct the arc set aSet. */
slSet := select all v from children ipObject where
    siteLink in v!objectClass
foreach sl in slSet
    foreach u in sl!siteList
        foreach v in sl!siteList - {u}
            aSet := aSet + {[initial: u, final: v, cost: sl!cost]}
        endfor
    endfor
endfor
/* Construct minimum-cost arc set aSetC.
 * See [MS-ADTS] section 7.1.1.2.2.3.1 for definition
 * of the option NTDSTRANSFORM_OPT_BRIDGES_REQUIRED. */
if NTDSTRANSFORM_OPT_BRIDGES_REQUIRED in ipObject!options then
    /* Perform construction using siteLinkBridge objects.
     * Initial minimum cost is the cost of a direct arc if any. */
    aSetC := aSet
    sbSet := select all v from children ipObject where
        siteLinkBridge in v!objectClass
    foreach sb in sbSet
        /* Compute the minimum cost using this siteLinkBridge. */
        aSetB := {}
        foreach sl in sb!siteLinkList
            foreach u in sl!siteList
                foreach v in sl!siteList - {u}
                    aSetB := aSetB + {[initial: u, final: v, cost: sl!cost]}
                endfor
            endfor
        endfor
    endfor
    aSetD := MinWeightPath(vSet, aSetB)
    /* Here aSetD contains the minimum cost arc set using this
     * siteLinkBridge. Improve the current minimum cost using
     * aSetD. */
    foreach [initial: u, final: v, cost: c] in aSetD
        min := select one t from aSetC where

```



```

        t.initial = u and t.final = v
    if min = null then
        aSetC := aSetC + {[initial: u, final: v, cost: c]}
    else if min.cost > c then
        aSetC := aSetC - {[initial: u, final: v, cost: min.cost]}
        + {[initial: u, final: v, cost: c]}
    endif
endfor
endif
endif
else
    /* Fully transitive network, ignore siteLinkBridge objects. */
    aSetC := MinWeightPath(vSet, aSet)
endif
/* Construct result message. */
pdwOutVersion^ := 1
pmsgOut^.Vl.cToSites := msgIn.cToSites
pmsgOut^.Vl.dwFlags := 0
for i:= 0 to msgIn.cToSites - 1
    toSite := select one v from children siteContainer where
        site in v!objectClass and v!name = msgIn.rgszToSites[i]
    if not (toSite in vSet) then
        pmsgOut^.Vl.rgCostInfo[i].dwErrorCode := ERROR_DS_OBJ_NOT_FOUND
        pmsgOut^.Vl.rgCostInfo[i].dwCost := 0xffffffff
    else
        min := select one t from aSetC where
            t.initial = fromSite and t.final = toSite
        if min ≠ null then
            pmsgOut^.Vl.rgCostInfo[i].dwErrorCode := 0
            pmsgOut^.Vl.rgCostInfo[i].dwCost := min.cost
        else
            pmsgOut^.Vl.rgCostInfo[i].dwErrorCode = 0
            pmsgOut^.Vl.rgCostInfo[i].dwCost := 0xffffffff
        endif
    endif
endfor
return 0

```

4.1.17 IDL_DRSRemoveDsDomain (Opnum 15)

The **IDL_DRSRemoveDsDomain** method removes the representation (also known as the metadata) of a domain from the directory.

```

ULONG IDL_DRSRemoveDsDomain(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_RMDMNREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_RMDMNREPLY* pmsgOut
);

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message. This MUST be set to 1, as this is the only version supported.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message. The value is always 1 because that is the only version supported.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, or a Windows error code if a failure occurs.

4.1.17.1 Method-Specific Concrete Types

4.1.17.1.1 DRS_MSG_RMDMNREQ

The **DRS_MSG_RMDMNREQ** union defines the request messages sent to the [IDL DRSRemoveDsDomain](#) method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_RMDMNREQ_V1 V1;
} DRS_MSG_RMDMNREQ;
```

V1: The version 1 request.

4.1.17.1.2 DRS_MSG_RMDMNREQ_V1

The **DRS_MSG_RMDMNREQ_V1** structure defines a request message sent to the [IDL DRSRemoveDsDomain](#) method.

```
typedef struct {
    [string] LPWSTR DomainDN;
} DRS_MSG_RMDMNREQ_V1;
```

DomainDN: The DN of the NC root of the domain NC to remove.

4.1.17.1.3 DRS_MSG_RMDMNREPLY

The **DRS_MSG_RMDMNREPLY** union defines the response messages received from the [IDL DRSRemoveDsDomain](#) method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_RMDMNREPLY_V1 V1;
```

```
} DRS_MSG_RMDMNREPLY;
```

V1: The version 1 response.

4.1.17.1.4 DRS_MSG_RMDMNREPLY_V1

The **DRS_MSG_RMDMNREPLY_V1** structure defines a response message received from the [IDL_DRSRemoveDsDomain](#) method.

```
typedef struct {  
    DWORD Reserved;  
} DRS_MSG_RMDMNREPLY_V1;
```

Reserved: MUST be 0.

4.1.17.2 Method-Specific Abstract Types and Procedures

4.1.17.2.1 HasNCReplicated

```
procedure HasNCReplicated(nc: DSName): boolean
```

Returns true if the DC's NC replica of the NC specified by nc has replicated at least once with another DC that hosts that NC since the DC was booted; otherwise, returns false.

4.1.17.3 Server Behavior of the IDL_DRSRemoveDsDomain Method

Informative summary of behavior: Removes the [crossRef](#) object that defines a domain NC. Fails if any DC is currently hosting this domain as its default NC, as indicated by the state of that DC's [nTDSDSA](#) object. Fails if the server is not the domain naming FSMO role owner for the forest.

The removal of the [crossRef](#) object signals any DC currently hosting a partial replica of the removed domain NC to remove that replica from its state.

This method undoes the effects of the [IDL_DRSAddEntry](#) method when **IDL_DRSAddEntry** is used to create a [crossRef](#) object.

The [IDL_DRSRemoveDsServer](#) method removes the state within a forest, including the state on a DC's [nTDSDSA](#) object, associated with hosting a domain as a default NC on some DC. Therefore, **IDL_DRSRemoveDsServer** can be used to establish a precondition for the success of [IDL_DRSRemoveDsDomain](#).

```
ULONG  
IDL_DRSRemoveDsDomain(  
    [in, ref] DRS_HANDLE hDrs,  
    [in] DWORD dwInVersion,  
    [in, ref, switch_is(dwInVersion)]  
        DRS_MSG_RMDMNREQ *pmsgIn,  
    [out, ref] DWORD *pdwOutVersion,  
    [out, ref, switch_is(*pdwOutVersion)]  
        DRS_MSG_RMDMNREPLY *pmsgOut);
```

```

domainDN: unicodestring
otherNtdsdsa: DSName
cr: DSName

pdwOutVersion^ := 1
pmsgOut^.V1.Reserved := 0

if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif

domainDN := pmsgIn^.V1.DomainDN

if domainDN = null or domainDN = "" then
    return ERROR_INVALID_PARAMETER
endif

/* Originating updates are not allowed on RODC */
if AmIRODC() then
    return ERROR_DS_OBJ_NOT_FOUND
endif

/* This function cannot be called on a DC for the domain
 * to be removed. */
if DefaultNC().dn = domainDN then
    return ERROR_DS_ILLEGAL_MOD_OPERATION
endif

/* Make sure no DCs still have NC replicas of this domain NC. */
otherNtdsdsa := select one o from ConfigNC() where
    (nTDSDSA in o!objectClass) and
    (domainDN in o!hasMasterNCs or
     domainDN in o!msDS-hasMasterNCs)
if otherNtdsdsa ≠ null then
    return ERROR_DS_NC_STILL_HAS_DSAS
endif

/* Find the crossRef object for the domain named by domainDN. */
cr := select one o from ConfigNC() where
    (o!nCName = domainDN) and (crossRef in o!objectClass)

if cr = null then
    return ERROR_DS_NO_CROSSREF_FOR_NC
endif

/* Make sure we are the Domain Naming FSMO role owner */
if GetFSMORoleOwner(FSMO_DOMAIN_NAMING) ≠ DSAObj() then
    /* We are not the Domain Naming FSMO role owner */
    return ERROR_DS_OBJ_NOT_FOUND
else
    /* We are the Domain Naming FSMO role owner. If the Config NC
     * has not replicated at least once since startup, our ownership
     * of the NC is not considered to be verified, and we exit with
     * an error. */
    if not HasNCReplicated(ConfigNC()) then
        return ERROR_DS_ROLE_NOT_VERIFIED;
    endif
endif

```

```

endif

if (not AccessCheckObject(cr, RIGHT_DS_DELETE)) and
    (not AccessCheckObject(cr.parent, RIGHT_DS_DELETE_CHILD)) then
    return ERROR_ACCESS_DENIED
endif

RemoveObj(cr)
return 0

```

4.1.18 IDL_DRSRemoveDsServer (Opnum 14)

The **IDL_DRSRemoveDsServer** method removes the representation (also known as the metadata) of a DC from the directory.

```

ULONG IDL_DRSRemoveDsServer(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_RMSVRREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_RMSVRREPLY* pmsgOut
);

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message. MUST be set to 1, because that is the only version supported.

pmsgIn: The pointer to the request message.

pdwOutVersion: the pointer to the version of the response message. The value is always 1, because that is the only version supported.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful or a Windows error code if a failure occurs.

4.1.18.1 Method-Specific Concrete Types

4.1.18.1.1 DRS_MSG_RMSVRREQ

The **DRS_MSG_RMSVRREQ** union defines the request messages sent to the [IDL_DRSRemoveDsServer](#) method. Only one version, identified by `dwInVersion = 1`, is currently defined.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_RMSVRREQ_V1 V1;
} DRS_MSG_RMSVRREQ;

```

V1: The version 1 request.

4.1.18.1.2 DRS_MSG_RMSVRREQ_V1

The **DRS_MSG_RMSVRREQ_V1** structure defines a request message sent to the [IDL DRSRemoveDsServer](#) method.

```
typedef struct {
    [string] LPWSTR ServerDN;
    [string] LPWSTR DomainDN;
    BOOL fCommit;
} DRS_MSG_RMSVRREQ_V1;
```

ServerDN: The DN of the server object of the DC to remove.

DomainDN: The DN of the NC root of the domain that the DC to be removed belongs to. May be set to NULL if the client does not want the server to compute the value of pmsgOut^.V1.DomainDN.

fCommit: True if the DC's metadata should actually be removed from the directory. False if the metadata should not be removed. (This is used by a client that wants to determine the value of pmsgOut^.V1.fLastDcInDomain without altering the directory.)

4.1.18.1.3 DRS_MSG_RMSVRREPLY

The **DRS_MSG_RMSVRREPLY** union defines the response messages received from the [IDL DRSRemoveDsServer](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_RMSVRREPLY_V1 V1;
} DRS_MSG_RMSVRREPLY;
```

V1: The version 1 response.

4.1.18.1.4 DRS_MSG_RMSVRREPLY_V1

The **DRS_MSG_RMSVRREPLY_V1** structure defines a response message received from the [IDL DRSRemoveDsServer](#) method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

```
typedef struct {
    BOOL fLastDcInDomain;
} DRS_MSG_RMSVRREPLY_V1;
```

fLastDcInDomain: True if the DC is the last DC in its domain, and pmsgIn^.V1.DomainDN was set to the DN of the NC root of the domain to which the DC belongs. False otherwise.

4.1.18.2 Server Behavior of the IDL_DRSRemoveDsServer Method

Informative summary of behavior: Removes the metadata defining a DC, which consists of the tree of objects rooted at the DC's [nTDSDSA](#) object as well as the [rIDSet](#) objects and DRS SPNs associated with the DC's [computer](#) object. This method is typically used when a DC is demoted. As part of the demotion process, the DC being demoted calls this method on another DC (either in the same domain, if such a DC exists, or in the parent domain, if there are no other DCs in the same domain but there is a parent domain) to remove the metadata of the DC being demoted from the forest. Alternatively, if a DC is removed from the domain without being properly demoted (for example, if the DC suffers a fatal hardware failure), a client may make this call to remove the metadata of the now-nonexistent DC. When pmsgIn^.V1.DomainDN is specified, this function also computes whether the DC is the last replica of its default domain NC.

The behavior of this function has two variants. If pmsgIn^.V1.fCommit is false, the function is read-only with regards to abstract state, that is, it does not make any changes to the directory contents. In this mode, the main purpose of the function is to compute pmsgOut^.V1.fLastDcInDomain (and so there is little point to calling the function in this mode without setting pmsgIn^.V1.DomainDN). For example, prior to removing the DC's metadata, a client application might want to determine whether any DCs would be left in the domain, so that it can warn the user if the user is removing the last DC in the domain.

When pmsgIn^.V1.fCommit is true, the second variant of the behavior is performed. In this mode, the function actually removes the DC metadata. The pmsgOut^.V1.fLastDcInDomain value is also computed in this mode (provided that pmsgIn^.V1.DomainDN was passed in). This method undoes the effects of the [IDL_DRSAddEntry](#) method when **IDL_DRSAddEntry** is used to create an [nTDSDSA](#) object. The removal of the DC's metadata signals other DCs in the forest that this particular DC no longer exists.

```
ULONG
IDL_DRSRemoveDsServer(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_RMSVRREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_RMSVRREPLY *pmsgOut);

serverDn: unicodestring
domainDn: unicodestring
ntdsdsa: DSName
otherNtdsdsa: DSName
spnsToRemove: set of unicodestring
computerDn: unicodestring
computer: DSName
objectsToDelete: set of DSName

serverDn := pmsgIn^.V1.ServerDN
domainDn := pmsgIn^.V1.DomainDN

pdwOutVersion^ := 1

/* Basic parameter validation */
```

```

if dwInVersion # 1 then
    return ERROR_INVALID_PARAMETER
endif

if serverDn = null or serverDn = "" then
    return ERROR_INVALID_PARAMETER
endif

/* Note that DomainDN may be null, but it can not be empty. */
if domainDn = "" then
    return ERROR_INVALID_PARAMETER
endif

/* Originating updates are not allowed on RODC */
if AmIRODC() then
    return ERROR_DS_OBJ_NOT_FOUND
endif

ntdsdsa := DescendantObject([dn: serverDn], "CN=NTDS Settings,")
if ntdsdsa = null then
    return ERROR_DS_NO_SUCH_OBJECT
endif

/* Compute fLastDcInDomain if domainDn is non-null. */
if domainDn # null then
    otherNtdsdsa := select one o from subtree ConfigNC() where
        (o!objectCategory = nTDSDSA)
        and
        (domainDn in o!hasMasterNCs or domainDn in o!msDS-hasMasterNCs)
        and
        (o # ntdsdsa)
    if otherNtdsdsa = null then
        pmsgOut^.V1.fLastDcInDomain = true
    else
        pmsgOut^.V1.fLastDcInDomain = false
    endif
endif

/* If nothing to commit, processing is complete. */
if not pmsgIn^.V1.fCommit then
    return 0
endif

/* Perform the actual DC metadata removal. */

/* Locate the computer object for the DC's account. */
computerDn := ntdsdsa!serverReference
computer := null
if computerDn # null then
    computer := GetDSNameFromDN(computerDn)
endif

/* Remove the subtree of objects rooted at the DC's ntdsDsa object.*/
objectsToDelete := select all o from subtree ntdsdsa where (true)

if not AccessCheckObject(ntdsdsa, RIGHT_DS_DELETE_TREE) then
    return ERROR_ACCESS_DENIED
endif

```



```

foreach o in objectsToDelete
    RemoveObj(o)
endfor

/* If the DC's computer account exists, remove rIDSet objects and
* remove the DRS SPNs from the computer object. */

if computer ≠ null then
    foreach r in computer!rIDSetReferences
        if (not AccessCheckObject(r, RIGHT_DS_DELETE)) and
            (not AccessCheckObject(r.parent, RIGHT_DS_DELETE_CHILD)) then
            return ERROR_ACCESS_DENIED
        endif

        RemoveObj(r)
    endfor

    foreach spn in computer!servicePrincipalName
        if StartsWith(spn, "ldap/") or
            StartsWith(spn, "GC/") or
            StartsWith(spn, "E3514235-4B06-11D1-AB04-00C04FC2DCD2/") then
            spnsToRemove := spnsToRemove + {spn}
        endif
    endfor

    if not AccessCheckAttr(computer, servicePrincipalName,
        RIGHT_DS_WRITE_PROPERTY) then
        return ERROR_ACCESS_DENIED
    endif

    computer!servicePrincipalName :=
        computer!servicePrincipalName - spnsToRemove
    endif

return 0

```

4.1.19 IDL_DRSReplicaAdd (Opnum 5)

The **IDL_DRSReplicaAdd** method adds a replication source reference for the specified NC.

```

ULONG IDL_DRSReplicaAdd(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)]
        DRS_MSG_REPADD* pmsgAdd
);

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwVersion: The version of the request message.

pmsgAdd: The pointer to the request message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.19.1 Method-Specific Concrete Types

4.1.19.1.1 DRS_MSG_REPADD

The **DRS_MSG_REPADD** union defines request messages that are sent to the [IDL DRSReplicaAdd](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_REPADD_V1 V1;
    [case(2)]
        DRS_MSG_REPADD_V2 V2;
} DRS_MSG_REPADD;
```

V1: The version 1 request (Windows 2000 and later).

V2: The version 2 request (Windows 2000 and later; a superset of V1).

4.1.19.1.2 DRS_MSG_REPADD_V1

The **DRS_MSG_REPADD_V1** structure defines a request message sent to the [IDL DRSReplicaAdd](#) method. This request version is supported by Windows 2000 and later releases of Windows.

```
typedef struct {
    [ref] DSNAME* pNC;
    [ref, string] char* pszDsaSrc;
    REPLTIMES rtSchedule;
    ULONG ulOptions;
} DRS_MSG_REPADD_V1;
```

pNC: The NC root of the NC to replicate.

pszDsaSrc: The transport-specific [NetworkAddress](#) of the DC from which to replicate updates.

rtSchedule: The schedule at which to perform periodic replication.

ulOptions: Zero or more [DRS_OPTIONS](#) flags.

4.1.19.1.3 DRS_MSG_REPADD_V2

The **DRS_MSG_REPADD_V2** structure defines a request message sent to the [IDL DRSReplicaAdd](#) method. This request version is a superset of V1 and is supported by Windows 2000 and later releases of Windows.

```
typedef struct {
    [ref] DSNAME* pNC;
```

```

[unique] DSNAME* pSourceDsaDN;
[unique] DSNAME* pTransportDN;
[ref, string] char* pszSourceDsaAddress;
REPLTIMES rtSchedule;
ULONG ulOptions;
} DRS_MSG_REPADD_V2;

```

pNC: The NC root of the NC to replicate.

pSourceDsaDN: The [nTDSDSA](#) object for the DC from which to replicate changes.

pTransportDN: The [interSiteTransport](#) object that identifies the network transport over which replication should be performed.

pszSourceDsaAddress: The transport-specific [NetworkAddress](#) of the DC from which to replicate updates.

rtSchedule: The schedule at which to perform periodic replication.

ulOptions: Zero or more [DRS_OPTIONS](#) flags.

4.1.19.2 Server Behavior of the IDL_DRSReplicaAdd Method

Informative summary of behavior: The server adds a value to the [repsFrom](#) of the specified NC replica. If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously. The client can be an administrative client or another DC. The client includes DRS_WRIT_REP in ulOptions if the specified NC replica is writable at the server. The client includes DRS_SPECIAL_SECRET_PROCESSING in ulOptions if the specified NC replica is a default NC replica on a read-only DC. The server adds a value to [repsFrom](#), the value has replicaFlags derived from ulOptions (see below), a serverAddress equal to pszSourceDsaAddress (pszDsaSrc if V1), and a schedule equal to rtSchedule. If ulOptions contains DRS_ASYNC_REP but not DRS_MAIL_REP or DRS_NEVER_NOTIFY, the server sends a request to the DC specified by pszSourceDsaAddress to add a value to the [repsTo](#) of the specified NC replica by calling [IDL_DRSUpdateRefs](#). Finally, the server begins a replication cycle by sending an [IDL_DRSGetNCChanges](#) request.

```

ULONG
IDL_DRSReplicaAdd(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPADD *pmsgAdd);

options: DRS_OPTIONS
nc: DSName
partitionsObj: DSName
cr: DSName
rf: RepsFrom
msgIn: DRS_MSG_REPADD_V2
updRefs: DRS_MSG_UPDREFS /* See IDL_DRSUpdateRefs structures. */

/* Validate the version */
if dwVersion != 1 and dwVersion != 2 then
    return ERROR_INVALID_PARAMETER
endif
if dwVersion = 1 then

```

```

    msgIn := pmsgAdd^.V1
    msgIn.pszSourceDsaAddress = pmsgAdd^.V1.pszDsaSrc
else
    msgIn := pmsgAdd^.V2
endif

if msgIn.pNC = null
    or msgIn.pszSourceDsaAddress = null
    or msgIn.pszSourceDsaAddress = "" then
        return ERROR_INVALID_PARAMETER
    endif

options := msgIn.ulOptions
nc := msgIn.pNC^

partitionsObj :=
    select one o from children ConfigNC() where o!name = "Partitions"
cr := select o from children partitionsObj where o!nCName = nc
if cr = null then
    return ERROR_DS_DRA_BAD_DN
endif

if AmIRODC() and DRS_WRIT_REP in options then
    return ERROR_INVALID_PARAMETER
endif

if ObjExists(nc) then
    if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
        return ERROR_DS_DRA_ACCESS_DENIED
    endif
else
    if not AccessCheckCAR(DefaultNC(), DS-Replication-Manage-Topology)
        then
        return ERROR_DS_DRA_ACCESS_DENIED
    endif
endif

if DRS_ASYNC_OP in options then
    Asynchronous Processing: Initiate a logical thread of control
    to process the remainder of this request asynchronously
    return 0
endif

if ObjExists(nc) then
    if (IT_WRITE in nc!instanceType) ≠ (DRS_WRIT_REP in options) then
        return ERROR_DS_DRA_BAD_NC
    endif
    /* Disallow addition if server already replicates from this
    * source */
    if (select one v from nc!repsFrom
        where v.serverAddress = msgIn.pszSourceDsaAddress) ≠ null
        then
        return ERROR_DS_DRA_DN_EXISTS
    endif
endif

if msgIn.pSourceDsaDN ≠ null
    and (not ObjExists(msgIn.pSourceDsaDN^))

```

```

        or not nTDSDSA in msgIn.pSourceDsaDN^!objectClass
        or GetObjectNC(msgIn.pSourceDsaDN^) # ConfigNC()) then
    return ERROR_INVALID_PARAMETER
endif

if msgIn.pTransportDN # null
    and (not ObjExists(msgIn.pTransportDN^)
        or not interSiteTransport in msgIn.pTransportDN^!objectClass
        or GetObjectNC(msgIn.pTransportDN^) # ConfigNC()) then
    return ERROR_INVALID_PARAMETER
endif

/* Construct RepsFrom value. */
if msgIn.pSourceDsaDN # null then
    rf.uuidDsa := msgIn.pSourceDsaDN^!objectGUID
endif
if msgIn.pTransportDN # null then
    rf.uuidTransportObj := msgIn.pTransportDN^!objectGUID
endif
rf.replicaFlags := msgIn.ulOptions ∩ {DRS_DISABLE_AUTO_SYNC,
    DRS_DISABLE_PERIODIC_SYNC, DRS_INIT_SYNC, DRS_MAIL_REP,
    DRS_NEVER_NOTIFY, DRS_PER_SYNC, DRS_TWOWAY_SYNC,
    DRS_USE_COMPRESSION, DRS_WRIT_REP,
    DRS_SPECIAL_SECRET_PROCESSING }
rf.schedule := msgIn.rtSchedule^
rf.serverAddress := msgIn.pszSourceDsaAddress^
rf.timeLastAttempt := current time

if msgIn.ulOptions ∩ {DRS_ASYNC_REP, DRS_NEVER_NOTIFY, DRS_MAIL_REP}
    = {DRS_ASYNC_REP} then
    /* Enable replication notifications by requesting the server DC
    * to add a repsTo for this DC. */
    updRefs.pNC^ := ADR(nc)
    updRefs.pszDsaDest := NetworkAddress of this DC
    updRefs.uuidDsaDest := dc.serverGuid
    updRefs.ulOptions := {DRS_ASYNC_OP, DRS_ADD_REF, DRS_DEL_REF}
    if DRS_WRIT_REP in msgIn.ulOptions then
        updRefs.ulOptions := updRefs.ulOptions + {DRS_WRIT_REP}
    endif
    Send updRefs request by calling IDL_DRSUpdateRefs() on server
    msgIn.pszSourceDsaAddress^
endif

Perform a replication cycle as a client of IDL_DRSGetNCChanges. Call
ReplicateNCRequestMsg (see Section 4.10.4.1) to form the first
request and send it. If not DRS_MAIL_REP in msgIn.ulOptions, then
wait for the response, process it, send the next request, etc. until
the replication cycle is complete.

err := 0 or the result of processing the last replication response

return err

```

4.1.20 IDL_DRSReplicaDel (Opnum 6)

The **IDL_DRSReplicaDel** method deletes a replication source reference for the specified NC.

```

ULONG IDL_DRSReplicaDel(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)]
        DRS_MSG_REPDEL* pmsgDel
);

```

hDrs: The RPC context handle returned by [IDL_DRSBind](#).

dwVersion: The version of the request message.

pmsgDel: The pointer to the request message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.20.1 Method-Specific Concrete Types

4.1.20.1.1 DRS_MSG_REPDEL

The **DRS_MSG_REPDEL** union defines the request messages sent to the [IDL_DRSReplicaDel](#) method. Only one version, identified by dwVersion = 1, is currently defined.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_REPDEL_V1 V1;
} DRS_MSG_REPDEL;

```

V1: The version 1 request.

4.1.20.1.2 DRS_MSG_REPDEL_V1

The **DRS_MSG_REPDEL_V1** structure defines a request message sent to the [IDL_DRSReplicaDel](#) method.

```

typedef struct {
    [ref] DSNAME* pNC;
    [string] char* pszDsaSrc;
    ULONG ulOptions;
} DRS_MSG_REPDEL_V1;

```

pNC: The pointer to [DSName](#) of the root of an NC replica on the server.

pszDsaSrc: The transport-specific [NetworkAddress](#) of a DC.

ulOptions: The [DRS_OPTIONS](#) flags.

4.1.20.2 Server Behavior of the IDL_DRSReplicaDel Method

Informative summary of behavior: When DRS_NO_SOURCE is not specified, the server removes a value from [repsFrom](#) of the specified NC replica. If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously. The client must include DRS_WRIT_REP in ulOptions if the specified NC replica is a writable replica. The server removes the value from [repsFrom](#) whose serverAddress matches pszDsaSrc. If ulOptions does not contain DRS_LOCAL_ONLY, the server sends a request to the DC specified by pszDsaSrc to remove this DC from the values in [repsTo](#) of the specified NC replica by calling [IDL_DRSUpdateRefs](#).

When DRS_NO_SOURCE is specified, the server expunges the NC replica and all its children. If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously. The client must include DRS_WRIT_REP in ulOptions if the specified NC replica is writable. If ulOptions contains DRS_ASYNC_REP, the server expunges the objects asynchronously.

```
ULONG
IDL_DRSReplicaDel(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPDEL *pmsgDel);
options: DRS_OPTIONS
nc: DSName
rf: RepsFrom
msgIn: DRS_MSG_REPDEL V1
updRefs: DRS_MSG_UPDREFS /* See IDL_DRSUpdateRefs structures. */
/* Validate dwVersion */
if dwInVersion != 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgDel^.V1
/* Validate the NC */
if msgIn.pNC = null then
    return ERROR_INVALID_PARAMETER
endif
nc := msgIn.pNC^
options := msgIn.ulOptions
if not ObjExists(nc) then
    return ERROR_DS_DRA_BAD_DN
endif
if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
    return ERROR_DS_DRA_ACCESS_DENIED
endif
if DRS_ASYNC_OP in options then
    Asynchronous Processing: Initiate a logical thread of control
    to process the remainder of this request asynchronously
    return 0
endif
options := msgIn.ulOptions
if (IT_WRITE in nc!instanceType) != (DRS_WRIT_REP in options) then
    return ERROR_DS_DRA_BAD_NC
endif
if DRS_NO_SOURCE in options then
    /* Expunging local copy of an NC. */
    /* NC must not replicate from any other DC. */
    if (select one v from nc!repsFrom where (true)) != null then
        return ERROR_INVALID_PARAMETER
    endif
endif
```

```

/* NC should not replicate to any other DC. */
if (select one v from nc!repsTo where (true)) ≠ null
    and (not DRS_REF_OK in options) then
        return ERROR_DS_DRA_OBJ_IS_REP_SOURCE
    endif
/* Do not permit removal of important NCs. */
if IT_WRITE in nc!instanceType
    and (nc = DefaultNC()
        or nc = ConfigNC()
        or nc = SchemaNC()) then
        return ERROR_INVALID_PARAMETER
    endif
if DRS_ASYNC_REP in options then
    Asynchronous Processing: Initiate a logical thread of control
    to process the remainder of this request asynchronously
    return 0
endif
/* Expunge the subtree rooted at dn, inclusive. */
foreach o in (select all v from subtree nc where (true))
    Expunge(o)
endfor
return 0
else /* not DRS_NO_SOURCE in options */
/* Removing a single source from repsFrom, but leaving NC replica
* on DC. */
if msgIn.pszDsaSrc = null or msgIn.pszDsaSrc^ = "" then
    return ERROR_INVALID_PARAMETER
endif
rf := select one v from nc!repsFrom
    where (v.serverAddress = msgIn.pszDsaSrc)
if rf = null then
    return ERROR_DS_DRA_NO_REPLICA
endif
nc!repsFrom := nc!repsFrom - {rf}
if (not DRS_LOCAL_ONLY in options)
    and (not DRS_MAIL_REP in rf.options) then
/* Disable replication notifications by requesting the DC
* specified by msgIn.pszDsaSrc to remove this DC's
* entry from its msgIn.pNC^!repsTo. */
updRefs.pNC^ := ADR(nc)
updRefs.pszDsaDest := NetworkAddress of this DC
updRefs.uuidDsaDest := dc.serverGuid
updRefs.ulOptions := {DRS_ASYNC_OP, DRS_DEL_REF}
if DRS_WRIT_REP in msgIn.ulOptions then
    updRefs.ulOptions := updRefs.ulOptions + {DRS_WRIT_REP}
endif
Send updRefs request by calling IDL_DRSUpdateRefs() on server
    msgIn.pszDsaSrc^
endif
return 0
endif
endif

```

4.1.21 IDL_DRSReplicaDemotion (Opnum 26)

The **IDL_DRSReplicaDemotion** method replicates off all changes to the specified NC and moves any FSMOs held to another server. This function is supported only by AD/LDS.


```

ULONG IDL_DRSReplicaDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_REPLICA_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_REPLICA_DEMOTIONREPLY* pmsgOut
);

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.21.1 Method-Specific Concrete Types

4.1.21.1.1 DRS_MSG_REPLICA_DEMOTIONREQ

The **DRS_MSG_REPLICA_DEMOTIONREQ** union defines the request messages sent to the [IDL_DRSReplicaDemotion](#) method. Only one version, identified by dwInVersion = 1, is currently defined.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_REPLICA_DEMOTIONREQ_V1 V1;
} DRS_MSG_REPLICA_DEMOTIONREQ;

```

V1: The version 1 request. Only one version is defined.

4.1.21.1.2 DRS_MSG_REPLICA_DEMOTIONREQ_V1

The **DRS_MSG_REPLICA_DEMOTIONREQ_V1** structure defines a request message sent to the [IDL_DRSReplicaDemotion](#) method.

```

typedef struct {
    DWORD dwFlags;
    UUID uuidHelperDest;
    DSNAME* pNC;
} DRS_MSG_REPLICA_DEMOTIONREQ_V1;

```

dwFlags: Zero or more of the following bit flags.

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
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	X	X	X	X	X	X	X

X: Unused. MUST be zero and ignored.

T (DS_REPLICA_DEMOTE_TRY_ALL_SRCS): MUST be set.

uuidHelperDest: MUST be NULL GUID.

pNC: The [DSNAME](#) of the NC to replicate off.

4.1.21.1.3 DRS_MSG_REPLICA_DEMOTIONREPLY

The **DRS_MSG_REPLICA_DEMOTIONREPLY** union defines the response messages received from the [IDL DRSReplicaDemotion](#) method. Only one version, identified by `pdwOutVersion^ = 1`, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_REPLICA_DEMOTIONREPLY_V1 V1;
} DRS_MSG_REPLICA_DEMOTIONREPLY;
```

V1: The version 1 reply.

4.1.21.1.4 DRS_MSG_REPLICA_DEMOTIONREPLY_V1

The **DRS_MSG_REPLICA_DEMOTIONREPLY_V1** structure defines a response message received from the [IDL DRSReplicaDemotion](#) method.

```
typedef struct {
    DWORD dwOpError;
} DRS_MSG_REPLICA_DEMOTIONREPLY_V1;
```

dwOpError: The Win32 error code, as specified in [\[MS-ERREF\]](#) section 3.0.

4.1.21.2 Method-Specific Abstract Types and Procedures

4.1.21.2.1 ReplicationPartners()

```
procedure ReplicationPartners(nc: DSNAME): sequence of DSNAME
```

The DC D executing this procedure hosts a portion of forest F. This procedure computes the set of all DCs in F that host the specified NC, excluding D. It returns this set as a sequence in an arbitrary order.

4.1.21.2.2 BindToDSA()

```
procedure BindToDSA(dsa: DSNAME): DRS_HANDLE
```

Procedure BindToDSA() establishes an RPC connection to the target DC represented by its DSA object. It also performs the [IDL_DRSBind](#) call. It returns the RPC handle on success, or null on failure.

4.1.21.2.3 UnbindFromDSA()

```
procedure UnbindFromDSA(hDRS: DRS_HANDLE)
```

The UnbindFromDSA() procedure closes the RPC connection that was established by the [BindToDSA](#) procedure.

4.1.21.2.4 AbandonAllFSMORoles()

```
procedure AbandonAllFSMORoles(nc: DSNAME): DWORD
```

The AbandonAllFSMORoles() procedure abandons any FSMO roles represented in the supplied NC that are held by this DC. The new holder of the FSMO roles is arbitrary. AbandonAllFSMORoles() returns a Win32 error value.

```
targetDSAs: sequence of DSNAME
fsmoContainer: DSNAME
ret: DWORD
bGivenAway: boolean
i: integer
msgReq: DRS_MSG_GETCHGREQ_V8
msgUpd: DRS_MSG_GETCHGREPLY_V6
hDRS: DRS_HANDLE

if nc = ConfigNC() then
    /* check domain naming FSMO role */
    fsmoContainer := DescendantObject(ConfigNC(), "CN=Partitions,")
else if nc = SchemaNC() then
    /* check schema master FSMO role */
    fsmoContainer := SchemaNC()
else
    /* application NCs don't hold FSMOs */
    return ERROR_SUCCESS
endif

/* check if we hold the fsmo */
if fsmoContainer!fsmoRoleOwner ≠ DSAObj() then
    /* we don't own the role! All's well */
    return ERROR_SUCCESS
endif

/* yes, we own the role! Let's give it away */
bGivenAway := false
targetDSAs := ReplicationPartners(nc)
i := 0
```

```

while not bGivenAway
  if i ≥ targetDSAs.length then
    /* no more replication partners that would take our FSMO! */
    return ERROR_DS_UNABLE_TO_SURRENDER_ROLES
  endif
  hDRS := BindToDSA(possibleTargetDSAs[i])
  if hDRS ≠ null then
    /* the targetDSA appears to be up. Let's try to transfer the
    * role */
    /* Perform an IDL_DRSGetNCChanges(EXOP_FSMO_ABANDON_ROLE) call */
    msgReq.uuidDsaObjDest := dc.serverGuid
    msgReq.pNC := ADDR(fsmoContainer)
    msgReq.ulFlags := DRS_WRIT_REP
    msgReq.ulExtendedOp := EXOP_FSMO_ABANDON_ROLE
    ret :=
      IDL_DRSGetNCChanges(hDRS, 8, ADDR(msgReq), 6, ADDR(msgUpd))
    if ret = ERROR_SUCCESS then
      /* yes! We got rid of it */
      bGivenAway := true
    endif
    UnbindFromDSA(hDRS)
  endif
  i := i + 1
endwhile
/* if we got here, then we managed to give away the role */
return ERROR_SUCCESS

```

4.1.21.2.5 ReplicateOffChanges()

```

procedure ReplicateOffChanges(nc: DSNAME): DWORD

```

The ReplicateOffChanges() procedure replicates all local changes in the NC to a randomly selected replication partner.

```

targetDSAs: sequence of DSNAME
ret: DWORD
bReplicated: boolean
i: integer
msgSyncReq: DRS_MSG_REPSYNC_V1
msgAddReq: DRS_MSG_REPADD_V2
hDRS: DRS_HANDLE
bReplicated := false
targetDSAs := ReplicationPartners(nc)
i := 0
while not bReplicated
  if i ≥ targetDSAs.length then
    /* no more replication partners that host the NC! */
    return ERROR_DS_CANT_FIND_DSA_OBJ
  endif
  hDRS := BindToDSA(possibleTargetDSAs[i])
  if hDRS ≠ null then
    /* the targetDSA appears to be up. Let's try to replicate to
    * it */
    /* Invoke IDL_DRSReplicaSync to get changes from us */
    msgSyncReq.pszDsaSrc := NetworkAddress of targetDSA

```

```

msgSyncReq.uuidDsaSrc := dc.serverGuid
msgSyncReq.pNC := ADDR(nc)
msgSyncReq.ulOptions := DRS_WRIT_REP
ret := IDL_DRSReplicaSync(hDRS, &msgSyncReq, 1)
if ret = ERROR_DS_DRA_NO_REPLICA then
    /* the targetDSA does not currently have replication agreement
       (repsFrom) with this DC. Tell it to add one */
    msgAddReq.pNC := ADDR(nc)
    msgAddReq.pszSourceDsaAddress := NetworkAddress of this DC
    msgAddReq.ulOptions := DRS_WRIT_REP
    ret := IDL_DRSReplicaAdd(hDRS, &msgAddReq, 2)
endif
UnbindFromDSA(hDRS)
if ret = ERROR_SUCCESS then
    /* we did it! */
    bReplicated := true
endif
endif
i := i + 1
endwhile
/* if we got here, then we successfully replicated off our changes */
return ERROR_SUCCESS

```

4.1.21.3 Server Behavior of the IDL_DRSReplicaDemotion Method

Informative summary of behavior: For a given NC, the IDL_DRSReplicaDemotion method replicates out any changes that had not previously been replicated out. It also abandons any NC-specific FSMO roles that are owned by this DC. This function accomplishes nothing when the DC being demoted is the last DC in the forest.

```

ULONG
IDL_DRSReplicaDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_REPLICA_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_REPLICA_DEMOTIONREPLY* pmsgOut
)
msgIn: DRS_MSG_REPLICA_DEMOTIONREQ_V1
ret: DWORD
nc: DSNAME
if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^V1
if msgIn.pNC = null or
    msgIn.dwFlags ≠ DS_REPLICA_DEMOTE_TRY_ALL_SRCS then
    return ERROR_INVALID_PARAMETER
endif
if not IsMemberOfBuiltinAdminGroup() then
    /* only BA is allowed to demote an AD/LDS service */
    return ERROR_ACCESS_DENIED
endif
nc := msgIn.pNC^

```

```

ret := AbandonAllFSMORoles(nc)
if ret = ERROR_SUCCESS then
    ret := ReplicateOffChanges(nc)
endif
if ret = ERROR_SUCCESS then
    /* mark instanceType as going and not coming */
    nc!instanceType := nc!instanceType + {IT_NC_GOING} - {IT_NC_COMING}
    /* remove any repsFrom */
    nc!repsFrom := null
endif
pmsgOut^.dwOpError := ret
pdwMsgOut^ := 1
return ERROR_SUCCESS

```

4.1.22 IDL_DRSReplicaModify (Opnum 7)

The **IDL_DRSReplicaModify** method updates the value for [repsFrom](#) for the NC replica.

```

ULONG IDL_DRSReplicaModify(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)]
        DRS_MSG_REPMOD* pmsgMod
);

```

hDrs: The RPC context handle returned by [IDL_DRSBind](#).

dwVersion: The version of the request message.

pmsgMod: The pointer to the request message.

Return Values: 0 if successful, or a Windows error code if a failure occurs.

4.1.22.1 Method-Specific Concrete Types

4.1.22.1.1 DRS_MSG_REPMOD

The **DRS_MSG_REPMOD** union defines the request messages for the [IDL_DRSReplicaModify](#) method. Only one version, identified by `dwVersion = 1`, is currently defined.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_REPMOD_V1 V1;
} DRS_MSG_REPMOD;

```

V1: The version 1 request.

4.1.22.1.2 DRS_MSG_REPMOD_V1

The **DRS_MSG_REPMOD_V1** structure defines a request message for the [IDL_DRSReplicaModify](#) method.

```
typedef struct {
    [ref] DSNAME* pNC;
    UUID uuidSourceDRA;
    [unique, string] char* pszSourceDRA;
    REPLTIMES rtSchedule;
    ULONG ulReplicaFlags;
    ULONG ulModifyFields;
    ULONG ulOptions;
} DRS_MSG_REPMOD_V1;
```

pNC: The pointer to [DSName](#) of the root of an NC replica on the server.

uuidSourceDRA: The DSA GUID.

pszSourceDRA: The transport-specific [NetworkAddress](#) of a DC.

rtSchedule: The periodic replication schedule.

ulReplicaFlags: The [DRS_OPTIONS](#) flags for the [repsFrom](#) value.

ulModifyFields: The fields to update.

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
U	U	U	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
F	A	S																													

X: Unused. MUST be zero and ignored.

UF (DRS_UPDATE_FLAGS): Updates the flags associated with the server.

UA (DRS_UPDATE_ADDRESS): Updates the transport-specific address associated with the server.

US (DRS_UPDATE_SCHEDULE): Updates the replication schedule associated with the server.

ulOptions: The [DRS_OPTIONS](#) flags for executing this method.

4.1.22.2 Server Behavior of the IDL_DRSReplicaModify Method

Informative summary of behavior: The server replaces fields in the [repsFrom](#) of the specified NC replica. If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously. The client must include DRS_WRIT_REP in ulOptions if the specified NC replica is a full replica. The server alters timeLastSuccess and consecutiveErrors and optionally replaces (as specified by ulModifyFields) serverAddress, schedule, and replicaFlags in [repsFrom](#) with the corresponding value from pszSourceDRA, rtSchedule, and ulReplicaFlags.

```

ULONG
IDL_DRSReplicaModify(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)]
        DRS_MSG_REPMOD *pmsgMod);
options: DRS_OPTIONS
nc: DSName
rf: RepsFrom
msgIn: DRS_MSG_REPMOD_V1
/* Validate the version */
if dwInVersion != 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgMod^.V1
/* Validate input parameters */
if msgIn.pNC = null
    or msgIn.pNC^ = ""
    or (msgIn.pszSourceDRA = null
        and msgIn.uuidSourceDRA = null)
    or (DRS_UPDATE_ADDRESS in {msgIn.ulModifyFields}
        and (msgIn.pszSourceDRA = null
            or msgIn.pszSourceDRA = ""))
    or (DRS_UPDATE_SCHEDULE in {msgIn.ulModifyFields}
        and msgIn.rtSchedule = null)
    or {msgIn.ulmodifyFields} -
        {DRS_UPDATE_ADDRESS, DRS_UPDATE_SCHEDULE, DRS_UPDATE_FLAGS}
        != {} then
    return ERROR_INVALID_PARAMETER
endif
/* Validate the specified NC */
options := msgIn.ulOptions
nc := msgIn.pNC^
if not ObjExists(nc) then
    return ERROR_DS_DRA_BAD_DN
endif
if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
    return ERROR_DS_DRA_ACCESS_DENIED
endif
if DRS_ASYNC_OP in options then
    Asynchronous Processing: Initiate a logical thread of control
    to process the remainder of this request asynchronously
    return 0
endif
/* Validate the specified NC's writability. */
if (IT_WRITE in nc!instanceType) != (DRS_WRIT_REP in options) then
    return ERROR_DS_DRA_BAD_NC
endif

/* Find the specified repsFrom. */
rf := select one v from nc!repsFrom
    where (v.serverAddress = msgIn.pszDsaSrc)
if rf = null then
    return ERROR_DS_DRA_NO_REPLICA
endif
/* Update the specified repsFrom. */
nc!repsFrom := nc!repsFrom - {rf}
rf.timeLastSuccess := current time

```



```

rf.consecutiveFailures := 0
if DRS_UPDATE_ADDRESS in {msgIn.ulModifyFields} then
    rf.serverAddress := msgIn.pszSourceDRA
endif
if DRS_UPDATE_SCHEDULE in {msgIn.ulModifyFields} then
    rf.schedule := msgIn.rtSchedule
endif
if DRS_UPDATE_FLAGS in {msgIn.ulModifyFields} then
    rf.replicaFlags := msgIn.ulReplicaFlags
endif
nc!repsFrom := nc!repsFrom + {rf}
return 0

```

4.1.23 IDL_DRSReplicaSync (Opnum 2)

The **IDL_DRSReplicaSync** method triggers replication from another DC.

```

ULONG IDL_DRSReplicaSync(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)]
    DRS_MSG_REPSYNC* pmsgSync
);

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwVersion: The version of the request message.

pmsgSync: The pointer to the request message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.23.1 Method-Specific Concrete Types

4.1.23.1.1 DRS_MSG_REPSYNC

The **DRS_MSG_REPSYNC** union defines the request messages sent to the [IDL_DRSReplicaSync](#) method. Only one version, identified by `dwVersion = 1`, is currently defined.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_REPSYNC_V1 V1;
} DRS_MSG_REPSYNC;

```

V1: The version 1 request.

4.1.23.1.2 DRS_MSG_REPSYNC_V1

The **DRS_MSG_REPSYNC_V1** structure defines a request message sent to the [IDL_DRSReplicaSync](#) method.

```
typedef struct {
    [ref] DSNAME* pNC;
    UUID uuidDsaSrc;
    [unique, string] char* pszDsaSrc;
    ULONG ulOptions;
} DRS_MSG_REPSYNC_V1;
```

pNC: The pointer to [DSName](#) of the root of an NC replica on the server.

uuidDsaSrc: A DSA GUID.

pszDsaSrc: The transport-specific [NetworkAddress](#) of a DC.

ulOptions: The [DRS_OPTIONS](#) flags.

4.1.23.2 Server Behavior of the IDL_DRSReplicaSync Method

Informative summary of behavior: The server starts or resumes a replication cycle by sending an [IDL_DRSGetNCChanges](#) request to the specified DC. If ulOptions contains DRS_ASYNC_OP, the server performs this operation asynchronously.

```
ULONG
IDL_DRSReplicaSync(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)]
        DRS_MSG_REPSYNC *pmsgSync);

options: DRS_OPTIONS
nc: DSName
rf: sequence of RepsFrom
msgIn: DRS_MSG_REPSYNC_V1

/* Validate the version */
if dwVersion != 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgSync^.V1

/* Validate input params */
options := msgIn.ulOptions
if msgIn.pNC = null
    or (not DRS_SYNC_ALL in options
        and msgIn.uuidDsaSrc = null
        and msgIn.pszDsaSrc = null) then
    return ERROR_INVALID_PARAMETER
endif

/* Validate the specified NC. */
nc := msgIn.pNC^
if not ObjExists(nc) then
    return ERROR_DS_DRA_BAD_DN
endif

if AccessCheckCAR(nc, DS-Replication-Synchronize) then
```

```

    return ERROR_DS_DRA_ACCESS_DENIED
endif

if DRS_ASYNC_OP in options then
    Asynchronous Processing: Initiate a logical thread of control
    to process the remainder of this request asynchronously
    return 0
endif

rf := select all v in nc!repsFrom
    where DRS_SYNC_ALL in options
        or (DRS_SYNC_BYNAME in options
            and v.uuidDsa = msgIn.pszDsaSrc)
        or (not DRS_SYNC_BYNAME in options
            and v.uuidDsa = msgIn.uuidDsaSrc)
if rf = null then
    return ERROR_DS_DRA_NO_REPLICA
endif

foreach r in rf
    /* Replicate nc from the DC specified by r.uuidDsa. */
    Perform a replication cycle as a client of IDL_DRSGetNCChanges.
    Call ReplicateNCRequestMsg (see Section 4.10.4.1) to form the
    first request and send it. If not DRS_MAIL_REP in r.options,
    then wait for the response, process it, send the next request,
    etc., until the replication cycle is complete.
endfor

return 0

```

4.1.24 IDL_DRSReplicaVerifyObjects (Opnum 22)

The **IDL_DRSReplicaVerifyObjects** method verifies the existence of objects in an NC replica by comparing them against a replica of the same NC on a reference DC, optionally deleting any objects that do not exist on the reference DC.

```

ULONG IDL_DRSReplicaVerifyObjects(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)]
        DRS_MSG_REPVERIFYOBJ* pmsgVerify
);

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwVersion: The version of the request message.

pmsgVerify: The pointer to the request message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.24.1 Method-Specific Concrete Types

4.1.24.1.1 DRS_MSG_REPVERIFYOBJ

The **DRS_MSG_REPVERIFYOBJ** union defines the request messages sent to the [IDL DRSReplicaVerifyObjects](#) method. Only one version, identified by dwVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_REPVERIFYOBJ_V1 V1;
} DRS_MSG_REPVERIFYOBJ;
```

V1: The version 1 request.

4.1.24.1.2 DRS_MSG_REPVERIFYOBJ_V1

The **DRS_MSG_REPVERIFYOBJ_V1** structure defines a request message sent to the [IDL DRSReplicaVerifyObjects](#) method.

```
typedef struct {
    [ref] DSNAME* pNC;
    UUID uuidDsaSrc;
    ULONG ulOptions;
} DRS_MSG_REPVERIFYOBJ_V1;
```

pNC: The NC to verify.

uuidDsaSrc: The [objectGUID](#) of the [nTDSDSA](#) object for the reference DC.

ulOptions: 0 to expunge each object that is not verified, or 1 to log an event that identifies each such object.

4.1.24.2 Method-Specific Abstract Types and Procedures

4.1.24.2.1 GetRemoteUTD

```
procedure GetRemoteUTD(
    dsa: DSName, nc: DSName): UPTODATE_VECTOR_V1_EXT
```

The GetRemoteUTD procedure uses the [IDL DRSGetReplInfo](#) method to remotely retrieve the [UPTODATE_VECTOR_V1_EXT](#) for the NC with the [DSName](#) nc from the DC whose [nTDSDSA](#) object has the [DSName](#) dsa.

4.1.24.2.2 ObjectExistsAtDC

```
procedure ObjectExistsAtDC(o object, dsa: DSName): boolean
```

The `ObjectExistsAtDC` procedure executes the tasks necessary to verify that the object `o` exists on the DC whose `nTDSDSA` object has the `DSName` `dsa`. If the object exists, the procedure returns true; otherwise, the procedure returns false.

4.1.24.3 Server Behavior of the `IDL_DRSReplicaVerifyObjects` Method

Informative summary of behavior: Let `N` be the NC `pNC^`, and let the reference DC be the DC corresponding to the `nTDSDSA` object `uuidDsaSrc`.

For the purposes of this method, an object *exists* within an NC replica if it is either an object or a tombstone.

Let `S` be the set of objects that exists in `N` at the server running `IDL_DRSReplicaVerifyObjects` at the time `IDL_DRSReplicaVerifyObjects` begins processing. Let the set `S'` be `S` minus the members of `S` that have never existed in `N` at the reference DC when

`IDL_DRSReplicaVerifyObjects` begins processing. The members of `(S - S')` must be objects recently added to `N` on the server, since otherwise they would have replicated to the reference DC. The set `S'` is computable using the `replUpToDateVector` for `N` at the server and at the reference DC.

For each object `o` in `S'` that does not exist in `N` at the reference DC while

`IDL_DRSReplicaVerifyObjects` is processing, either expunge `o` at the server (if `ulOptions = 0`) or log an administrator-visible event at the server (if `ulOptions = 1`).

If an object goes out of existence in `N` at the reference DC during processing of `IDL_DRSReplicaVerifyObjects`, then `IDL_DRSReplicaVerifyObjects` might or might not Expunge/log the object at the server.

```
ULONG IDL_DRSReplicaVerifyObjects(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)]
        DRS_MSG_REPVERIFYOBJ *pmsgVerify)

msgIn: DRS_MSG_REPVERIFYOBJ_V1
nc, refDsa, o: DSName
uTDServer, uTDRef, uTDMerge: UPTODATE_VECTOR_V1_EXT
sPrime: set of DSName

/* Perform input validation and access check */
if dwInVersion ≠ 0x1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgVerify^.V1
if msgIn.pNC = null then
    return ERROR_DS_DRA_BAD_NC
endif
nc := msgIn.pNC^
if not FullReplicaExists(nc) and
    not PartialGCReplicaExists(nc) then
    return ERROR_DS_DRA_BAD_NC
endif
refDsa := select one object o from subtree ConfigNC() where
    o!objectGUID = msgIn.uuidDsaSrc and nTDSDSA in o!objectClass
if refDsa = null then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
```

```

    return ERROR_DS_DRA_ACCESS_DENIED
endif

/* Compute the set S' */
uTDServer := nc!replUpToDateVector
uTDRef := GetRemoteUTD(refDsa, nc)
if uTDRef = null then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
uTDMerge := MergeUTD(uTDServer, uTDRef)

sPrime := select all objects o from subtree-ts-included nc where
    StampLessThanOrEqualUTD(AttrStamp(o, whenCreated), uTDMerge)

/* Process the set S' */
for each o in sPrime
    if ObjectExistsAtDC(o, refDSA) then
        if msgIn.ulOptions = 0 then
            Expunge(o)
        else if msgIn.ulOptions = 1 then
            Log a message: o exists on server but does not exist on refDsa
        endif
    endif
endfor

return 0

```

Windows behavior about the for loop is specified in the following citation: [<24>](#)

4.1.25 IDL_DRSUnbind (Opnum 1)

The **IDL_DRSUnbind** method destroys a context handle previously created by the [IDL_DRSBind](#) method.

```

ULONG IDL_DRSUnbind(
    [in, out, ref] DRS_HANDLE* phDrs
);

```

phDrs: The pointer to the RPC context handle returned by the **IDL_DRSBind** method. The value is set to NULL on return.

Return Values: 0 if successful, or a Windows error code if a failure occurs.

4.1.25.1 Server Behavior of the IDL_DRSUnbind Method

Informative summary of behavior: The server releases any resources associated with the context handle, making the context handle unusable by the client. The server sets phDrs to null.

```

ULONG
IDL_DRSUnbind(
    [in, out, ref] DRS_HANDLE *phDrs)
phDrs^ := null
return 0

```

4.1.26 IDL_DRSUpdateRefs (Opnum 4)

The **IDL_DRSUpdateRefs** method adds or deletes a value from the [repsTo](#) of a specified NC replica.

```
ULONG IDL_DRSUpdateRefs(  
    [in, ref] DRS_HANDLE hDrs,  
    [in] DWORD dwVersion,  
    [in, ref, switch_is(dwVersion)]  
        DRS_MSG_UPDREFS* pmsgUpdRefs  
);
```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwVersion: The version of the request message.

pmsgUpdRefs: The pointer to the request message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.26.1 Method-Specific Concrete Types

4.1.26.1.1 DRS_MSG_UPDREFS

The **DRS_MSG_UPDREFS** union defines the request message versions sent to the [IDL_DRSUpdateRefs](#) method. Only one version, identified by `dwVersion = 1`, is currently defined.

```
typedef  
[switch_type(DWORD)]  
union {  
    [case(1)]  
        DRS_MSG_UPDREFS_V1 V1;  
} DRS_MSG_UPDREFS;
```

V1: The version 1 request.

4.1.26.1.2 DRS_MSG_UPDREFS_V1

The **DRS_MSG_UPDREFS_V1** structure defines a request message sent to the [IDL_DRSUpdateRefs](#) method.

```
typedef struct {  
    [ref] DSNAME* pNC;  
    [ref, string] char* pszDsaDest;  
    UUID uuidDsaObjDest;  
    ULONG ulOptions;  
} DRS_MSG_UPDREFS_V1;
```

pNC: The pointer to the [DSName](#) of the root of an NC replica on the server.

pszDsaDest: The transport-specific [NetworkAddress](#) of a DC.

uuidDsaObjDest: A DSA GUID.

ulOptions: The [DRS_OPTIONS](#) that control the update.

4.1.26.2 Server Behavior of the IDL_DRSUpdateRefs Method

Informative summary of behavior: If ulOptions contains DRS_ADD_REF, the server adds a value to the [repsTo](#) of the specified NC replica; if ulOptions contains DRS_DEL_REF, the server deletes a value. These options may be combined to replace an existing [repsTo](#) value with a new value; if a corresponding value does not already exist, this is the same as if ulOptions contained DRS_ADD_REF but not DRS_DEL_REF. The client includes DRS_WRIT_REP in ulOptions if the specified NC replica is writable. The client specifies at least one of pszDsaDest and uuidDsaObjDest to identify the value to be added or removed. If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously. If the server adds a value to [repsTo](#), the value has ulReplicaFlags equal to ulOptions \cap {DRS_WRIT_REP}.

```
ULONG IDL_DRSUpdateRefs(  
    [in, ref] DRS_HANDLE hDrs,  
    [in] DWORD dwVersion,  
    [in, ref, switch_is(dwVersion)] DRS_MSG_UPDREFS *pmsgUpdRefs);  
msgIn: DRS_MSG_UPDREFS_V1  
options: DRS_OPTIONS  
err: DWORD  
nc: DSName  
rt: RepsTo  
if dwInVersion  $\neq$  1 then  
    return ERROR_INVALID_PARAMETER  
endif  
msgIn := pmsgUpdRefs^.V1  
options := msgIn.ulOptions  
if msgIn.pNC = null or  
    (msgIn.pszDsaDest = null and  
     msgIn.uuidDsaObjDest = null) or  
    (options  $\cap$  {DRS_ADD_REF, DRS_DEL_REF} = null)  
    return ERROR_INVALID_PARAMETER  
endif  
nc := msgIn.pNC^  
if not ObjExists(nc) then  
    return ERROR_DS_DRA_BAD_DN  
endif  
if (IT_WRITE in nc!instanceType)  $\neq$   
    (DRS_WRIT_REP in options) then  
    return ERROR_DS_DRA_BAD_NC  
endif  
if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then  
    return ERROR_DS_DRA_ACCESS_DENIED  
endif  
if DRS_ASYNC_OP in options then  
    Asynchronous Processing: Initiate a logical thread of control  
    to process the remainder of this request asynchronously  
    return 0  
endif  
if DRS_DEL_REF in options then  
    rt := select one v from nc!repsTo where  
        (v.naDsa = msgIn.pszDsaDest or  
         v.uuidDsa = msgIn.uuidDsaObjDest)  
    if rt = null then
```



```

        err := ERROR_DS_DRA_REF_NOT_FOUND
    else
        nc!repsTo := nc!repsTo - {rt}
        err := 0
    endif
endif
endif
/* If DRS_DEL_REF and DRS_ADD_REF are both specified, the return
 * value is that associated with the DRS_ADD_REF. */
if DRS_ADD_REF in options then
    rt := select one v from nc!repsTo where
        (v.naDsa = msgIn.pszDsaDest or
         v.uuidDsa = msgIn.uuidDsaObjDest)
    if rt = null then
        rt.naDsa := msgIn.pszDsaDest
        rt.uuidDsa := msgIn.uuidDsaObjDest
        rt.options := options ∩ {DRS_WRIT_REF}
        rt.timeLastAttempt := 0
        rt.timeLastSuccess := 0
        rt.consecutiveFailures := 0
        rt.resultLastAttempt := 0
        nc!repsTo := nc!repsTo + {rt}
        err := 0
    else
        err := ERROR_DS_DRA_REF_ALREADY_EXISTS
    endif
endif
endif
return err

```

4.1.26.3 Examples of the IDL_DRSUpdateRefs Method

4.1.26.3.1 Initial State

The [repsTo](#) attribute on the NC root object for domain NC CONTOSO.COM on DC1 does not contain a value for DC2:

```
ldap_search_s("DC=CONTOSO,DC=COM", baseObject, "(objectclass=*)", [repsTo])
```

Result <0>: (null)

Matched DNs:

Getting 1 entries:

```
>> Dn: DC=CONTOSO,DC=COM
```

4.1.26.3.2 Client Request

DC2 invokes the [IDL_DRSUpdateRefs](#) method against DC1, with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwVersion = 1
- pmsgUpdRefs = 0x0006fe08 ; Pointer to the following structure:
- pNC: 0x0008cdb8 _DSNAME DC=CONTOSO, DC=COM

- pszDsaDest : "DC2.CONTOSO.COM"
- uuidDsaObjDest: _GUID {0e9240b6-b3dd-4c86-92e3-2757e56ff0b3}
- ulOptions: DRS_WRIT_REP | DRS_ADD_REF

4.1.26.3.3 Server Response

Return code of 0.

4.1.26.3.4 Final State

The [repsTo](#) attribute on the NC root object for domain NC CONTOSO.COM on DC1 contains one value:

```
ldap_search_s("DC=CONTOSO,DC=COM", baseObject, "(objectclass=*)", [repsTo])
```

Result <0>: (null)

Matched DNs:

Getting 1 entries:

```
>> Dn: DC=CONTOSO,DC=COM
```

```
1> repsTo: dwVersion = 1,
```

- V1.cb: 229, V1.cConsecutiveFailures: 0, V1.timeLastSuccess: 0,
- V1.timeLastAttempt: 0, V1.ulResultLastAttempt: 0x0,
- V1.cbOtherDraOffset: 216,
- V1.cbOtherDra: 13, V1.ulReplicaFlags: 0x10, V1.rtSchedule: <ldp:skipped>,
- V1.usnvec.usnHighObjUpdate: 0, V1.usnvec.usnHighPropUpdate: 0,
- V1.uuidDsaObj: 0e9240b6-b3dd-4c86-92e3-2757e56ff0b3
- V1.uuidInvocId: 00000000-0000-0000-0000-000000000000
- V1.uuidTransportObj: 00000000-0000-0000-0000-000000000000
- V1.mtx_address: DC2.CONTOSO.COM
- V1.cbPASDataOffset: 0 V1.PasData: version = -1, size = -1, flag = -1;

4.1.27 IDL_DRSVerifyNames (Opnum 8)

The **IDL_DRSVerifyNames** method resolves a sequence of object identities.

```
ULONG IDL_DRSVerifyNames(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_VERIFYREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
```

```

        DRS_MSG_VERIFYREPLY* pmsgOut
    );

```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, otherwise a Windows error code.

4.1.27.1 Method-Specific Concrete Types

4.1.27.1.1 DRS_MSG_VERIFYREQ

The **DRS_MSG_VERIFYREQ** union defines the request messages sent to the [IDL_DRSVerifyNames](#) method. Only one version, identified by dwVersion = 1, is currently defined.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_VERIFYREQ_V1 V1;
} DRS_MSG_VERIFYREQ;

```

V1: The version 1 request.

4.1.27.1.2 DRS_MSG_VERIFYREQ_V1

The **DRS_MSG_VERIFYREQ_V1** structure defines a request message sent to the [IDL_DRSVerifyNames](#) method.

```

typedef struct {
    DWORD dwFlags;
    [range(1,10000)] DWORD cNames;
    [size_is(cNames)] DSNAME** rpNames;
    ATTRBLOCK RequiredAttrs;
    SCHEMA_PREFIX_TABLE PrefixTable;
} DRS_MSG_VERIFYREQ_V1;

```

dwFlags: The type of name to be verified; MUST have one of the following values:

Value	Meaning
DRS_VERIFY_DSNAME 0x00000000	Verifies DSName values.

Value	Meaning
DRS_VERIFY_SIDS 0x00000001	Verifies objectSid values.
DRS_VERIFY_SAM_ACCOUNT_NAMES 0x00000002	Verifies sAMAccountName values.
DRS_VERIFY_FPOS 0x00000003	Verifies foreign principal object names.

cNames: The count of items in the rpNames array.

rpNames: An array of pointers to DSNames that need to be verified.

RequiredAttrs: The list of attributes to be retrieved for each name that is verified.

PrefixTable: The prefix table with which to translate [ATTRTYP](#) values in RequiredAttrs to [OID](#) values.

4.1.27.1.3 DRS_MSG_VERIFYREPLY

The **DRS_MSG_VERIFYREPLY** union defines the response messages received from the [IDL DRSVerifyNames](#) method. Only one version, identified by dwVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_VERIFYREPLY_V1 V1;
} DRS_MSG_VERIFYREPLY;
```

V1: The version 1 reply.

4.1.27.1.4 DRS_MSG_VERIFYREPLY_V1

The **DRS_MSG_VERIFYREPLY_V1** structure defines a response message received from the [IDL DRSVerifyNames](#) method.

```
typedef struct {
    DWORD error;
    [range(0,10000)] DWORD cNames;
    [size_is(cNames)] ENTINF* rpEntInf;
    SCHEMA_PREFIX_TABLE PrefixTable;
} DRS_MSG_VERIFYREPLY_V1;
```

error: Unused. MUST be 0 and ignored.

cNames: The count of items in the rpEntInf array.

rpEntInf: The array of [ENTINF](#) structures that contain the attributes requested in the **RequestAttrs** field of the input [DRS_MSG_VERIFYREQ_V1](#) structure if the corresponding name is verified.

PrefixTable: The prefix table with which to translate [ATTRTYP](#) values in the response to **OIDs**.

4.1.27.2 Server Behavior of the IDL_DRSVerifyNames Method

Informative summary of behavior: The server resolves each of a sequence of object names and returns its [DSName](#) and the values of zero or more of its attributes. The type of the input object name is indicated by the **dwFlags** field in the request. The [IDL_DRSVerifyNames](#) method verifies the names of both deleted and normal objects.

```

ULONG
IDL_DRSVerifyNames(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_VERIFYREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_VERIFYREPLY *pmsgOut);

msgIn: DRS_MSG_VERIFYREQ_V1
msgOut: DRS_MSG_VERIFYREPLY_V1
nc, d: DSName
o: sequence of DSName
i, j, k: int
domainName, username: unicodestring
done: boolean
attribute: ATTRTYP
FilterPAS: PARTIAL_ATTR_VECTOR_V1_EXT
GCPAS: PARTIAL_ATTR_VECTOR_V1_EXT

/* Perform input validation and access check */
if dwInVersion ≠ 0x1 then
    return ERROR_INVALID_PARAMETER
endif
if not AccessCheckCAR(DefaultNC(), DS-Replication-Get-Changes)
    then
        return ERROR_DS_DRA_ACCESS_DENIED
    endif
msgIn := pmsgIn^.V1
if msgIn.dwFlags ≠ DSR_VERIFY_DSNames and
    msgIn.dwFlags ≠ DRS_VERIFY_SAM_ACCOUNT_NAMES and
    msgIn.dwFlags ≠ DRS_VERIFY_SIDS and
    msgIn.dwFlags ≠ DRS_VERIFY_FPOS then
    return ERROR_INVALID_PARAMETER
endif
if msgIn.cNames > 0 and msgIn.rpNames = null then
    return ERROR_INVALID_PARAMETER
endif
if (msgIn.dwFlags = DRS_VERIFY_SIDS or
    msgIn.dwFlags = DRS_VERIFY_SAM_ACCOUNT_NAMES or
    msgIn.dwFlags = DRS_VERIFY_FPOS) and
    not IsGC() then
    return ERROR_DS_GC_REQUIRED

```

```

endif
if msgIn.dwFlags = DRS_VERIFY_NAMES and not IsGC() then
    for i := 0 to msgIn.cNames-1
        if DefaultNC() ≠ GetObjectNC(msgIn.rpNames[i]^) then
            return ERROR_DS_GC_REQUIRED
        endif
    endfor
endif

/* Compute output */
msgOut.PrefixTable := dc.prefixTable
for i := 0 to msgIn.cNames - 1
    d := msgIn.rpNames[i]
    o := null
    done := false

    if msgIn.dwFlags = DRS_VERIFY_SAM_ACCOUNT_NAMES then
        domainName := DomainNameFromNT4AccountName(d.dn)
        username := UserNameFromNT4AccountName(d.dn)

        if domainName ≠ null and username ≠ null and
            IsDomainNameInTrustedForest(domainName) then
            /* Provide a hint as to which forest this name could be coming
             * from. Note that 0xFFFF0009 is a hardcoded attribute ID
             * recognized by clients of this method. This attribute ID does
             * not correspond to any attribute defined in the schema. */
            msgOut.rpEntInf[i].pName := null
            msgOut.rpEntInf[i].AttrBlock.AttrCount := 1
            msgOut.rpEntInf[i].AttrBlock.pAttr[0].AttrTyp := 0xFFFF0009
            msgOut.rpEntInf[i].AttrBlock.pAttr[0].AttrVal.valCount := 1
            msgOut.rpEntInf[i].AttrBlock.pAttr[0].AttrVal.pAVal[0].valLen
                := Length in characters of domainName, excluding any
                terminating null
            msgOut.rpEntInf[i].AttrBlock.pAttr[0].AttrVal.pAVal[0].pAVal :=
                domainName
            done := true
        endif
    endif

    if not done
        /* locate object or objects in question */
        if msgIn.dwFlags = DRS_VERIFY_NAMES then
            if ObjExists(d) then
                o := {d}
            endif
        else if msgIn.dwFlags = DRS_VERIFY_SIDS then
            o := select all v from all-ts-included
                where v!objectSid = d.sid and
                foreignSecurityPrincipal not in v!objectClass
        else if msgIn.dwFlags = DRS_VERIFY_SAM_ACCOUNT_NAMES then
            if domainName ≠ NULL and username ≠ NULL then
                nc := select one v from all
                    where v!nETBIOSName = domainName and GetObjectNC(v) = v
                /* The following query returns both normal objects
                 * and tombstones */
                o := select all v from subtree-ts-included nc where
                    v!sAMAccountName = username
            else

```

```

        o := select all v from all-ts-included
            where v!userPrincipalName =
                d.dn
    endif
else if msgIn.dwFlags = DRS_VERIFY_FPOS then
    o := select all v from all-ts-included
        where v!objectSid = d.sid
        and foreignSecurityPrincipal in v!objectClass
endif

/* Compute returned info and get requested attributes */
if o.length = 1 then
    msgOut.rpEntInf[i].pName = o[0]!distinguishedName
    if MasterReplicaExists(GetObjectNC(o[0])) then
        msgOut.rpEntInf[i].ulFlags := ENTINF_FROM_MASTER
    else
        msgOut.rpEntInf[i].ulFlags := 0
    endif
    msgOut.rpEntInf[i].AttrBlock.AttrCount :=
        msgIn.RequiredAttrs.AttrCount

    FilterPas := FilteredPAS()
    GCPas := GCPas()

    for j := 0 to msgIn.RequiredAttrs.AttrCount - 1

        if AmILHServer() then
            if (not (msgIn.RequiredAttrs.pAttr[j].AttrType in FilterPas
                &&
                msgIn.RequiredAttrs.pAttr[j].AttrType in GCPas))
            then
                /* skip requested attributes not part of both FilterPAS
                and GCPas */
                msgOut.rpEntInf[i] := null
                continue;
            endif
        else
            /* pre-LH server */
            if (not (msgIn.RequiredAttrs.pAttr[j].AttrType in GCPas))
            then
                /* skip requested attributes not part of GCPas */
                msgOut.rpEntInf[i] := null
                continue;
            endif
        endif

        attribute := LocalAttidFromRemoteAttid(
            msgIn.PrefixTable,
            msgIn.RequiredAttrs.pAttr[j].attrType)
        msgOut.rpEntInf[i].AttrBlock.pAttr[j].attrType := attribute
        k := 0
        foreach val in GetAttrVals(o, attribute, false)
            msgOut.rpEntInf[i].AttrBlock.pAttr[j].AttrVal.pAVal :=
                ADR(ATRVALFromValue(val,
                    Syntax(attribute),
                    dc.prefixTable))
            msgOut.rpEntInf[i].AttrBlock.pAttr[j].AttrVal.valCount :=
                k + 1
        end
    end
end

```

```

        endfor
    endfor
    else
        msgOut.rpEntInf[i] := null
    endif
    endif
endfor /* i := */

pmsgOut^.V1 := msgOut
return 0

```

4.1.27.3 Examples of the IDL_DRSVerifyNames Method

4.1.27.3.1 Initial State

Querying the [user](#) object JaneDow on DC=CONTOSO, DC=COM

- `ldap_search_s("CN=JaneDow,CN=Users,DC=contoso,DC=com", baseObject, "(objectClass=*)", [objectGUID, objectSid, sAMAccountName, sAMAccountType])`
- Getting 1 entries:
- `>> Dn: CN=JaneDow,CN=Users,DC=contoso,DC=com`
 - `1> objectGUID: 772cf177-00f8-45ed-9c72-5e5206bead02;`
 - `1> objectSid: S-1-5-21-3263199975-614030967-162443871-1603;`
 - `1> sAMAccountName: JaneDow;`
 - `1> sAMAccountType: SAM_NORMAL_USER_ACCOUNT;`

4.1.27.3.2 Client Request

To get a user's SID, DC2 invokes the [IDL_DRSVerifyNames](#) method against DC1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- `dwInVersion = 1`
- `pmsgIn = DRS_MSG_VERIFYREQ_V1`
 - `dwFlags: 2`
 - `cNames: 1`
 - `rpNames: DSNAME`
 - `StringName: "CN=Jane Dow,CN=Users,DC=contoso,DC=com"`
 - `RequiredAttrs: ATTRBLOCK`
 - `attrCount: 3`
 - `pAttr: ATTR`
 - `sAMAccountType`

- objectSid
- sAMAccountName

4.1.27.3.3 Server Response

The server responds with a return code of 0 with the following values:

- pMsgOut = DRS_MSG_VERIFYREPLY_V1
 - cNames: 1
 - rpEntInf: ENTINF
 - pName: DSNAME
 - Guid: GUID {772cf177-00f8-45ed-9c72-5e5206bead02}
 - SID: S-1-5-21-3263199975-614030967-162443871-1603
 - String Name: "CN=Jane Dow,CN=Users,DC=contoso,DC=com"
 - ulFlags: ENTINF_FROM_MASTER
 - AttrBlock: ATTRBLOCK
 - sAMAccountType: 0x30000000
 - objectSid: S-1-5-21-3263199975-614030967-162443871-1603
 - sAMAccountName: JaneDow
 - PrefixTable: SCHEMA_PREFIX_TABLE

4.1.27.3.4 Final State

No change in state.

4.1.28 IDL_DRSWriteSPN (Opnum 13)

The **IDL_DRSWriteSPN** method updates the set of SPNs on an object.

```
ULONG IDL_DRSWriteSPN(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_SPNREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_SPNREPLY* pmsgOut
);
```

hDrs: The RPC context handle returned by the [IDL_DRSBind](#) method.

dwInVersion: The version of the request message. MUST be set to 1, because that is the only version supported.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message. The value is always 1, because that is the only version supported.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, or a Windows error code if a failure occurs.

4.1.28.1 Method-Specific Concrete Types

4.1.28.1.1 DRS_MSG_SPNREQ

The **DRS_MSG_SPNREQ** union defines the request messages sent to the [IDL DRSWriteSPN](#) method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_SPNREQ_V1 V1;
} DRS_MSG_SPNREQ;
```

V1: The version 1 request.

4.1.28.1.2 DRS_MSG_SPNREQ_V1

The **DRS_MSG_SPNREQ_V1** structure defines a request message sent to the [IDL DRSWriteSPN](#) method.

```
typedef struct {
    DWORD operation;
    DWORD flags;
    [string] const WCHAR* pwszAccount;
    [range(0,10000)] DWORD cSPN;
    [string, size_is(cSPN)] const WCHAR** rpwszSPN;
} DRS_MSG_SPNREQ_V1;
```

operation: The SPN operation to perform. MUST be one of the [DS_SPN_OPERATION](#) values.

flags: MUST be 0.

pwszAccount: The DN of the object to modify.

cSPN: The count of items in the rpwszSPN array.

rpwszSPN: The SPN values.

4.1.28.1.3 DRS_MSG_SPNREPLY

The **DRS_MSG_SPNREPLY** union defines the response messages received from the [IDL DRSWriteSPN](#) method. Only one version, identified by `pdwOutVersion^ = 1`, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DRS_MSG_SPNREPLY_V1 V1;
} DRS_MSG_SPNREPLY;
```

V1: The version 1 response.

4.1.28.1.4 DRS_MSG_SPNREPLY_V1

The **DRS_MSG_SPNREPLY_V1** structure defines a response message received from the [IDL DRSWriteSPN](#) method.

```
typedef struct {
    DWORD retVal;
} DRS_MSG_SPNREPLY_V1;
```

retVal: 0, or a Windows error code.

4.1.28.1.5 DS_SPN_OPERATION

The **DS_SPN_OPERATION** type indicates the operation to perform.

This type is declared as follows:

```
typedef DWORD DS_SPN_OPERATION;
```

It must be one of the following values:

Value	Meaning
DS_SPN_ADD_SPN_OP (0x00000000)	Adds the specified values to the existing set of SPNs.
DS_SPN_REPLACE_SPN_OP (0x00000001)	Removes all the existing SPNs, then adds the specified values. If the set of specified values is empty (cSPN is zero), no values are added.
DS_SPN_DELETE_SPN_OP (0x00000002)	Removes all the existing SPNs.

4.1.28.2 Server Behavior of the IDL_DRSWriteSPN Method

Informative summary of behavior: The [IDL_DRSWriteSPN](#) method updates the [servicePrincipalName](#) attribute of an object. The values of this multivalued attribute are called service principal names (SPNs). The **IDL_DRSWriteSPN** method does one of three things:

- Adds a non-empty set of SPNs to the object's [servicePrincipalName](#). If a member of the set is already present on the object's [servicePrincipalName](#), it is ignored.
- Removes all current values from the object's [servicePrincipalName](#), then adds a (possibly empty) set of SPNs to the object's [servicePrincipalName](#).
- Removes a non-empty set of SPNs from the object's [servicePrincipalName](#). If a member of the set is not present on the object's [servicePrincipalName](#), it is ignored.

The effect of this method can be achieved by an LDAP Modify operation to the [servicePrincipalName](#) attribute of an object. Some manipulations of the [servicePrincipalName](#) attribute that cannot be performed using this method can be performed using LDAP Modify. For example, an LDAP Modify can remove one specific SPN from the [servicePrincipalName](#) attribute while adding another SPN to the [servicePrincipalName](#) attribute in the same transaction; IDL_DRSWriteSPN cannot do this.

```
ULONG
IDL_DRSWriteSPN(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_SPNREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_SPNREPLY *pmsgOut);
accountDN: unicodestring
account: DSName
operation: DS_SPN_OPERATION
cSPN: integer
spnSet: set of unicodestring
instanceName: unicodestring
pdwOutVersion^ := 1
pmsgOut^.V1.retVal := 0
/* Input parameter validation */
if dwInVersion != 1 then
    pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
    return ERROR_INVALID_PARAMETER
endif
/* RODCs do not perform originating updates */
if AmIRODC() then
    pmsgOut^.V1.retVal := ERROR_DS_REFERRAL
    return ERROR_DS_REFERRAL
endif
accountDN := pmsgIn^.V1.pwszAccount
operation := pmsgIn^.V1.operation
cSPN := pmsgIn^.V1.cSPN
spnSet := pmsgIn^.V1.rpwszSPN
if accountDN = null or accountDN = "" then
    pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
    return ERROR_INVALID_PARAMETER
endif
if not operation in [DS_SPN_ADD_SPN_OP .. DS_SPN_DELETE_SPN_OP] then
```

```

    pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
    return ERROR_INVALID_PARAMETER
endif
/* DS_SPN_REPLACE_SPN_OP permits 0 SPNs to be specified (meaning
 * "delete all SPNs"). Other operations require >=1 SPNs to be
 * specified. */
if (operation ≠ DS_SPN_REPLACE_SPN_OP) and (cSPN = 0) then
    pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
    return ERROR_INVALID_PARAMETER
endif
/* The empty string is an invalid SPN. */
foreach spn in spnSet
    if spn = null or spn = "" then
        pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
        return ERROR_INVALID_PARAMETER
    endif
endfor
account := GetDSNameFromDN(accountDN);
if not ObjExists(account) then
    pmsgOut^.V1.retVal := ERROR_DS_NO_SUCH_OBJECT
    return ERROR_DS_NO_SUCH_OBJECT
endif
/* Perform access checks */
if not AccessCheckWriteToSpnAttribute(account, spnSet) then
    pmsgOut^.V1.retVal := ERROR_ACCESS_DENIED
    return ERROR_ACCESS_DENIED
endif
if (operation = DS_SPN_DELETE_SPN_OP) then
    /* Remove specified SPNs */
    foreach spn in spnSet
        if spn in account!servicePrincipalName then
            account!servicePrincipalName :=
                account!servicePrincipalName - {spn}
        endif
    endfor
    return 0
endif
if (operation = DS_SPN_ADD_SPN_OP) then
    /* Add specified SPNs */
    foreach spn in spnSet
        account!servicePrincipalName :=
            account!servicePrincipalName + {spn}
    endfor
    return 0
endif
/* Must be DS_SPN_REPLACE_SPN_OP.
 * Remove all existing SPNs, then add in the specified SPNs. */
account!servicePrincipalName := {null}
foreach spn in spnSet
    account!servicePrincipalName :=
        account!servicePrincipalName + {spn}
endfor
return 0

```

4.2 dsaop RPC Interface

This section specifies the methods for the **dsaop** RPC interface of the DRS Remote Protocol, in addition to the processing rules.

Methods in RPC Opnum Order

Method	Description
IDL_DSAPrepareScript	Prepares the DC to run a maintenance script. Opnum: 0
IDL_DSASExecuteScript	Executes a maintenance script. Opnum: 1

For information on the order of method calls, see section [1.3.2](#).

All methods MUST NOT throw exceptions.

4.2.1 IDL_DSAPrepareScript (Opnum 0)

The **IDL_DSAPrepareScript** method prepares the DC to run a maintenance script.

```
ULONG IDL_DSAPrepareScript(  
    [in] handle_t hRpc,  
    [in] DWORD dwInVersion,  
    [in, ref, switch_is(dwInVersion)]  
        DSA_MSG_PREPARE_SCRIPT_REQ* pmsgIn,  
    [out, ref] DWORD* pdwOutVersion,  
    [out, ref, switch_is(*pdwOutVersion)]  
        DSA_MSG_PREPARE_SCRIPT_REPLY* pmsgOut  
);
```

hRpc: The RPC binding handle, as specified in [\[C706\]](#).

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, or a Windows error code if a failure occurs.

4.2.1.1 Method-Specific Concrete Types

4.2.1.1.1 DSA_MSG_PREPARE_SCRIPT_REQ

The **DSA_MSG_PREPARE_SCRIPT_REQ** union defines the request messages sent to the [IDL_DSAPrepareScript](#) method.

```
typedef  
[switch_type(DWORD)]
```

```

union {
    [case(1)]
        DSA_MSG_PREPARE_SCRIPT_REQ_V1 V1;
} DSA_MSG_PREPARE_SCRIPT_REQ;

```

V1: The version 1 request.

4.2.1.1.2 DSA_MSG_PREPARE_SCRIPT_REQ_V1

The **DSA_MSG_PREPARE_SCRIPT_REQ_V1** structure defines a request message sent to the [IDL DSAPrepareScript](#) method.

```

typedef struct {
    DWORD Reserved;
} DSA_MSG_PREPARE_SCRIPT_REQ_V1;

```

Reserved: MUST be 0.

4.2.1.1.3 DSA_MSG_PREPARE_SCRIPT_REPLY

The **DSA_MSG_PREPARE_SCRIPT_REPLY** union defines the response messages received from the [IDL DSAPrepareScript](#) method.

```

typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DSA_MSG_PREPARE_SCRIPT_REPLY_V1 V1;
} DSA_MSG_PREPARE_SCRIPT_REPLY;

```

V1: The version 1 response.

4.2.1.1.4 DSA_MSG_PREPARE_SCRIPT_REPLY_V1

The **DSA_MSG_PREPARE_SCRIPT_REPLY_V1** structure defines a response message received from the [IDL DSAPrepareScript](#) method.

```

typedef struct {
    DWORD dwOperationStatus;
    [string] LPWSTR pwErrorMessage;
    [range(0,1024)] DWORD cbPassword;
    [size_is(cbPassword)] BYTE* pbPassword;
    [range(0,10485760)] DWORD cbHashBody;
    [size_is(cbHashBody)] BYTE* pbHashBody;
    [range(0,10485760)] DWORD cbHashSignature;
    [size_is(cbHashSignature)] BYTE* pbHashSignature;
} DSA_MSG_PREPARE_SCRIPT_REPLY_V1;

```

dwOperationStatus: 0 if successful, or a Windows error code if a fatal error occurred.

pwErrorMessage: NULL if successful, or a description of an error if a fatal error occurred.

cbPassword: The count, in bytes, of the pbPassword array.

pbPassword: The password.

cbHashBody: The count, in bytes, of the pbHashBody array.

pbHashBody: The hash of the script value.

cbHashSignature: The count, in bytes, of the pbHashSignature array.

pbHashSignature: The script signature.

4.2.1.2 Method-Specific Abstract Types and Procedures

4.2.1.2.1 GetKeyLengthHandleT

```
procedure GetKeyLengthHandleT(hRpc: handle_t): integer
```

Returns the key length, in bits, of the encryption used on the hRpc connection. Returns 0 if no encryption is in use on the connection.

4.2.1.2.2 PrepareScriptInProgress

```
procedure PrepareScriptInProgress(): boolean
```

Returns true if an instance of the [IDL_DSAPrepareScript\(\)](#) method is already executing, and false otherwise.

4.2.1.2.3 PrepareScriptVerifyScript

```
procedure PrepareScriptVerifyScript(pc: DSName): boolean
```

Returns true if the value of pc![msDS-UpdateScript](#) is valid where the validation criterion is implementation-specific, and false otherwise.

4.2.1.2.4 PrepareScriptHashBody

```
procedure PrepareScriptHashBody(pc: DSName): sequence of BYTE
```

Returns a SHA1 hash of the value of pc![msDS-UpdateScript](#).

4.2.1.2.5 PrepareScriptHashSignature

```
procedure PrepareScriptHashSignature(pc: DSName): sequence of BYTE
```


Returns a SHA1 hash of the value formed by appending the GUID {0916C8E3-3431-4586-AF77-44BD3B16F961} to the value of pc![msDS-UpdateScript](#).

4.2.1.2.6 PrepareScriptGeneratePassword

procedure PrepareScriptGeneratePassword(): sequence of BYTE

Returns a randomly generated password for use in a subsequent call to [IDL_DSAExecuteScript](#).

4.2.1.3 Server Behavior of the IDL_DSAPrepareScript Method

Informative summary of behavior: The [IDL_DSAPrepareScript](#) method prepares for a subsequent call to [IDL_DSAExecuteScript\(\)](#). The partitions container that is a child of the root of the configuration NC is altered as follows:

- The value of [msDS-UpdateScript](#) is validated.
- If valid, a password is generated and stored in the value for [msDS-ExecuteScriptPassword](#).

The password, a hash of the value stored in [msDS-UpdateScript](#), and a hash of that same value with the GUID {0916C8E3-3431-4586-AF77-44BD3B16F961} appended are returned to the client. The returned password value is later passed back by the client in a call to **IDL_DSAExecuteScript** as a form of authorization.

```
ULONG
IDL_DSAPrepareScript(
    [in] handle_t hRpc,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DSA_MSG_PREPARE_SCRIPT_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DSA_MSG_PREPARE_SCRIPT_REPLY *pmsgOut);
pc: DSName
msgIn: DSA_MSG_PREPARE_SCRIPT_REQ_V1
byteseq: sequence of BYTE
/* Validate the version */
if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
/* Validate input params */
if msgIn.Reserved ≠ 0 then
    return ERROR_INVALID_PARAMETER
endif
/* Returned message will be version 1 */
pdwOutVersion^ := 1
/* Only 1 instance of this call can be running. */
if PrepareScriptInProgress() then
    pmsgOut^.V1.dwOperationStatus := ERROR_ACCESS_DENIED
    pmsgOut^.V1.pwErrMessage := human-readable description of the error
    return 0
endif
/* Locate the Partitions container directly beneath ConfigNC */
pc := DescendantObject(ConfigNC(), "CN=Partitions,")
/* Forest functionality level must be Win2K3 or above */
```

```

if pc!msDS-Behavior-Version = null or
    pc!msDS-Behavior-Version < DS_BEHAVIOR_WIN2003 then
    return ERROR_DS_NOT_SUPPORTED
endif
/* Security checks */
if not AccessCheckAttr(
    pc, msDS-UpdateScript, RIGHT_DS_WRITE_PROPERTY) then
    pmsgOut^.V1.dwOperationStatus := ERROR_DS_AUTHORIZATION_FAILED
    pmsgOut^.V1.pwErrMsg := human-readable description of the error
    return 0
endif

if not AccessCheckCAR(pc, DS-Execute-Intentions-Script) then
    pmsgOut^.V1.dwOperationStatus := ERROR_DS_AUTHORIZATION_FAILED
    pmsgOut^.V1.pwErrMsg := human-readable description of the error
    return 0
endif
if GetKeyLengthHandleT(hRpc) < 128 then
    pmsgOut^.V1.dwOperationStatus := ERROR_DS_STRONG_AUTH_REQUIRED
    pmsgOut^.V1.pwErrMsg := human-readable description of the error
    return 0
endif
/* Validate stored script */
if not PrepareScriptVerifyScript(pc) then
    pmsgOut^.V1.dwOperationStatus := ERROR_DS_INVALID_SCRIPT
    pmsgOut^.V1.pwErrMsg := human-readable description of the error
    return 0
endif
/* Generate and return password for subsequent call to
 * IDL_DSAScriptExecuteScript() */
pc!msDS-ExecuteScriptPassword := PrepareScriptGeneratePassword()
/* Return password and hashes */
byteseq := pc!msDS-ExecuteScriptPassword
pmsgOut^.V1.pbPassword := byteseq
pmsgOut^.V1.cbPassword := byteseq.length
byteseq := PrepareScriptHashBody(pc)
pmsgOut^.V1.pbHashBody := byteseq
pmsgOut^.V1.cbHashBody := byteseq.length
byteseq := PrepareScriptHashSignature(pc)
pmsgOut^.V1.pbHashSignature := byteseq
pmsgOut^.V1.cbHashSignature := byteseq.length
pmsgOut^.V1.dwOperationStatus := 0
return 0

```

4.2.2 IDL_DSAScriptExecuteScript (Opnum 1)

The **IDL_DSAScriptExecuteScript** method executes a maintenance script.

```

ULONG IDL_DSAScriptExecuteScript(
    [in] handle_t hRpc,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DSA_MSG_EXECUTE_SCRIPT_REQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DSA_MSG_EXECUTE_SCRIPT_REPLY* pmsgOut
)

```

);

hRpc: The RPC binding handle (as specified in [\[C7061\]](#)).

dwInVersion: The version of the request message.

pmsgIn: The pointer to the request message.

pdwOutVersion: The pointer to the version of the response message.

pmsgOut: The pointer to the response message.

Return Values: 0 if successful, or a Windows error code if a failure occurs.

4.2.2.1 Method-Specific Concrete Types

4.2.2.1.1 DSA_MSG_EXECUTE_SCRIPT_REQ

The **DSA_MSG_EXECUTE_SCRIPT_REQ** union defines the request messages sent to the [IDL DSAExecuteScript](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DSA_MSG_EXECUTE_SCRIPT_REQ_V1 V1;
} DSA_MSG_EXECUTE_SCRIPT_REQ;
```

V1: The version 1 request.

4.2.2.1.2 DSA_MSG_EXECUTE_SCRIPT_REQ_V1

The **DSA_MSG_EXECUTE_SCRIPT_REQ_V1** structure defines a request message sent to the [IDL DSAExecuteScript](#) method.

```
typedef struct {
    DWORD Flags;
    [range(1,1024)] DWORD cbPassword;
    [size_is(cbPassword)] BYTE* pbPassword;
} DSA_MSG_EXECUTE_SCRIPT_REQ_V1;
```

Flags: MUST be 0.

cbPassword: The count, in bytes, of the pbPassword array.

pbPassword: The password.

4.2.2.1.3 DSA_MSG_EXECUTE_SCRIPT_REPLY

The **DSA_MSG_EXECUTE_SCRIPT_REPLY** union defines the response messages received from the [IDL_DSAExecuteScript](#) method.

```
typedef
[switch_type(DWORD)]
union {
    [case(1)]
        DSA_MSG_EXECUTE_SCRIPT_REPLY_V1 V1;
} DSA_MSG_EXECUTE_SCRIPT_REPLY;
```

V1: The version 1 request.

4.2.2.1.4 DSA_MSG_EXECUTE_SCRIPT_REPLY_V1

The **DSA_MSG_EXECUTE_SCRIPT_REPLY_V1** structure defines a response message received from the [IDL_DSAExecuteScript](#) method.

```
typedef struct {
    DWORD dwOperationStatus;
    [string] LPWSTR pwErrorMessage;
} DSA_MSG_EXECUTE_SCRIPT_REPLY_V1;
```

dwOperationStatus: 0 if successful, or a Windows error code if a fatal error occurred.

pwErrorMessage: NULL if successful, or a description of the error if a fatal error occurred.

4.2.2.2 Method-Specific Abstract Types and Procedures

4.2.2.2.1 ExecuteScriptInProgress

```
procedure ExecuteScriptInProgress(): boolean
```

Returns true if an instance of the [IDL_DSAExecuteScript](#) method is already executing, and false otherwise.

4.2.2.2.2 ExecuteScript

```
procedure ExecuteScript(pc: DSName): ULONG
```

The ExecuteScript procedure executes the value of pc![msDS-UpdateScript](#) as a script; returns 0 if successful, or a Windows error code if a failure occurs.

4.2.2.3 Server Behavior of the IDL_DSAExecuteScript Method

Informative summary of behavior: The value of the attribute [msDS-UpdateScript](#) is executed as a transacted sequence of updates. The RPC call is not authenticated using normal means (that is, it can be performed by an anonymous caller). However, the password value passed by the caller must

match the password that was obtained by a prior call to the [IDL_DSAPrepareScript](#) method on the same DC.

```
ULONG
IDL_DSAScriptExecuteScript(
    [in] handle_t hRpc,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DSA_MSG_EXECUTE_SCRIPT_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DSA_MSG_EXECUTE_SCRIPT_REPLY *pmsgOut);
pc: DSNamespace
msgIn: DSA_MSG_EXECUTE_SCRIPT_REQ_V1
/* Validate the version */
if dwInVersion != 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
/* returned message is version 1 */
pdwOutVersion^ := 1
/* Only 1 instance of this call can be running. */
if ExecuteScriptInProgress() then
    pmsgOut^.V1.dwOperationStatus := ERROR_ACCESS_DENIED
    pmsgOut^.V1.pwErrorMessage := human-readable description of the error
    return 0
endif
pc := DescendantObject(ConfigNC(), "CN=Partitions,")
/* Forest functionality level must be Win2K3 or above */
if pc!msDS-Behavior-Version = null or
   pc!msDS-Behavior-Version < DS_BEHAVIOR_WIN2003 then
    return ERROR_DS_NOT_SUPPORTED
endif
/* Passwords match? */
if pc!msDS-ExecuteScriptPassword != msgIn.pbPassword then
    pmsgOut^.V1.dwOperationStatus := ERROR_DS_AUTHORIZATION_FAILED
    pmsgOut^.V1.pwErrorMessage := human-readable description of the error
    return 0
endif
/* Execute and delete the script. */
pmsgOut^.V1.dwOperationStatus := ExecuteScript(pc)
if pmsgOut^.V1.dwOperationStatus = 0 then
    /* Script executed successfully. Remove the script value */
    pc!msDS-UpdateScript := null
else
    pmsgOut^.V1.pwErrorMessage := human-readable description of the error
endif
return 0
```

5 Common Data Types, Variables, and Procedures

This section contains types that are used by two or more drsuapi or dsaop methods, or types that are used in this specification but that are normatively specified in other specifications. It also contains types and procedures used only within the specification. This section is arranged in order by type or procedure name.

The specification of message syntax in this section is normative for syntax only. The behavior descriptions for types representing messages are informative. Consult the behavior description for each method that uses a type for the normative specification of behavior related to that type.

"Hand-marshaled" types are types passed as BLOBs through RPC and types stored as BLOBs in the directory. Any type that is "hand-marshaled" is specified pictorially in this section to emphasize the layout of any multibyte quantities it contains. The layout is always little-endian. If a type is both "hand-marshaled" and marshaled by RPC, then an **IDL** specification of the type is given, in addition to the pictorial specification.

This specification uses the definitions of RPC base types. Additional data types used in this protocol are specified in this section.

Note that values of some types are marshaled by RPC as structures in some cases and as little-endian byte arrays in other cases. An example is [DSName](#), which may be marshaled as a [DSName](#) *pObject field of an [ENTINF](#), or as a [UCHAR](#) *pVal field of an [ATTRVAL](#). Where such cases exist, the structure is defined both in **MIDL** syntax and in a byte diagram, and the byte array cases are clearly identified so that big-endian architectures can perform the necessary byte swapping. (For example, see [ATTRVAL](#) conversions.)

5.1 AbstractPTFromConcretePT

```
procedure AbstractPTFromConcretePT(  
    concretePrefixTable: SCHEMA_PREFIX_TABLE): PrefixTable
```

Informative summary of behavior: The AbstractPTFromConcretePT procedure translates the [SCHEMA_PREFIX_TABLE](#) structure to the abstract [PrefixTable](#).

```
prefixTable: PrefixTable  
schemaSignature: sequence of BYTE  
i: DWORD  
  
for i := 0 to (concretePrefixTable.PrefixCount - 1)  
    prefixTable[i].prefixString :=  
        concretePrefixTable.pPrefixTableEntry[i].prefix  
    prefixTable[i].prefixIndex :=  
        concretePrefixTable.pPrefixTableEntry[i].ndx  
endfor  
  
return concretePrefixTable
```

5.2 AccessCheckAttr

```
procedure AccessCheckAttr(  
    dsName: DSName, attr: ATTRTYP, right: Right): boolean
```

The AccessCheckAttr procedure returns true if dsName identifies an object within an NC replica hosted by the server, and the client's security context has the access indicated by the access right **right** to the attribute **attr** on that object; returns false otherwise.

See [\[MS-ADTS\]](#) section 5.1.3 for the specification of this procedure.

See [\[MS-ADTS\]](#) section 5.1.3.2 for the list of symbolic names for access rights (for example, RIGHT_DS_WRITE_PROPERTY) and the numeric value of each.

5.3 AccessCheckCAR

```
procedure AccessCheckCAR(dsName: DSName; right: Right): boolean
```

The AccessCheckCAR procedure returns true if dsName identifies an object within an NC replica hosted by the server, and the client's security context has the access indicated by the control access right **right** on that object; returns false otherwise.

See [\[MS-ADTS\]](#) section 5.1.3 for the specification of this procedure.

See [\[MS-ADTS\]](#) section 5.1.3.2.1 for the list of symbolic names for control access rights (for example, DS-Replication-Manage-Topology) and the numeric value of each.

5.4 AccessCheckObject

```
procedure AccessCheckObject(dsName: DSName, right: Right): boolean
```

The AccessCheckObject procedure returns true if dsName identifies an object within an NC replica hosted by the server, and the client's security context has the access indicated by the access right **right** on that object; returns false otherwise.

See [\[MS-ADTS\]](#) section 5.1.3 for the specification of this procedure.

See [\[MS-ADTS\]](#) section 5.1.3.2 for the list of symbolic names for access rights (for example, RIGHT_DS_DELETE_CHILD) and the numeric value of each.

5.5 AccessCheckWriteToSpnAttribute

```
procedure AccessCheckWriteToSpnAttribute(  
  obj: DSName, spnSet: set of unicodestring) : boolean
```

The AccessCheckWriteToSpnAttribute procedure performs an access check to determine if the client security context has the right to modify the [servicePrincipalName](#) attribute of object obj with the SPN values specified in spnSet, taking into consideration both regular and extended write property rights.

```
if AccessCheckAttr(obj,  
  servicePrincipalName,  
  RIGHT_DS_WRITE_PROPERTY) then  
  return true  
else  
  if AccessCheckAttr(obj,  
    servicePrincipalName,
```

```

        RIGHT_DS_WRITE_PROPERTY_EXTENDED) then
/* Extended write access permits the attribute to be written */
/* provided the proposed SPNs meet certain constraints. */

foreach spn in spnSet
    if not Is2PartSPN(spn) then
        if (Is3PartSPN(spn) and IsDCAccount(obj)) then

            /* Three part SPNs are permitted for DC computer accounts */
            /* However, in addition to the constraints on 2 part SPNs, */
            /* the service name must meet additional constraints */

            serviceName := GetServiceNameFromSPN(spn)
            if not IsValidServiceName(obj, serviceName)
                return false
            endif
        else
            return false
        endif
    endif

    instanceName := GetInstanceNameFromSPN(spn)
    if (instanceName != obj!dNSHostName) and
        (not instanceName + "$" = obj!sAMAccountName) and
        (not instanceName in obj!msDS-AdditionalDnsHostName) and
        (not instanceName + "$" in
            obj!msDS-AdditionalSamAccountName) then
/* If this is a DC computer account */
/* the instance name may be a GUID based dns host name */
if IsDCAccount(obj) then
    if not IsGUIDBasedDNSName(obj, instanceName) then
        return false
    endif
else
    return false
endif
endif
endif
return true
endif

return false
endif

```

5.6 AmIRODC

```

procedure AmIRODC() : Boolean

```

The AmIRODC procedure returns true if the DC is an RODC.

```

return DSAObj()!objectCategory = SchemaObj(nTDSDSARO)

```


5.7 AmILHServer

```
procedure AmILHServer() : boolean
```

The AmILHServer procedure returns true if the DC is Windows Server 2008 or later.

```
/* DS_BEHAVIOR_LH defined in [MS-ADTS] section 7.1.4.2 */  
return DSAObj() !msDS-Behavior-Version ≥ DS_BEHAVIOR_LH
```

5.8 ATTR

The **ATTR** structure defines a concrete type for the identity and values of an attribute.

```
typedef struct {  
    ATTRTYP attrTyp;  
    ATTRVALBLOCK AttrVal;  
} ATTR;
```

attrTyp: An attribute.

AttrVal: The sequence of values for this attribute.

5.9 ATTRBLOCK

The **ATTRBLOCK** structure defines a concrete type for a set of attributes and their values.

```
typedef struct {  
    [range(0,1048576)] ULONG attrCount;  
    [size_is(attrCount)] ATTR* pAttr;  
} ATTRBLOCK;
```

attrCount: The count of items in the pAttr array.

pAttr: The array of attributes and their values.

5.10 AttributeStamp

AttributeStamp is an abstract type that contains information about the last originating update to an attribute. It is a tuple of the following:

- **dwVersion:** A 32-bit integer. Set to 1 when a value for the attribute is set for the first time, incremented by 1 on each subsequent originating update.
- **timeChanged:** The date and time at which the last originating update was made.
- **uuidOriginating:** The invocation ID of the DC that performed the last originating update.
- **usnOriginating:** The USN assigned to the last originating update by the DC that performed it.

Comparisons

Values of `AttributeStamp` are partially ordered. Let d be the result of $x.dwVersion - y.dwVersion$, cast as a 32-bit integer. For example, if $x.dwVersion$ is 1 and $y.dwVersion$ is 3, d is -2. If $x.dwVersion$ is 5 and $y.dwVersion$ is 0xFFFFFFFF, d is 11. Then given two stamps x and y , x is said to be greater than y if any of the following is true:

- x is not null and y is null
- $d > 0$
- $d = 0$ and $x.timeChanged > y.timeChanged$
- $d = 0$ and $x.timeChanged = y.timeChanged$ and $x.uuidOriginating > y.uuidOriginating$

Conversions

A value x of type `AttributeStamp` may be converted to and from its wire format y of type [PROPERTY META DATA EXT](#) by associating the values of fields in x with the values of the like-named fields in y .

5.11 AttributeSyntax

`AttributeSyntax` is an abstract type that represents an LDAP attribute syntax. The valid values are the names from the LDAP Syntax Name column of the table in section [5.15.2](#), for example, `Object(DS-DN)` and `Object(DN-Binary)`.

5.12 AttrStamp

```
procedure AttrStamp(o: DSName, attr: ATTRTYP) : AttributeStamp
```

The `AttrStamp` procedure returns the [AttributeStamp](#) for the attribute whose [ATTRTYP](#) is `attr` on the object whose [DSName](#) is `o`.

5.13 ATTRTYP

ATTRTYP is a concrete type for a compact representation of an OID.

This type is declared as follows:

```
typedef ULONG ATTRTYP;
```

Section [5.15.4](#) specifies the procedures that map between **ATTRTYP** and [OID](#) with the aid of a [SCHEMA PREFIX TABLE](#).

5.14 AttrtypFromSchemaObj

```
procedure AttrtypFromSchemaObj(o: DSName): ATTRTYP
```

Given the `dsname o` of an [attributeSchema](#) or [classSchema](#) object, the `AttrtypFromSchemaObj` procedure returns the [ATTRTYP](#) that identifies this schema object on this DC.

```

if o!msDS-IntId ≠ null then
    return o!msDS-IntId
endif
if attributeSchema in o!objectClass then
    return MakeAttid(dc.prefixTable, o!attributeID)
else
    return MakeAttid(dc.prefixTable, o!governsID)
endif

```

5.15 ATTRVAL

The **ATTRVAL** structure defines a concrete type for the value of a single attribute.

```

typedef struct {
    [range(0,10485760)] ULONG valLen;
    [size_is(valLen)] UCHAR* pVal;
} ATTRVAL;

```

valLen: The size, in bytes, of the pVal array.

pVal: The value of the attribute. The encoding of the attribute varies by syntax, as described in the following sections.

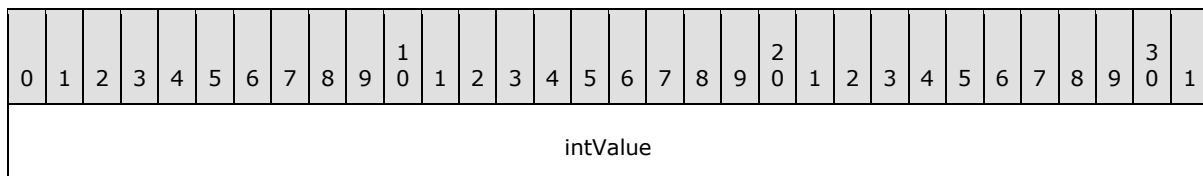
5.15.1 Concrete Value Representations

This section defines types used for concrete value representations. In addition to the types described here, the following types are also used for concrete value representations:

- [ATTRTYP \(section 5.13\)](#)
- [DSNAME \(section 5.44\)](#)
- [DSTIME \(section 5.45\)](#)
- [SYNTAX_ADDRESS \(section 5.170\)](#)
- [SYNTAX_DISTNAME_BINARY \(section 5.171\)](#)

5.15.1.1 INT32

The INT32 type is a 4-byte integer in little-endian form. See [\[MS-DTYP\]](#) section **2.2** for its definition.



intValue: A 32-bit signed number in little-endian byte order.

5.15.1.2 INT64

The INT64 type is an 8-byte integer in little-endian form. See [\[MS-DTYP\]](#) section 2.2 for its definition.

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
int64Value																															
...																															

int64Value: A 64-bit signed number in little-endian byte order.

5.15.1.3 OctetString

The OctetString represents an array of bytes. The number of bytes in the array equals the **valLen** field of the [ATTRVAL](#) structure.

5.15.1.4 String8

The String8 type is an array of ASCII characters. Each character is encoded as a single byte. The number of bytes in the array equals the **valLen** field of the [ATTRVAL](#) structure.

5.15.1.5 String16

The String16 type is an array of Unicode characters. Each Unicode character is encoded as 2 bytes. The number of bytes in the array equals the **valLen** field of the [ATTRVAL](#) structure. The byte ordering is little-endian.

5.15.1.6 SECURITY_DESCRIPTOR

The SECURITY_DESCRIPTOR structure is an NT security descriptor in self-relative format, as specified in [\[MS-DTYP\]](#) section 2.4.6. The data is stored in little-endian byte order.

5.15.1.7 SID

The SID is an NT security identifier structure, as specified in [\[MS-DTYP\]](#) section 2.4.2. The data is stored in little-endian byte order.

5.15.2 Abstract Value Representations

The abstract data model utilizes a representation of data values that are used by LDAP, minus the basic encoding rules (BER) encoding. Several of these syntaxes are adopted from [\[RFC2252\]](#).

The following table lists all the supported syntaxes and how they are represented in the model. The syntaxes that share an OID are identified by name, not OID.

LDAP syntax name (OID)	[RFC2252] name	Reference section in [RFC2252] or in this document
Boolean (2.2.5.8)	Boolean	[RFC2252] section 6.4

LDAP syntax name (OID)	[RFC2252] name	Reference section in [RFC2252] or in this document
Enumeration (2.5.5.9)	INTEGER	[RFC2252] section 6.16
Integer (2.5.5.9)	INTEGER	[RFC2252] section 6.16
LargeInteger (2.5.5.16)	INTEGER	[RFC2252] section 6.16
Object(Presentation-Address) (2.5.5.13)	Presentation Address	[RFC2252] section 6.28
Object(Replica-Link) (2.5.5.10)	Binary	[RFC2252] section 6.2
String(IA5) (2.5.5.5)	IA5 String	[RFC2252] section 6.15
String(Numeric) (2.5.5.6)	Numeric String	[RFC2252] section 6.23
String(Object-Identifier) (2.5.5.2)	OID	[RFC2252] section 6.25
String(Octet) (2.5.5.10)	Binary	[RFC2252] section 6.2
String(Printable) (2.5.5.5)	Printable String	[RFC2252] section 6.29
String(Unicode) (2.5.5.12)	Directory String	[RFC2252] section 6.10
String(UTC-Time) (2.5.5.11)	UTC Time	[RFC2252] section 6.31
String(Generalized-Time) (2.5.5.11)	Generalized Time	[RFC2252] section 6.14
Object(DS-DN) (2.5.5.1)	-	Section 5.15.2.1
Object(DN-String) (2.5.5.14)	-	Section 5.15.2.2
Object(DN-Binary) (2.5.5.7)	-	Section 5.15.2.3
Object(Access-Point) (2.5.5.14)	-	Section 5.15.2.4
Object(OR-Name) (2.5.5.7)	-	Section 5.15.2.5
String(NT-Sec-Desc) (2.5.5.15)	-	Section 5.15.2.6
String(SID) (2.5.5.17)	-	Section 5.15.2.7
String(Teletex) (2.5.5.4)	-	Section 5.15.2.8

The LDAP syntaxes that are not defined in [\[RFC2252\]](#) are described in the following sections.

5.15.2.1 Object(DS-DN)

A value with the Object(DS-DN) syntax is a UTF-8 string in the following format:

<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 in its binary format (as specified in [\[MS-DTYP\]](#) section **2.4.2**).
- *dn* is the string representation of a DN (as specified by [\[RFC2252\]](#) section 6.9, and further specified by [\[RFC2253\]](#)).

For reference to objects that do not have an [objectSid](#), the format is as follows:

<GUID=*guid_value*>;*dn*

where *guid_value* and *dn* have the same meaning as in the previous case.

5.15.2.2 Object(DN-String)

A value with the Object(DN-String) syntax is a UTF-8 string in the following format:

S:*char_count*:*string_value*:*object_DN*

where:

- *char_count* is the number of characters in the *string_value* string.
- *object_DN* is an object reference in the format of Object(DS-DN).

5.15.2.3 Object(DN-Binary)

A value with the Object(DN-Binary) syntax is a UTF-8 string in the following format:

B:*char_count*:*binary_value*:*object_DN*

where:

- *char_count* is the number of hexadecimal digits in *binary_value*.
- *binary_value* is the hexadecimal representation of a binary value.
- *object_DN* is an object reference in the format of Object(DS-DN).

5.15.2.4 Object(Access-Point)

A value with the Object(Access-Point) 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.
- *object_DN* is an object reference in the format of Object(DS-DN).

5.15.2.5 Object(OR-Name)

A value with the Object(OR-Name) syntax is a UTF-8 string in the following format:

X400:*OR_Name*#X500:*object_DN*

where:

- *OR_Name* is an X.400 O/R name in the format described in [\[RFC1327\]](#) section 4.1.
- *object_DN* is an object reference in the format of Object(DS-DN).

Alternatively, the *OR_Name* portion may be omitted, in which case the value contains only the *object_DN*.

5.15.2.6 String(NT-Sec-Desc)

A value with the String(NT-Sec-Desc) syntax contains a Windows security descriptor in self-relative binary form. The binary form is that of a [SECURITY_DESCRIPTOR](#) structure and is documented in [\[MS-DTYP\]](#) section 2.4.6.

5.15.2.7 String(Sid)

A value with the String(Sid) syntax is a Windows SID in binary form. The binary form is that of a SID structure and is specified in [\[MS-DTYP\]](#) section 2.4.2.

5.15.2.8 String(Teletex)

A value with the String(Teletex) syntax is a UTF-8 string restricted to characters with values between 0x20 and 0x7e, inclusive.

5.15.3 Converting Between Abstract and Concrete Value Representations

The type [ATTRVAL](#) is an encoding that several methods use to send individual directory attribute values across the network. When an attribute has multiple values, and all those values need to be sent, this is performed by sending multiple [ATTRVALs](#).

An [ATTRVAL](#) that encodes an OID requires a prefix table for decoding. In some cases the prefix table accompanies the [ATTRVAL](#) in the same RPC request or response. In other cases, a predefined prefix table is sufficient. The process of creating the [ATTRVAL](#) for an OID can add an entry to the prefix table that will accompany the [ATTRVAL](#).

The abstract directory model specified in [\[MS-ADTS\]](#) section 3.1.1 represents individual attribute values in the form used by LDAP (see [\[RFC2252\]](#)), minus the BER encoding. In short, values are represented as strings in a variety of formats. The abstract type [Value](#) is used to represent an attribute value in the model. Section 5.15.2 specifies the abstract representation for each LDAP syntax.

Therefore, this specification requires procedures that convert between the concrete [ATTRVAL](#) encoding and the abstract [Value](#) encoding, creating a prefix table while creating the [ATTRVAL](#), and reading a prefix table while decoding the [ATTRVAL](#). The signatures of these procedures include the following:

```
procedure ATTRVALFromValue(
    v: Value, s: AttributeSyntax, var t: PrefixTable) : ATTRVAL
procedure ValueFromATTRVAL(
    a: ATTRVAL, s: AttributeSyntax, t: PrefixTable) : Value
```

where:

- *s* is an LDAP attribute syntax from the table in section 5.15.2.
- *t* is an abstract [PrefixTable](#) object, representing a prefix table.

[ATTRVALFromValue](#) mutates its input [PrefixTable](#) object `t`; [ValueFromATTRVAL](#) does not.

Apart from the prefix table complication, these two procedures are straightforward given the two value representations. These procedures obey the mappings shown in the following table for converting between abstract and concrete value representations.

LDAP syntax name	Encoding of ATTRVAL payload
Boolean (2.2.5.8)	INT32
Enumeration (2.5.5.9)	INT32
Integer (2.5.5.9)	INT32
LargeInteger (2.5.5.16)	INT64
Object(Presentation-Address) (2.5.5.13)	SYNTAX_ADDRESS
Object(Replica-Link) (2.5.5.10)	OctetString
String(IA5) (2.5.5.5)	String8
String(Numeric) (2.5.5.6)	String8
String(Object-Identifier) (2.5.5.2)	ATTRTYP
String(Octet) (2.5.5.10)	OctetString
String(Printable) (2.5.5.5)	String8
String(Unicode) (2.5.5.12)	String16
String(UTC-Time) (2.5.5.11)	DSTIME
String(Generalized-Time) (2.5.5.11)	DSTIME
Object(DS-DN) (2.5.5.1)	DSName
Object(DN-String) (2.5.5.14)	SYNTAX_DISTNAME_BINARY
Object(DN-Binary) (2.5.5.7)	SYNTAX_DISTNAME_BINARY
Object(Access-Point) (2.5.5.14)	SYNTAX_DISTNAME_BINARY
Object(OR-Name) (2.5.5.7)	SYNTAX_DISTNAME_BINARY
String(NT-Sec-Desc) (2.5.5.15)	SECURITY_DESCRIPTOR
String(SID) (2.5.5.17)	SID
String(Teletex) (2.5.5.4)	String8

Since the preceding procedures require a prefix table, a procedure to produce a prefix table is also required, as follows:

```
procedure NewPrefixTable() : PrefixTable
```


The special case value conversion between [ATTRTYP](#) and [OID](#) is provided by the following two procedures:

```
procedure MakeAttid(t: PrefixTable, o: OID) : ATTRTYP
procedure OidFromAttid(t: PrefixTable, attr: ATTRTYP) : OID
```

These three procedures, specified in section [5.15.4](#), describe the algorithm for converting values of type [OID](#) to and from their [ATTRVAL](#) payload representation using a [PrefixTable](#).

The conversion between an abstract [Value](#) representation and a concrete [ATTRVAL](#) representation is specified in the following sections, which are organized by abstract value type. In the examples shown, LDAP Value: represents the LDAP value; valLen: represents the value in the **valLen** field of the [ATTRVAL](#) structure; and payload: represents the data in payload (the referent of pVal in the [ATTRVAL](#) structure).

Because prefix tables are communicated over the wire, the [ConcretePTFromAbstractPT](#) and [AbstractPTFromConcretePT](#) procedures are defined to convert between the abstract [PrefixTable](#) and the concrete [SCHEMA_PREFIX_TABLE](#).

5.15.3.1 Boolean

The Boolean LDAP attribute value FALSE corresponds to an INT32 with value 0. The Boolean LDAP attribute value TRUE corresponds to an [INT32](#) with a nonzero value. INT32 is in little-endian format. The **valLen** field of [ATTRVAL](#) equals 4.

Example:

```
LDAP value: TRUE
INT32 value 0x1

valLen: 4.

payload:
01 00 00 00          . . . .
```

5.15.3.2 Enumeration and Integer

The Enumeration and Integer LDAP syntax types are represented in the same manner. The LDAP representation of the integer as a string expressed in base-10 notation corresponds to an [INT32](#), which is in little-endian format. The **valLen** field of [ATTRVAL](#) equals 4.

Example:

```
LDAP value: 5
INT32 value: 0x5
valLen: 4,
payload:
05 00 00 00          . . . .
```

5.15.3.3 LargeInteger

The LDAP representation of the integer as a string expressed in base-10 notation corresponds to an [INT64](#), which is in little-endian format. The **valLen** field of [ATTRVAL](#) equals 8.

Example:

```
LDAP value: 12605
INT64 value: Hexadecimal 0x313d
valLen: 8,
payload:
3d 31 00 00 00 00 00 00      =1.....
```

5.15.3.4 Object(Presentation-Address)

To convert from the LDAP representation to the [SYNTAX_ADDRESS](#) representation, the UTF-8 encoded string is converted to a UCS-16 encoded Unicode string. The resulting string is not null-terminated. The **dataLen** field of [SYNTAX_ADDRESS](#) equals the length of the Unicode string in bytes, plus 4. The **valLen** field of [ATTRVAL](#) equals the **dataLen** field of [SYNTAX_ADDRESS](#).

Example:

```
LDAP value: 12345 (Unicode string encoded as UTF-8)
This represents the following SYNTAX_ADDRESS struct:

+0x000 dataLen      : 0xe
+0x004 uVal         : L"12345"
valLen: 14
payload:
0e 00 00 00 31 00 32 00 33 00 34 00 35 00      ....1.2.3.4.5.
```

5.15.3.5 Object(Replica-Link) String (Octet)

The representation used in LDAP syntax and encoding of the [ATTRVAL](#) payload is the same. Therefore, the payload is set to the same value. The **valLen** field of [ATTRVAL](#) equals the length of the byte array.

5.15.3.6 String(IA5) String(Printable) String(Numeric) String(Teletex)

The representation used in LDAP syntax and encoding of the [ATTRVAL](#) payload is the same. Therefore, the payload is set to the same value. The string is not null-terminated. The **valLen** field of [ATTRVAL](#) equals the number of bytes in the string.

Example:

```
LDAP value: 123456789
This represents an ASCII string "123456789"
```

valLen: 9

payload:

5.15.3.7 String(Unicode)

To convert from the LDAP representation to the Unicode syntax representation, the UTF-8 encoded string is converted to a UCS-16 encoded [String16](#). Each Unicode character is in little-endian format. The resulting string is not null-terminated. The `valLength` field of [ATTRVAL](#) equals the total number of bytes in the Unicode string.

Example:

```
LDAP value: Administrator (Unicode string encoded in UTF-8)
valLen: 26
payload:
41 00 64 00 6d 00 69 00 6e 00 69 00 73 00 74 00  A.d.m.i.n.i.s.t.
72 00 61 00 74 00 6f 00 72 00                    r.a.t.o.r.
```

5.15.3.8 String(Object-Identifier)

Conversion from the LDAP representation to [ATTRTYP](#) is performed via the [MakeAttid](#) function. The length of the `valLen` field in [ATTRVAL](#) equals 4.

Conversion from [ATTRTYP](#) to LDAP representation is performed by the [OidFromAttid](#) function.

Example:

```
LDAP value: 2.5.6.5
This corresponds to ATTRTYP value: 0x00010005.
valLen: 4
payload:
05 00 01 00                                     ....
```

5.15.3.9 String(UTC-Time) and String(Generalized-Time)

For both the [String\(UTC-Time\)](#) and [String\(Generalized-Time\)](#) LDAP syntaxes, the time expressed in the LDAP value corresponds to [DSTIME](#). It is in little-endian format. The `valLen` field of [ATTRVAL](#) equals 8.

```
LDAP value: 20060609211106.0Z (06/09/2006 14:11:06 PST).
This corresponds to DSTIME value: 0x2fa9a74ea
valLen: 8,
payload:
ea 74 9a fa 02 00 00 00                        .t.....
```

5.15.3.10 Object(DS-DN)

The LDAP representation of [Object\(DS-DN\)](#) is defined in section [5.15.2.1](#). This corresponds to [DSName](#) as follows:

The *dn* part of the LDAP representation is converted to a UCS-16 encoded Unicode string. Then, the *attributeValue* component (defined in [\[RFC2253\]](#)) of each RDN in the DN is canonicalized according to the following rules:

- The first leading space, if any, is escaped as a backslash (\) followed by a space.
- Any carriage return or line-feed characters are escaped as a backslash followed by the 2-digit hexadecimal value of that character, as specified in [\[RFC2253\]](#) section 2.4.
- Any of the following characters—number sign (#), plus sign (+), comma (,), semicolon (;), quotation mark ("), left angle bracket (<), equal sign (=), right angle bracket (>), and backslash (\)—are escaped as a backslash followed by the character.
- The trailing space, if any, is escaped as a backslash followed by a space.

The resulting string (including a terminating NULL character) is inserted into the **StringName** field of the **DSNAME**. The length of the string, in Unicode characters, is inserted into the **NameLen** field. The length of the string in the **NameLen** field does not include the terminating NULL character. The value of *guid_value* in LDAP representation is expressed as a GUID and inserted into the **Guid** field of the **DSNAME** structure. If the *sid_value* is present, it is copied into the **Sid** field of the **DSNAME** and the **SidLen** field is set to the length, in bytes, of the SID. If the *sid_value* part is not present, then the **SidLen** field is set to 0. The **valLen** field of **ATTRVAL** equals the length of the **DSNAME** structure. All the multibyte quantities in the **DSNAME** are stored in little-endian format.

Example:

```
LDAP Value:
<GUID=alb4ea3c47fc714a8195454faa6423a3>;<SID=01050000000000051500000089598d33d3c56b6894e1
f2e6f4010000>;CN=Administrator,OU=Users,DC=test,DC=com
```

This corresponds to the following DSNAME:

```
+0x000 structLen      : 0x8a
+0x004 SidLen         : 0x1c
+0x008 Guid           : 3ceab4a1-fc47-4a71-8195-454faa6423a3
+0x018 Sid            : S-1-5-21-864901513-1751893459-3874677140-500
+0x034 NameLen        : 0x28
+0x038 StringName     : L"CN=Administrator,OU=Users,DC=test,DC=com"
```

valLen: 138,

```
payload:
8a 00 00 00 1c 00 00 00 a1 b4 ea 3c 47 fc 71 4a .....<G.qJ
81 95 45 4f aa 64 23 a3 01 05 00 00 00 00 00 05 ..EO.d#.....
15 00 00 00 89 59 8d 33 d3 c5 6b 68 94 e1 f2 e6 .....Y.3..kh....
f4 01 00 00 28 00 00 00 43 00 4e 00 3d 00 41 00 ....(...C.N.=.A.
64 00 6d 00 69 00 6e 00 69 00 73 00 74 00 72 00 d.m.i.n.i.s.t.r.
61 00 74 00 6f 00 72 00 2c 00 4f 00 55 00 3d 00 a.t.o.r.,.O.U.=.
55 00 73 00 65 00 72 00 73 00 2c 00 44 00 43 00 U.s.e.r.s.,.D.C.
3d 00 74 00 65 00 73 00 74 00 2c 00 44 00 43 00 =.t.e.s.t.,.D.C.
3d 00 63 00 6f 00 6d 00 00 00 .....=.c.o.m....
```

5.15.3.11 Object(DN-Binary)

The LDAP representation of the attribute value corresponds to [SYNTAX_DISTNAME_BINARY](#). The *object_DN* portion of the LDAP representation is treated as if it were in Object(DS-DN) syntax and converted to the [DSNAME](#) syntax representation, as explained in section 5.15.2.3. The *binary_value* portion of the LDAP representation is converted to the binary value (an array of bytes) and stored in the *byteVal* field of the [SYNTAX_ADDRESS](#) structure. The *dataLen* field of [SYNTAX_ADDRESS](#) is set to the length of the array, in bytes, plus 4, where 4 is the length of the *dataLen* field.

Padding is added between the [DSNAME](#) and [SYNTAX_ADDRESS](#) structures so that the length of [DSNAME](#) plus padding modulo 4 equals 0. The padding is an array of bytes, each byte of value 0. The *valLen* field of [ATTRVAL](#) equals the length of the [DSNAME](#) structure, plus the number of bytes added for padding, plus the length of the [SYNTAX_ADDRESS](#) structure.

All the multibyte quantities in the [DSNAME](#) and [SYNTAX_ADDRESS](#) structures are stored in little-endian format.

Example where padding is required because [DSNAME](#) is not 4-byte aligned:

```
LDAP value:
B:8:00000005:<GUID=2d8b0ce6-aa32-4f31-a6e8-
88343e6244a5>;<SID=010100001cd509a018459359>;DC=test,DC=com

Representation of data as SYNTAX_DISTNAME_BINARY:
+0x000 Name          : DSNAME
  +0x000 structLen    : 0x56
  +0x004 SidLen       : 0xc
  +0x008 Guid         : 2d8b0ce6-aa32-4f31-a6e8-88343e6244a5
  +0x018 Sid          : S-1-483723680-1502823704
  +0x034 NameLen      : 0xe
  +0x038 StringName   : "DC=test,DC=com"
+0x058 Data          : SYNTAX_ADDRESS
  +0x000 dataLen      : 8
  +0x004 byteVal      : 00 00 00 05
valLength: 96
payload:
56 00 00 00 0c 00 00 00 e6 0c 8b 2d 32 aa 31 4f V.....-2.10
a6 e8 88 34 3e 62 44 a5 01 01 00 00 1c d5 09 a0 ...4>bD.....
18 45 93 59 00 00 00 00 00 00 00 00 00 00 00 00 .E.Y.....
00 00 00 00 0e 00 00 00 44 00 43 00 3d 00 74 00 .....D.C.=.t.
65 00 73 00 74 00 2c 00 44 00 43 00 3d 00 63 00 e.s.t.,.D.C.=.c.
6f 00 6d 00 00 00 00 00 08 00 00 00 00 00 00 05 o.m.....
```

Example where padding is not required because [DSNAME](#) is 4-byte aligned:

```
LDAP value:
B:8:0000000D:<GUID= ff432fe0-8c94-43cf-915c-
286b197b0164>;<SID=010100001a180dba5ec27614>;DC=test1,DC=test,DC=com.

Representation of data as SYNTAX_DISTNAME_BINARY:
+0x000 Name          : DSNAME
  +0x000 structLen    : 0x68
  +0x004 SidLen       : 0xc
  +0x008 Guid         : ff432fe0-8c94-43cf-915c-286b197b0164
  +0x018 Sid          : S-1-437783994-343327326
```

```

+0x034 NameLen          : 0x17
+0x038 StringName       : "DC=test1,DC=test,DC=com"
+0x068 Data             : SYNTAX_ADDRESS
+0x000 dataLen          : 0x74003d
+0x004 byteVal          : 00 00 00 0d
68 00 00 00 0c 00 00 00 e0 2f 43 ff 94 8c cf 43  h...../C....C
91 5c 28 6b 19 7b 01 64 01 01 00 00 1a 18 0d ba  .\ (k.{.d.....
5e c2 76 14 00 00 00 00 00 00 00 00 00 00 00 00  ^..v.....
00 00 00 00 17 00 00 00 44 00 43 00 3d 00 74 00  .....D.C.=.t.
65 00 73 00 74 00 31 00 2c 00 44 00 43 00 3d 00  e.s.t.l.,.D.C.=.
74 00 65 00 73 00 74 00 2c 00 44 00 43 00 3d 00  t.e.s.t.,.D.C.=.
63 00 6f 00 6d 00 00 00 08 00 00 00 00 00 00 0d  c.o.m.....

```

5.15.3.12 Object(DN-String)

The LDAP representation of the attribute value corresponds to [SYNTAX_DISTNAME_BINARY](#). The *object_DN* portion of the LDAP representation is treated as if it were in Object(DS-DN) syntax and converted to the [DSNAME](#) representation, as explained in section [5.15.2.2](#). The result is stored in the Name field of that structure. The *string_value* portion of the LDAP representation is converted to a UCS-16 encoded Unicode string and stored in the byteVal field of the [SYNTAX_ADDRESS](#) structure. The dataLen field of [SYNTAX_ADDRESS](#) is set to the length of the string, in bytes, plus 4, where 4 is the length of the dataLen field. Enough padding is added between the [DSNAME](#) and [SYNTAX_ADDRESS](#) structures, so that the length of [DSNAME](#) plus padding modulo 4 equals 0. The padding is an array of bytes of value 0. The valLen field of [ATTRVAL](#) equals the length of the [DSNAME](#) structure, plus the number of bytes added for padding, plus the length of the [SYNTAX_ADDRESS](#) structure. All the multibyte quantities in the [DSNAME](#) and [SYNTAX_ADDRESS](#) structures are stored in little-endian format.

Example:

```

LDAP value:
S:7:Unicode:<GUID=2d8b0ce6-aa32-4f31-a6e8-
88343e6244a5>;<SID=010100001cd509a018459359>;DC=test,DC=com

```

```

This represents data as SYNTAX_DISTNAME_BINARY
(note the structure SYNTAX_ADDRESS is 4-byte aligned):
+0x000 Name             : DSNAME
+0x000 structLen        : 0x56
+0x004 SidLen           : 0xc
+0x008 Guid             : 2d8b0ce6-aa32-4f31-a6e8-88343e6244a5
+0x018 Sid              : S-1-483723680-1502823704
+0x034 NameLen          : 0xe
+0x038 StringName       : "DC=test,DC=com"
+0x058 Data             : SYNTAX_ADDRESS
+0x000 dataLen          : 0x12
+0x004 uVal             : "Unicode"
valLength: 106
payload:
56 00 00 00 0c 00 00 00 e6 0c 8b 2d 32 aa 31 4f  V.....-2.10
a6 e8 88 34 3e 62 44 a5 01 01 00 00 1c d5 09 a0  ...4>bD.....
18 45 93 59 00 00 00 00 00 00 00 00 00 00 00 00  .E.Y.....
00 00 00 00 0e 00 00 00 44 00 43 00 3d 00 74 00  .....D.C.=.t.
65 00 73 00 74 00 2c 00 44 00 43 00 3d 00 63 00  e.s.t.,.D.C.=.c.
6f 00 6d 00 00 00 00 00 12 00 00 00 55 00 6e 00  o.m.....U.n.

```

5.15.3.13 Object(OR-Name)

The LDAP representation of the attribute value corresponds to [SYNTAX DISTNAME BINARY](#). The *object_DN* portion of the LDAP representation is treated as if it were in Object(DS-DN) syntax and converted to the [DSNAME](#) syntax representation, as explained in section [5.15.2.5](#). The *OR_Name* portion of the LDAP representation, if present, is converted to a UCS-16 encoded Unicode string and stored in the *byteVal* field of the [SYNTAX ADDRESS](#) structure. [<25>](#) The *dataLen* field of [SYNTAX ADDRESS](#) equals the length of data, in bytes, plus 4. All the multibyte quantities in the [DSNAME](#) and [SYNTAX ADDRESS](#) structures are stored in little-endian format.

5.15.3.14 Object(Access-Point)

The LDAP representation of the attribute value corresponds to [SYNTAX DISTNAME BINARY](#). The *object_DN* portion of the LDAP representation is treated as if it were in Object(DS-DN) syntax and converted to the [DSNAME](#) syntax representation, as explained in section [5.15.2.4](#). The *presentation_address* portion of the LDAP representation is treated as if it were in the [Object\(Presentation-Address\)](#) syntax and converted to the [SYNTAX ADDRESS](#) representation. All the multibyte quantities in the [DSNAME](#) and [SYNTAX ADDRESS](#) structures are stored in little-endian format.

5.15.3.15 String(Sid)

The representation used in LDAP syntax and encoding of [ATTRVAL](#) payload is the same. Therefore the payload is set to the same value. The *valLen* field of [ATTRVAL](#) equals the number of bytes in the payload. It is always 28. All the multibyte quantities in the [SID](#) structure are stored in little-endian format.

Example:

```
LDAP Value: 01050000000000051500000089598d33d3c56b6894e1f2e6f4010000
valLen: 28
payload:
01 05 00 00 00 00 05 15 00 00 00 89 59 8d 33 .....Y.3
d3 c5 6b 68 94 e1 f2 e6 f4 01 00 00 ..kh.....
```

5.15.3.16 String(NT-Sec-Desc)

The representation used in LDAP syntax and encoding of [ATTRVAL](#) payload is the same. Therefore the payload is set to the same value. The *valLen* field of [ATTRVAL](#) equals the number of bytes in the payload. All the multibyte quantities in the security descriptor structure are stored in little-endian format.

```
LDAP value: (binary blob, represented in hex format here)
0100048c700000008000000000000001400000004005c0003000000050028000001000001000000531a72ab2
f1ed011981900aa0040529b01010000000000050a00000000121800ff010f0001020000000000052000000020
020000001214009400020001010000000000050b000000010200001cd509a01845935900020000010200001cd
509a01845935900020000
```

```
This represents the following self-relative security descriptor
value:
SD Revision: 1
```

```

SD Control: 0x8c04
    SE_DACL_PRESENT
    SE_DACL_AUTO_INHERITED
    SE_SACL_AUTO_INHERITED
    SE_SELF_RELATIVE
Owner: S-1-483723680-1502823704-512
Group: S-1-483723680-1502823704-512
DACL:
    Revision      4
    Size:         92 bytes
    # Aces:       3
    Ace[0]
        Ace Type: 0x5 - ACCESS_ALLOWED_OBJECT_ACE_TYPE
        Ace Size: 40 bytes
        Ace Flags: 0x0
        Object Ace Mask: 0x00000100
            ACTRL_DS_CONTROL_ACCESS
        Object Ace Flags: 0x1
            ACE_OBJECT_TYPE_PRESENT
        Object Ace Type:
            Change Password-ab721a53-1e2f-11d0-9819-00aa0040529b
        Object Ace Sid: NT AUTHORITY\SELF [S-1-5-10]
    Ace[1]
        Ace Type: 0x0 - ACCESS_ALLOWED_ACE_TYPE
        Ace Size: 24 bytes
        Ace Flags: 0x12
            CONTAINER_INHERIT_ACE
            INHERITED_ACE
        Ace Mask: 0x000f01ff
            DELETE
            READ_CONTROL
            WRITE_DAC
            WRITE_OWNER
            ACTRL_DS_CREATE_CHILD
            ACTRL_DS_DELETE_CHILD
            ACTRL_DS_LIST
            ACTRL_DS_SELF
            ACTRL_DS_READ_PROP
            ACTRL_DS_WRITE_PROP
            ACTRL_DS_DELETE_TREE
            ACTRL_DS_LIST_OBJECT
            ACTRL_DS_CONTROL_ACCESS
        Ace Sid: BUILTIN\Administrators [S-1-5-32-544]
    Ace[2]
        Ace Type: 0x0 - ACCESS_ALLOWED_ACE_TYPE
        Ace Size: 20 bytes
        Ace Flags: 0x12
            CONTAINER_INHERIT_ACE
            INHERITED_ACE
        Ace Mask: 0x00020094
            READ_CONTROL
            ACTRL_DS_LIST
            ACTRL_DS_READ_PROP
            ACTRL_DS_LIST_OBJECT
        Ace Sid: NT AUTHORITY\Authenticated Users [S-1-5-11]
valLen: 144
payload:
01 00 04 8c 70 00 00 00 80 00 00 00 00 00 00 00 ....p.....

```



```

14 00 00 00 04 00 5c 00 03 00 00 00 05 00 28 00 .....\.
00 01 00 00 01 00 00 00 53 1a 72 ab 2f 1e d0 11 .....S.r./...
98 19 00 aa 00 40 52 9b 01 01 00 00 00 00 00 05 .....@R.....
0a 00 00 00 00 12 18 00 ff 01 0f 00 01 02 00 00 .....
00 00 00 05 20 00 00 00 20 02 00 00 00 12 14 00 .....
94 00 02 00 01 01 00 00 00 00 05 0b 00 00 00 .....
01 02 00 00 1c d5 09 a0 18 45 93 59 00 02 00 00 .....E.Y....
01 02 00 00 1c d5 09 a0 18 45 93 59 00 02 00 00 .....E.Y....

```

5.15.4 ATTRTYP-to-OID Conversion

This section describes the prefix mapping mechanism that allows the one-to-one mapping between OIDs and a 32-bit integer ([ATTRTYP](#)).

An OID can be represented in the binary form, with a BER encoding scheme. The standard BER encoding consists of four components. Only the third component (contents octets) is used here; other components are omitted.

Note The BER encoding of an OID is described in [ITUX690] section 8.19. To avoid ambiguity, the nonencoded form of the OID is referred to as the original form in this section.

The prefix of an OID is the binary OID, excluding the last one or two bytes. If the number following the final period (.) in the original form of the OID is less than 128, only the last byte is excluded; otherwise, the last two bytes are excluded.

A **PrefixTable** is a sequence of tuples defined as follows:

```

type PrefixTable = sequence of [
    prefixString: unistring,
    prefixIndex: integer
]

```

where:

- prefixString is the prefix of an OID.
- prefixIndex is an integer in the range [0 .. 0x0000ffff].

The integer prefixIndex is called the prefix index of prefixString. To allow one-to-one mappings between the prefix strings and the prefix indexes in the table, each prefixString **MUST** occur at most once in the table, and each prefixIndex **MUST** occur at most once in the table.

An [ATTRTYP](#) is a 32-bit, unsigned integer. If attr is an [ATTRTYP](#), define attr.upperWord to be the most significant 16 bits, and attr.lowerWord to be the least significant 16 bits.

The following types and helper procedures are used for mapping between OIDs and [ATTRTYP](#).

```

procedure ToBinary(st: unistring) : sequence of BYTE

```

Converts a string to a binary OID representation. For example, "\x55\x06" is the binary OID \x55\x06.

```
procedure CatBinary(o: sequence of BYTE, b: BYTE) : sequence of BYTE
```

Concatenates a byte onto a binary OID. For example, \x02 concatenated onto \x55\x06 is \x55\x06\x02.

```
procedure ToStringOID(o: sequence of BYTE) : unicodestring
```

Converts a binary OID to its string representation, as described in [\[ITU690\]](#) section 8.19; returns null if the conversion fails. For example, the binary OID \x55\x06\x02 is converted to the OID string "2.5.6.2".

```
procedure ToBinaryOID(s: unicodestring) : sequence of BYTE
```

Converts an OID string representation to a binary OID, as described in [\[ITU690\]](#) section 8.19; returns null if the conversion fails. For example, the OID string "2.5.6.2" is converted to the binary form \x55\x06\x02.

```
procedure ToByte(i: integer) : BYTE
```

Converts an integer into a byte representation, truncating to the least significant digits, if needed. For example, 2 converts to \x02.

```
procedure SubBinary(b: sequence of BYTE,  
    start: integer, end: integer) : sequence of BYTE
```

Returns the sequence [start .. end] of bytes in b.

```
procedure AddPrefixTableEntry(var t: PrefixTable, o: sequence of BYTE)
```

Sets t[t.length].prefixString to o. Generates a random number between 0 and 65535 that is unique in the values of prefixIndex in t, and sets t[t.length].prefixIndex to the generated random number. Increases t.length by one.

```
procedure ToInteger(s: unicodestring) : integer
```

Converts a string to its integer representation. For example, "127" is 127. Strings with nonnumeric characters are not defined for this procedure.

The following procedures are used for mapping between object identifiers and [ATTRTYP](#) representations.

```
procedure MakeAttid(var t: PrefixTable, o: OID): ATTRTYP
```

Informative summary of behavior: This procedure converts an OID to a corresponding [ATTRTYP](#) representation.

```

lastValueString: unicodestring
lastValue, lowerWord: integer
binaryOID, oidPrefix: sequence of BYTE
attr: ATTRTYP
pos: integer
/* get the last value in the original OID: the value
 * after the last '.' */
lastValueString := SubString(o,
                             FindCharRev(o, o.length, '.'),
                             o.length)
lastValue := ToInteger(lastValueString)
/* convert the dotted form of OID into a BER encoded binary
 * format. The BER encoding of OID is described in section
 * 8.19 of [ITUX690] */
binaryOID := ToBinaryOid(o)
/* get the prefix of the OID */
if lastValue < 128 then
    oidPrefix := SubBinary(binaryOID, 0, binaryOID.length - 1)
else
    oidPrefix := SubBinary(binaryOID, 0, binaryOID.length - 2)
endif
/* search the prefix in the prefix table, if none found, add
 * one entry for the new prefix. */
fToAdd := true
for i := 0 to t.length
    if ToBinary(t[i].prefixString) = oidPrefix then
        fToAdd := false
        pos := i
    endif
endfor
if fToAdd then
    pos := t.length
    AddPrefixTableEntry(t, oidPrefix)
endif
/*compose the attid*/
lowerWord := lastValue mod 16384
if lastValue ≥ 16384 then
    /*mark it so we know it is not the whole lastValue*/
    lowerWord := lowerWord + 32768
endif
upperWord := t[pos].prefixIndex
attr := upperWord * 65536 + lowerWord
return attr
procedure OidFromAttid(t: PrefixTable, attr: ATTRTYP): OID

```

Informative summary of behavior: This procedure converts an [ATTRTYP](#) representation to a corresponding OID.

```

i, upperWord, lowerWord: integer
binaryOID: sequence of BYTE
binaryOID = null
/* separate the ATTRTYP into two parts */
upperWord := attr / 65536
lowerWord := attr mod 65536
/* search in the prefix table to find the upperWord, if found,

```

```

    * construct the binary OID by appending lowerWord to the end of
    * found prefix.*/
    for i := 0 to t.length
        if t[i].prefixIndex = upperWord then
            if lowerWord < 128 then
                binaryOID := CatBinary(ToBinary(t[i].prefixString),
                    ToByte(lowerWord))
            else
                if lowerWord ≥ 32768 then
                    lowerWord := lowerWord - 32768
                endif
                binaryOID := CatBinary(ToBinary(t[i].prefixString),
                    ToByte(((lowerWord / 128) mod 128) + 128))
                binaryOID := CatBinary(binaryOID, ToByte(lowerWord mod 128))
            endif
        endif
    endfor
    if binaryOID = null then
        return null
    else
        return ToStringOID(binaryOID)
    endif
endprocedure NewPrefixTable( ): PrefixTable

```

This procedure creates a new **PrefixTable**, inserts the following tuples into the table, and returns the table as the result.

prefixString	Length of prefixString	prefixIndex
"\x55\x4"	2	0
"\x55\x6"	2	1
"\x2A\x86\x48\x86\xF7\x14\x01\x02"	8	2
"\x2A\x86\x48\x86\xF7\x14\x01\x03"	8	3
"\x60\x86\x48\x01\x65\x02\x02\x01"	8	4
"\x60\x86\x48\x01\x65\x02\x02\x03"	8	5
"\x60\x86\x48\x01\x65\x02\x01\x05"	8	6
"\x60\x86\x48\x01\x65\x02\x01\x04"	8	7
"\x55\x5"	2	8
"\x2A\x86\x48\x86\xF7\x14\x01\x04"	8	9
"\x2A\x86\x48\x86\xF7\x14\x01\x05"	8	10
"\x09\x92\x26\x89\x93\xF2\x2C\x64"	8	19
"\x60\x86\x48\x01\x86\xF8\x42\x03"	8	20
"\x09\x92\x26\x89\x93\xF2\x2C\x64\x01"	9	21

prefixString	Length of prefixString	prefixIndex
"\x60\x86\x48\x01\x86\xF8\x42\x03\x01"	9	22
"\x2A\x86\x48\x86\xF7\x14\x01\x05\xB6\x58"	10	23
"\x55\x15"	2	24
"\x55\x12"	2	25
"\x55\x14"	2	26

The following examples show the correspondence between [OID](#) and [ATTRTYP](#) by using the **PrefixTable** returned by the procedure [NewPrefixTable](#).

```

OID: 2.5.4.6 (countryName attribute)
Binary: \x55\x04\x06
Prefix string: "\x55\x04"
Prefix index: 0
ATTRTYP: 0x00000006
OID: 2.5.6.2 (country class)
Binary: \x55\x06\x02
Prefix string: "\x55\x06"
Prefix index: 1
ATTRTYP: 0x00010002
OID: 1.2.840.113556.1.2.1 (instanceType attribute)
Binary: \x2A\x86\x48\x86\xF7\x14\x01\x02\x01
Prefix string: "\x2A\x86\x48\x86\xF7\x14\x01\x02"
Prefix index: 2
ATTRTYP: 0x00020001
OID: 1.2.840.113556.1.3.23 (container class)
Binary: \x2A\x86\x48\x86\xF7\x14\x01\x03\x17
Prefix string: "\x2A\x86\x48\x86\xF7\x14\x01\x03"
Prefix index: 3
ATTRTYP: 0x00030017
OID: 2.5.5.1 (attribute syntax: distinguished name)
Binary: \x55\x5\x1
Prefix string: "\x55\x5"
Prefix index: 8
ATTRTYP: 0x00080001
OID: 1.2.840.113556.1.4.1 (RDN attribute)
Binary: \x2A\x86\x48\x86\xF7\x14\x01\x04\x01
Prefix string: "\x2A\x86\x48\x86\xF7\x14\x01\x04"
Prefix index: 9
ATTRTYP: 0x00090001
OID: 1.2.840.113556.1.5.1 (securityObject class)
Binary: \x2A\x86\x48\x86\xF7\x14\x01\x05\x01
Prefix string: "\x2A\x86\x48\x86\xF7\x14\x01\x05"
Prefix index: 10
ATTRTYP: 0x000a0001
OID: 0.9.2342.19200300.100.1.1 (uid attribute)
Binary: \x09\x92\x26\x89\x93\xF2\x2C\x64\x01\x01
Prefix string: "\x09\x92\x26\x89\x93\xF2\x2C\x64\x01"
Prefix index: 21
ATTRTYP: 0x00150001
OID: 2.16.840.1.113730.3.1.1 (carLicense attribute)
Binary: \x60\x86\x48\x01\x86\xF8\x42\x03\x01\x01
Prefix string: "\x60\x86\x48\x01\x86\xF8\x42\x03\x01"

```

```

Prefix index: 22
ATTRTYP: 0x00160001
OID: 1.2.840.113556.1.5.7000.53 (crossRefContainer class)
Binary: \x2A\x86\x48\x86\xF7\x14\x01\x05\xB6\x58\x35
Prefix string: "\x2A\x86\x48\x86\xF7\x14\x01\x05\xB6\x58"
Prefix index: 23
ATTRTYP: 0x00170035
OID: 2.5.21.2 (ditContentRules attribute)
Binary: \x55\x15\x02
Prefix string: "\x55\x15"
Prefix index: 24
ATTRTYP: 0x00180002
OID: 2.5.18.1 (createTimeStamp attribute)
Binary: \x55\x12\x01
Prefix string: "\x55\x12"
Prefix index: 25
ATTRTYP: 0x00190001
OID: 2.5.20.1 (subSchema class)
Binary: \x55\x14\x01
Prefix string: "\x55\x14"
Prefix index: 26
ATTRTYP: 0x001a0001

```

5.16 ATTRVALBLOCK

The **ATTRVALBLOCK** structure defines a concrete type for a sequence of attribute values.

```

typedef struct {
    [range(0,10485760)] ULONG valCount;
    [size_is(valCount)] ATTRVAL* pAVal;
} ATTRVALBLOCK;

```

valCount: The count of items in the pAVal array.

pAVal: The sequence of attribute values.

5.17 ATTRVALFromValue

```

procedure ATTRVALFromValue(v: Value,
                           s: Syntax,
                           var t: PrefixTable) : ATTRVAL

```

ATTRVALFromValue converts a value in the abstract [Value](#) encoding v of syntax s into a concrete [ATTRVAL](#), using the prefix table represented by t. This procedure may mutate the supplied prefix table.

See section [5.15.3](#) for the specification of this procedure.

5.18 BOOL

BOOL is a concrete type for a Boolean value, as specified in [\[MS-DTYP\]](#) section **2.2**.

5.19 BYTE

BYTE is a concrete type for a single byte, as specified in [\[MS-DTYP\]](#) section 2.2.

5.20 CHANGE_LOG_ENTRIES

CHANGE_LOG_ENTRIES is a concrete type, normatively specified in [\[MS-ADTS\]](#) section 3.1.1.7.3; the type of the pmsgOut.V1.pLog field of the [IDL_DRSGetNT4ChangeLog](#) response. The following five fields within this type are used in specifying [IDL_DRSGetNT4ChangeLog](#) server behavior:

Size: MUST be 0x00000010.

Version: MUST be 0x00000001.

SequenceNumber: The sequence number for the buffer. MUST be set to 0x00000001 in a response to an [IDL_DRSGetNT4ChangeLog](#) request with pmsgIn.V1.pRestart = NULL. The value of pmsgOut.V1.pRestart in any [IDL_DRSGetNT4ChangeLog](#) response MUST encapsulate SequenceNumber. In a response to an [IDL_DRSGetNT4ChangeLog](#) request with pmsgIn.V1.pRestart ≠ NULL, SequenceNumber is the value encapsulated in pmsgIn.V1.pRestart, plus one.

Flags: MUST be 0x00000000.

ChangeLogEntries: The pointer to an array of [CHANGELOG_ENTRY](#).

5.21 CHANGELOG_ENTRY

CHANGELOG_ENTRY is a concrete type that is defined in [\[MS-NRPC\]](#) section 3.5.4.5.4, with more information in [\[MS-ADTS\]](#) section 3.1.1.7.1.2. The abstract variable [dc.pdcChangeLog](#) is a sequence of CHANGELOG_ENTRY. The following two fields within this type are used in specifying [IDL_DRSGetNT4ChangeLog](#) server behavior:

ChangeLogEntrySize: A [DWORD](#) containing the size, in bytes, of the CHANGELOG_ENTRY structure.

SerialNumber: A [LARGE_INTEGER](#) containing the serial number of the update represented in this CHANGELOG_ENTRY.

5.22 CheckGroupMembership

```
procedure CheckGroupMembership(  
    token: ClientAuthorizationInfo,  
    groupSid: SID): boolean
```

The CheckGroupMembership procedure returns true only if the user represented by token is a member of the group whose SID is groupSid. For more information, see [\[MS-DTYP\]](#) section 2.5.2.

5.23 ClientAuthorizationInfo

ClientAuthorizationInfo is an abstract type that represents a client's security context that contains authorization information for a client.

5.24 ClientExtensions

```
procedure ClientExtensions(hDrs: DRS_HANDLE): DRS_EXTENSIONS_INT
```

The ClientExtensions server method gets the client extensions presented in the [IDL_DRSBind](#) call that created hDrs. Any fields not specified by the client in the pextClient parameter to **IDL_DRSBind** (such that pextClient^.cb is less than the offset of the end of the field of [DRS_EXTENSIONS_INT](#)) are set to 0.

5.25 ConcretePTFromAbstractPT

```
procedure ConcretePTFromAbstractPT(  
  prefixTable: PrefixTable): SCHEMA_PREFIX_TABLE
```

Informative summary of behavior: The ConcretePTFromAbstractPT procedure translates abstract [PrefixTable](#) to a [SCHEMA_PREFIX_TABLE](#) structure.

```
prefixCount: ULONG  
concretePrefixTable: SCHEMA_PREFIX_TABLE  
schemaSignature: sequence of BYTE  
prefixCount := prefixTable.length  
concretePrefixTable.PrefixCount := prefixCount  
for i := 0 to (prefixTable.length - 1)  
  concretePrefixTable.pPrefixTableEntry[i].prefix :=  
    prefixTable[i].prefixString  
  concretePrefixTable.pPrefixTableEntry[i].ndx :=  
    prefixTable[i].prefixIndex  
endfor  
return concretePrefixTable
```

5.26 ConfigNC

```
procedure ConfigNC(): DSName
```

The ConfigNC procedure returns the dsname of [dc.configNC](#).

5.27 dc, DC

DC and dc are a global variable that represents the state of a DC, as defined in [\[MS-ADTS\]](#) section 3.1.1.1.9, and the type of that variable. That definition is repeated here for convenience:

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
pdchangeLog: [PdcChangeLog](#)
nt4ReplicationState: [NT4ReplicationState](#)
ldapConnections: [LDAPConnections](#)
replicationQueue: [ReplicationQueue](#)
kccFailedConnections: [KCCFailedConnections](#)
kccFailedLinks: [KCCFailedLinks](#)
rpcClientContexts: [RPCClientContexts](#)
rpcOutgoingContexts: [RPCOutgoingContexts](#)

]

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

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

dc: DC

5.28 DefaultNC

```
procedure DefaultNC(): DSName
```

The DefaultNC procedure returns the dsname of the [dc](#).defaultNC.

5.29 DefaultNCofDC

```
procedure DefaultNCofDC(dcName: uncodestring): DSName
```

The DefaultNCofDC procedure returns the dsname of the default NC of the DC named by *dcName*, where *dcName* is the DNS name or the NetBIOS name of the DC.

5.30 DescendantObject

```
procedure DescendantObject(
    ancestor: DSName, rdns: uncodestring): DSName
```

The `DescendantObject` procedure constructs a DN string by concatenating `rdns` and `ancestor.dn`, and then verifies the existence of the descendant object. It returns the [DSName](#) if the descendant exists, and null otherwise.

5.31 DN

DN is an abstract type that is a *unicodestring* (for more information, see section [3.4.3](#)) that contains a DN of the form specified in [\[RFC2253\]](#).

5.32 DNBinary

DNBinary is an abstract type that represents the concrete type [SYNTAX_DISTNAME_BINARY](#). It consists of the following tuple:

type DNBinary = [dn: [DSName](#), binary: sequence of [BYTE](#)]

5.33 DomainNameFromNT4AccountName

```
procedure DomainNameFromNT4AccountName(  
    nt4AccountName: uncodestring): uncodestring
```

If `nt4AccountName` is a name in Windows NT 4.0 account name format, that is, two components separated by a backslash (for example, "DOMAIN\username"), then the `DomainNameFromNT4AccountName` procedure returns the first component (the domain name, or "DOMAIN" in this example). If the `nt4AccountName` is not in this format, null is returned.

5.34 DRS_EXTENSIONS

The **DRS_EXTENSIONS** structure defines a concrete type for capabilities information used in version negotiation.

```
typedef struct {  
    [range(1,10000)] DWORD cb;  
    [size_is(cb)] BYTE rgb[];  
} DRS_EXTENSIONS;
```

cb: The size, in bytes, of the `rgb` array.

rgb: To RPC this field is a string of **cb** bytes. It is interpreted by the client and the server as the first **cb** bytes of a [DRS_EXTENSIONS_INT](#) structure that follow the **cb** field of that structure. The fields of the `DRS_EXTENSIONS_INT` structure are in little-endian byte order. Since both **DRS_EXTENSIONS** and `DRS_EXTENSIONS_INT` begin with a **DWORD cb**, a field in `DRS_EXTENSIONS_INT` is at the same offset in **DRS_EXTENSIONS** as it is in `DRS_EXTENSIONS_INT`.

5.35 DRS_EXTENSIONS_INT

The `DRS_EXTENSIONS_INT` structure is a concrete type for structured capabilities information used in version negotiation.

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
cb																															
dwFlags																															
SiteObjGuid																															
...																															
...																															
...																															
Pid																															
dwReplEpoch																															
dwFlagsExt																															

cb: The count of bytes in the fields **dwFlags** through **dwReplEpoch**, inclusive. This field allows extended versions of this structure to carry more information in future product versions.

dwFlags: The **dwFlags** field contains individual bit flags that describe the capabilities of the DC that produced the DRS_EXTENSIONS_INT structure. [<26>](#26)

The following table lists the bit flags.

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
B	A	R	M	D	D	U	A	K	A	L	D	I	C	G	S	D	T	S	P	G	G	G	A	G	G	G	W	D	R	R	R
A	S	M	V	F	C	O	E	E	E	V	C	N	B	R	E	C	M	H	B	C	M	C	N	C	R	R	B	F	1	2	3
S									2	R	2	R		I		F			3	5	2	6	C	8	5	6	3	2			

BAS (DRS_EXT_BASE): Unused. MUST be 1 and ignored.

AS (DRS_EXT_ASYNCREPL): If present, signifies that the DC supports [DRS_MSG_REPADD_V2](#) and DRS_MSG_GETCHGREQ_V2.

RM (DRS_EXT_REMOVEAPI): If present, signifies that the DC supports [IDL DRSRemoveDsServer](#) and [IDL DRSRemoveDsDomain](#).

MV (DRS_EXT_MOVEREQ_V2): If present, signifies that the DC supports [DRS_MSG_MOVEREQ_V2](#).

DF (DRS_EXT_GETCHG_DEFLATE): If present, signifies that the DC supports [DRS_MSG_GETCHGREPLY_V2](#).

DC (DRS_EXT_DCINFO_V1): If present, signifies that the DC supports [IDL_DRSDomainControllerInfo](#).

UO (DRS_EXT_RESTORE_USN_OPTIMIZATION): Unused, MUST be 1 and ignored.

AE (DRS_EXT_ADDENTRY): If present, signifies that the DC supports [IDL_DRSAddEntry](#).

KE (DRS_EXT_KCC_EXECUTE): If present, signifies that the DC supports [IDL_DRSExecuteKCC](#).

AE2 (DRS_EXT_ADDENTRY_V2): If present, signifies that the DC supports [DRS_MSG_ADDENTRYREQ_V2](#).

LVR (DRS_EXT_LINKED_VALUE_REPLICATION): If present, signifies that the DC supports **link value** replication, and this support is enabled.

DC2 (DRS_EXT_DCINFO_V2): If present, signifies that the DC supports [DRS_MSG_DCINFOREPLY_V2](#).

INR (DRS_EXT_INSTANCE_TYPE_NOT_REQ_ON_MOD): Unused. MUST be 1 and ignored.

CB (DRS_EXT_CRYPTO_BIND): A client-only flag. If present, it indicates that the security provider used for the connection supports session keys through RPC (example, Kerberos connections with mutual authentication enable RPC to expose session keys, but NTLM connections do not enable RPC to expose session keys).

GRI (DRS_EXT_GET_REPL_INFO): If present, signifies that the DC supports [IDL_DRSGetReplInfo](#).

SE (DRS_EXT_STRONG_ENCRYPTION): If present, signifies that the DC supports additional 128-bit encryption for passwords over the wire. DCs MUST NOT replicate passwords from other DCs that do not support this extension.

DCF (DRS_EXT_DCINFO_VFFFFFFFF): If present, signifies that the DC supports [DRS_MSG_DCINFOREPLY_VFFFFFFFF](#).

TM (DRS_EXT_TRANSITIVE_MEMBERSHIP): If present, signifies that the DC supports [IDL_DRSGetMemberships](#).

SH (DRS_EXT_ADD_SID_HISTORY): If present, signifies that the DC supports [IDL_DRSAddSidHistory](#).

PB3 (DRS_EXT_POST_BETA3): Unused. MUST be 1 and ignored.

GC5 (DRS_EXT_GETCHGREQ_V5): If present, signifies that the DC supports [DRS_MSG_GETCHGREQ_V5](#).

GM2 (DRS_EXT_GETMEMBERSHIPS2): If present, signifies that the DC supports [IDL_DRSGetMemberships2](#).

GC6 (DRS_EXT_GETCHGREQ_V6): If present, signifies that the DC supports [DRS_MSG_GETCHGREQ_V6](#).

ANC (DRS_EXT_NONDOMAIN_NCS): If present, signifies that the DC supports application NCs.

GC8 (DRS_EXT_GETCHGREQ_V8): If present, signifies that the DC supports [DRS_MSG_GETCHGREQ_V8](#).

GR5 (DRS_EXT_GETCHGREPLY_V5): If present, signifies that the DC supports DRS_MSG_GETCHGREPLY_V5.

GR6 (DRS_EXT_GETCHGREPLY_V6): If present, signifies that the DC supports [DRS_MSG_GETCHGREPLY_V6](#).

WB3 (DRS_EXT_WHISTLER_BETA3): If present, signifies that the DC supports [DRS_MSG_ADDENTRYREPLY_V3](#), [DRS_MSG_REPVERIFYOBJ](#), and [DRS_MSG_GETCHGREPLY_V7](#).

DF2 (DRS_EXT_W2K3_DEFLATE): If present, signifies that the DC supports the W2K3 AD deflation library.

R1 (DRS_EXT_RESERVED_FOR_WIN2K_OR_DOTNET_PART1): Unused. MUST be 0 and ignored.

R2 (DRS_EXT_RESERVED_FOR_WIN2K_OR_DOTNET_PART2): Unused. MUST be 0 and ignored.

R3 (DRS_EXT_RESERVED_FOR_WIN2K_OR_DOTNET_PART3): Unused. MUST be 0 and ignored.

SiteObjGuid: A GUID. The [objectGUID](#) of the [site](#) object of which the DC's DSA object is a descendant. For non-DC client callers, this field SHOULD be set to zero.

Pid: A 32-bit signed integer value that specifies the process identifier of the client. This is for informational and debugging purposes only. The assignment of this field is implementation-specific. [<27>](#)

dwReplEpoch: A 32-bit, unsigned integer value that specifies the replication epoch.

dwFlagsExt: The extension of the **dwFlags** field that contains individual bit flags that describe the capabilities of the DC that produced the DRS_EXTENSIONS_INT structure. For non-DC client callers, no bits SHOULD be set. The following table lists the bit flags.

	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
D A	L H	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

DA (DRS_EXT_ADAM): If present, signifies that the DC supports DRS_MSG_REPSYNC_V2, DRS_MSG_UPDREFS_V2, [DRS_MSG_INIT_DEMOTIONREQ_V1](#), [DRS_MSG_REPLICA_DEMOTIONREQ_V1](#), and [DRS_MSG_FINISH_DEMOTIONREQ_V1](#).

LH (DRS_EXT_LH_BETA2): If present, signifies that the DC supports the DRS_SPECIAL_SECRET_PROCESSING and DRS_GET_ALL_GROUP_MEMBERSHIP flags as well as **InfoLevel** 3 in [DRS_MSG_DCINFOREQ_V1](#).

5.36 DRS_HANDLE

DRS_HANDLE is a concrete type for an RPC context handle (as specified in [C706]) for use in calls to methods in the [drsuapi RPC interface](#).

This type is declared as follows:

```
typedef [context_handle] void* DRS_HANDLE;
```

For the specification of [IDL DRSBind](#), see section 4.1.3.

Methods in the [dsaop RPC interface](#) do not use context handles.

5.37 DRS_OPTIONS

DRS_OPTIONS is a concrete type for a set of options sent to and received from various [drsuapi](#) methods.

This type is declared as follows:

```
typedef unsigned long DRS_OPTIONS;
```

Five elements of the set are interpreted differently by different methods; such elements have multiple symbolic names.

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
A S	X	A R	A L / D R	W R	I S	P S	M R	A S R / I E	T S	C O	G A	G S / L O	X	S N / R F	F S / N S	X	F S P	X	X	X	X	X	S S	X	S F	D A S	D P S	U C	N N	S P	G P

X: Unused. MUST be zero and ignored.

AS (DRS_ASYNC_OP): Performs the operation asynchronously.

AR (DRS_ADD_REF): Registers a client DC for notifications of updates to the NC replica.

ALL (DRS_SYNC_ALL): Replicates from all server DCs.

DR (DRS_DEL_REF): Deregisters a client DC from notifications of updates to the NC replica.

WR (DRS_WRIT_REP): Replicates a writable replica, not a read-only partial replica or read-only full replica.

IS (DRS_INIT_SYNC): Performs replication at startup.

PS (DRS_PER_SYNC): Performs replication periodically.

MR (DRS_MAIL_REP): Performs replication using SMTP as a transport.

ASR (DRS_ASYNC_REP): Populates the NC replica asynchronously.

IE (DRS_IGNORE_ERROR): Ignores errors.

TS (DRS_TWOWAY_SYNC): Informs the server DC to replicate from the client DC.

CO (DRS_CRITICAL_ONLY): Replicates only system-critical objects.

GA (DRS_GET_ANC): Includes updates to ancestor objects before updates to their descendants.

GS (DRS_GET_NC_SIZE): Gets the approximate size of the server NC replica.

LO (DRS_LOCAL_ONLY): Performs the operation locally without contacting any other DC.

SN (DRS_SYNC_BYNAME): Chooses the source server by network name.

RF (DRS_REF_OK): Allows the NC replica to be removed even if other DCs use this DC as a replication server DC.

FS (DRS_FULL_SYNC_NOW): Replicates all updates in the replication cycle, even those that would normally be filtered.

NS (DRS_NO_SOURCE): The NC replica has no server DCs.

FSP (DRS_FULL_SYNC_PACKET): Replicates all updates in the replication request, even those that would normally be filtered.

SS (DRS_SPECIAL_SECRET_PROCESSING): Does not replicate attribute values of attributes that contain secret data.

SF (DRS_SYNC_FORCED): Forces replication, even if the replication system is otherwise disabled.

DAS (DRS_DISABLE_AUTO_SYNC): Disables replication induced by update notifications.

DPS (DRS_DISABLE_PERIODIC_SYNC): Disables periodic replication.

UC (DRS_USE_COMPRESSION): Compresses response messages.

NN (DRS_NEVER_NOTIFY): Do not send update notifications.

SP (DRS_SYNC_PAS): Expands the partial attribute set of the partial replica.

GP (DRS_GET_ALL_GROUP_MEMBERSHIP): Replicates all kinds of group membership. If this flag is not present nonuniversal group membership are be replicated.

5.38 DRS_SecBuffer

DRS_SecBuffer is a concrete type for a buffer that contains authentication data.

```
typedef struct {
    [range(0,10000)] unsigned long cbBuffer;
    unsigned long BufferType;
    [size_is(cbBuffer)] BYTE* pvBuffer;
} DRS_SecBuffer;
```

cbBuffer: The size, in bytes, of the pvBuffer array.

BufferType: A bit field that contains the following 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
TYP			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	R
																															O

X: Unused. MUST be zero and ignored.

TYP: May be one of the following values:

Value	Meaning
SECBUFFER_EMPTY 0x00000000	A placeholder in the buffer array. The caller can supply several such entries in the array, and the security package can return data in them.
SECBUFFER_DATA 0x00000001	Used for common data. The security package can read this data and write it, for example, to encrypt some or all of it.
SECBUFFER_TOKEN 0x00000002	This buffer is used to indicate the security token portion of the message. This is read-only for input parameters or read/write for output parameters.
SECBUFFER_PKG_PARAMS 0x00000003	These are transport-to-package-specific parameters. For example, the Netware redirector may supply the server object identifier, while DCERPC can supply an association UUID , and so on.
SECBUFFER_MISSING 0x00000004	The security package uses this value to indicate the number of missing bytes in a particular message. The pvBuffer member is ignored in this type.
SECBUFFER_EXTRA 0x00000005	The security package uses this value to indicate the number of extra or unprocessed bytes in a message.
SECBUFFER_STREAM_TRAILER 0x00000006	Indicates a protocol-specific trailer for a particular record. This is not usually of interest to callers.
SECBUFFER_STREAM_HEADER 0x00000007	Indicates a protocol-specific header for a particular record. This is not usually of interest to callers.

RO (SECBUFFER_READONLY): The buffer is read-only. This flag is intended for sending header data to the security package for checksumming. The package can read this buffer but cannot modify it.

pvBuffer: Authentication data.

5.39 DRS_SecBufferDesc

DRS_SecBufferDesc is a Generic Security Services (GSS) Kerberos authentication token (as specified in [\[RFC1964\]](#)).

```
typedef struct {
    unsigned long ulVersion;
    [range(0,10000)] unsigned long cBuffers;
    [size_is(cBuffers)] DRS_SecBuffer* Buffers;
} DRS_SecBufferDesc;
```

ulVersion: MUST be 0.

cBuffers: The count of items in the Buffers array.

Buffers: Buffers that contain authentication data.

5.40 DRS_SPN_CLASS

A *unicodestring* constant (as specified in section [3.4.3](#)) that is used as the service class in the SPN for a DC. It has the value "E3514235-4B06-11D1-AB04-00C04FC2DCD2".

5.41 DS_REPL_OP_TYPE

DS_REPL_OP_TYPE is a concrete type for the replication operation type.

```
typedef enum
{
    DS_REPL_OP_TYPE_SYNC = 0x00000000,
    DS_REPL_OP_TYPE_ADD = 0x00000001,
    DS_REPL_OP_TYPE_DELETE = 0x00000002,
    DS_REPL_OP_TYPE_MODIFY = 0x00000003,
    DS_REPL_OP_TYPE_UPDATE_REFS = 0x00000004
} DS_REPL_OP_TYPE;
```

DS_REPL_OP_TYPE_SYNC: Syncs the NC replica from the server DC.

DS_REPL_OP_TYPE_ADD: Adds the NC replica to the server DC.

DS_REPL_OP_TYPE_DELETE: Removes the NC replica from the server DC.

DS_REPL_OP_TYPE_MODIFY: Modifies the NC replica of the server DC.

DS_REPL_OP_TYPE_UPDATE_REFS: Updates the NC replica of the client DC.

5.42 DSAObj

```
procedure DSAObj(): DSName
```

DSAObj returns the dsname of the DC's [nTDSDSA](#) object.

```
return select one o from children ConfigNC()
where o!objectGUID = dc.serverGUID
```

5.43 DSNName

DSName is an abstract type for representing a dsname. It corresponds to the concrete representation [DSNAME](#). It consists of a tuple that identifies an object in the directory. This tuple is discussed in [\[MS-ADTS\]](#) section 3.1.1.1.5. For this document, the fields of the tuple are defined as follows.

```
type DSNName = [dn: NameString , guid: GUID, sid: Sid]
```

The *dn* field corresponds to the NameString field of the [DSNAME](#) and contains the DN of the object. The *guid* field corresponds to the GUID field of the [DSNAME](#) and contains the value of the object's [objectGUID](#) attribute. The *sid* field corresponds to the Sid field of the [DSNAME](#). If the object possesses an [objectSid](#) attribute, it contains the value of the object's [objectSid](#) attribute. If the object does not possess an [objectSid](#) attribute, the field is null.

5.44 DSNAME

DSNAME is a concrete type for representing a [DSName](#) and identifying a directory object using the values of one or more of its LDAP attributes: [objectGUID](#), [objectSid](#), or [distinguishedName](#).

```
typedef struct {
    unsigned long structLen;
    unsigned long SidLen;
    GUID Guid;
    NT4SID Sid;
    unsigned long NameLen;
    [size_is(NameLen + 1)] WCHAR StringName[];
} DSNAME;
```

structLen: The length, in bytes, of the entire data structure.

SidLen: The number of bytes in the Sid field used to represent the object's [objectSid](#) attribute value. Zero indicates that the **DSNAME** does not identify the [objectSid](#) value of the directory object.

Guid: The value of the object's [objectGUID](#) attribute specified as a GUID structure, which is defined in [\[MS-DTYP\]](#) section **2.3.2**. Zero values for all fields in the GUID structure indicate that the **DSNAME** does not identify the [objectGUID](#) value of the directory object.

Sid: The value of the object's [objectSid](#) attribute, its security identifier (see [\[MS-SECO\]](#) section 2.1.2), specified as a **SID** structure, which is defined in [\[MS-DTYP\]](#) section **2.4.2**. The size of this field is exactly 28 bytes, regardless of the value of SidLen, which specifies how many bytes in this field are used. Note that this is smaller than the theoretical size limit of a SID, which is 68 bytes. While Windows publishes a general SID format, Windows never uses that format in its full generality. The 28 bytes is sufficient for a Windows SID.

StringName: The null-terminated Unicode value of the object's **distinguishedName** attribute, as specified in [\[MS-ADTS\]](#) section 3.1.1.1.4. This field always contains at least one character: the null terminator. Each Unicode value is encoded as 2 bytes. The byte ordering is little-endian.

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
structLen																															
SidLen																															
Guid.Data1																															
Guid.Data2																Guid.Data3															
Guid.Data4...																															
...Guid.Data4																															
Sid...																															
...Sid...																															
...Sid...																															
...Sid...																															
...Sid...																															
...Sid...																															
...Sid																															
NameLen																															
StringName (Variable Length) ...																															

[MS-DRSR] – v20080207
Directory Replication Service (DRS) Remote Protocol Specification
Copyright © 2008 Microsoft Corporation.
Release: Thursday, February 7, 2008

5.44.1 DSNAME Equality

When comparing **DSNAME** elements for equality, an implementation must be aware that multiple attributes may be specified. **DSNAME** values *x* and *y* are equal only if one of the following conditions holds:

- *x*.Guid is not zeros, *y*.Guid is not zeros, and *x*.Guid = *y*.Guid.
- All of the following are true:
 - *x*.Guid is zeros or *y*.Guid is zeros.
 - *x*.StringLen ≠ 0
 - The number of RDNs in *x* is the same as in *y*.
 - For each RDN *x_i* in *x* and RDN *y_i* in *y* (see [RFC2253]):
 - AttributeType of *x_i* = AttributeType of *y_i*
 - AttributeValue of *x_i* = AttributeValue of *y_i*, without regard to case differences, Hiragana and Katakana character differences, and nonspacing characters.
- All of the following are true:
 - *x*.Guid is zeros.
 - *y*.Guid is zeros.
 - *x*.StringLen = 0
 - *y*.StringLen = 0
 - *x*.SidLen ≠ 0
 - *x*.SidLen = *y*.SidLen
 - *x*.Sid and *y*.Sid contain identical values in the first *x*.SidLen array items.

5.45 DSTIME

DSTIME is a concrete type for time expressed as the number of seconds since January 1, 1601, 12:00:00am.

This type is declared as follows:

```
typedef LONGLONG DSTIME;
```

The following table shows an alternative representation of this 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
cSeconds...																															
...cSeconds																															

Note Byte ordering is little-endian.

5.46 DWORD

DWORD is a concrete type for a 32-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.

5.47 ENTINF

ENTINF is a concrete type for the identity and attributes (some or all) of a given object.

```
typedef struct {
    DSNAME* pName;
    unsigned long ulFlags;
    ATTRBLOCK AttrBlock;
} ENTINF;
```

pName: The identity of the object.

ulFlags: A flags field that supports the following flags.

											1										2												3	
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			
M	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	R	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
	O															M																		

X: Unused. MUST be zero and ignored.

M (ENTINF_FROM_MASTER): Retrieved from a full replica.

DO (ENTINF_DYNAMIC_OBJECT): A dynamic object.

RM (ENTINF_REMOTE_MODIFY): A remote modify request to [IDL DRSAddEntry](#) (see section [4.1.1.3](#)).

AttrBlock: Some of all of the attributes for this object, as determined by the particular method. See section [1.3.3](#) for an overview of methods using type **ENTINF**.

5.48 ENTINF_GetValue

```
procedure ENTINF_GetValue (
    entInf: ENTINF,
```

```

attribute: ATTRTYP,
prefixTable: PrefixTable): ATTRVAL

```

Informative summary of behavior: The ENTINF_GetValue procedure scans an [ENTINF](#) structure and returns the first [ATTRVAL](#) structure for the requested attribute. The attribute parameter is based on [dc.prefixTable](#), while the attributes within entInf are based on the prefixTable parameter.

```

attrType: ATTRTYP
oid : OID
oid := OidFromAttid(dc.prefixTable, attribute)
attrType := MakeAttid(prefixTable, oid)
for each i in [0 .. entInf.AttrBlock.attrCount -1] do
    if (entInf.AttrBlock.pAttr[i].attrTyp = attrType) and
        (entInf.AttrBlock.pAttr[i].AttrVal.valCount > 0) then
        return entInf.AttrBlock.pAttr[i].AttrVal.pAVal[0]
    endif
endfor
return null

```

5.49 ENTINF_SetValue

```

procedure ENTINF_SetValue (
    var entInf: ENTINF,
    attribute: ATTRTYP,
    attrVal: ATTRVAL,
    prefixTable: PrefixTable)

```

ENTINF_SetValue updates an attribute value within the [ENTINF](#). If attrVal is null, then the attribute is removed from the list (if it exists). If the value is non-null, then the attribute value is updated or added to the list (when a value is not already present). The attribute and attrVal parameters are based on [dc.prefixTable](#), while the attributes within entInf are based on the prefixTable parameter.

5.50 ENTINF_EnumerateAttributes

```

procedure ENTINF_EnumerateAttributes(
    e: ENTINF,
    prefixTable: PrefixTable): set of ATTRTYP

```

ENTINF_EnumerateAttributes returns the list of attributes (based on [dc.prefixTable](#)) that are present in the [ENTINF](#) e. Attributes within e are based on the prefixTable parameter.

5.51 ENTINFLIST

ENTINFLIST is a concrete type for a list of [ENTINF](#) entries.

```

typedef struct ENTINFLIST {
    struct ENTINFLIST* pNextEntInf;
    ENTINF Entinf;
} ENTINFLIST;

```

pNextEntInf: The next **ENTINFLIST** in the sequence, or NULL.

Entinf: An [ENTINF](#) entry.

5.52 Expunge

```
procedure Expunge(obj: DSName)
```

Expunge physically removes an object whose [DSName](#) is obj from the directory, without enforcing referential integrity constraints. The object is immediately removed without undergoing conversion to a tombstone.

5.53 FILETIME

FILETIME is a concrete type for a time, as specified in [\[MS-DTYP\]](#) section 2.3.1.

5.54 FilteredGCPAS

```
procedure FilteredGCPAS() : PARTIAL_ATTR_VECTOR_V1_EXT^
```

Informative summary of behavior: The FilteredGCPAS procedure returns a reference to an instance of structure [PARTIAL_ATTR_VECTOR_V1_EXT](#) that contains the list of attributes that may be present, based on the schema, on a filtered GC replica.

```
attrSetSeq: sequence of DSName
filteredAttributeSet: sequence of ATTRTYP
pPartialAttrVector: PARTIAL_ATTR_VECTOR_V1_EXT^
attrId: ATTRTYP
i, j:int

attrSetSeq := select o from subtree SchemaNC() where
    (attributeSchema in o!objectClass) and
    (o!isMemberOfPartialAttributeSet = true)

filteredAttributeSet := GetFilteredAttributeSet()

pPartialAttrVector = new PARTIAL_ATTR_VECTOR_V1_EXT sized to hold
    (attrSetSeq.length - filteredAttributeSet.length) entries in
    its rgPartialAttr field
pPartialAttrVector^.dwVersion := 1
-
j := 0
for i := 0 to attrSetSeq.length-1
    attrId = AttrtypFromSchemaObj(attrSetSeq[i]);
    if (not attrId in filteredAttributeSet) then
        /* attribute is not in the filtered list */
        partialAttrVector^.rgPartialAttr[j] := attrId
        j := j + 1
    endif
endfor
pPartialAttrVector^.cattrs := j

return pPartialAttrVector^
```

5.55 FilteredPAS

```
procedure FilteredPAS() : PARTIAL_ATTR_VECTOR_V1_EXT
```

Informative summary of behavior: The FilteredPAS procedure returns a reference to an instance of structure [PARTIAL_ATTR_VECTOR_V1_EXT](#) that contains the list of attributes that may be present, based on the schema, on a filtered NC replica.

```
attrSetSeq: sequence of DSName
filteredAttributeSet: sequence of ATTRTYP
pPartialAttrVector: PARTIAL_ATTR_VECTOR_V1_EXT^
attrId: ATTRTYP
i, j: int

attrSetSeq := select o from subtree SchemaNC() where
    (attributeSchema in o!objectClass) and
    (o!systemFlags &
        {FLAG_ATTR_NOT_REPLICATED,
         FLAG_ATTR_IS_CONSTRUCTED} = null)

filteredAttributeSet := GetFilteredAttributeSet()

pPartialAttrVector = new PARTIAL_ATTR_VECTOR_V1_EXT sized to hold
    (attrSetSeq.length - filteredAttributeSet.length) entries in
    its rgPartialAttr field
pPartialAttrVector^.dwVersion := 1
for i := 0 to attrSetSeq.length-1
    attrId = AttrtypFromSchemaObj(attrSetSeq[i]);
    if (not attrId in filteredAttributeSet = null) then
        /* attribute is not in the filtered list */
        pPartialAttrVector^.rgPartialAttr[j] := attrId
        j := j + 1
    endif
endfor
pPartialAttrVector^.cAttrs := j

return pPartialAttrVector^
```

5.56 FindChar

```
procedure FindChar(
    s: unicodestring, start: integer, c: UCHAR): integer
```

Informative summary of behavior: The FindChar procedure returns the zero-based index of the first occurrence of c in the portion of s between the start and the end of s.

If s = null, start < 0 or start > s.length-1, this procedure returns -1. Otherwise, let s be represented as the sequence of characters {s[0], ... s[s.length - 1]}. Let i be such that i >= start, i <= s.length - 1, s[i] = c, and s[start] ≠ c, ..., s[i-1] ≠ c. If such an i exists, this procedure returns i. Otherwise, this procedure returns -1.

5.57 FindCharRev

```
procedure FindCharRev(  
    s: unicodestring,  
    start: integer,  
    c: UCHAR): integer
```

Informative summary of behavior: The FindCharRev procedure returns the zero-based index of the last occurrence of c in the portion of s between the start and the end of s.

If s = null, start < 0 or start > s.length-1, this procedure returns -1. Otherwise, let s be represented as the sequence of characters {s[0], ... s[s.length - 1]}. Let i be such that i ≥ start, i ≤ s.length - 1, s[i] = c, and s[i+1] ≠ c, ..., s[s.length - 1] ≠ c. If such an i exists, this procedure returns i. Otherwise, this procedure returns -1.

5.58 FOREST_TRUST_INFORMATION

FOREST_TRUST_INFORMATION is a concrete type for state information about trust relationships with other forests. This data 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: The version of the data structure. The only supported version of the data structure is 1.

RecordCount: The number of records present in the data structure.

Records: Variable-length records that each contain specific the type of data about the forest trust relationship.

Note Records are not necessarily 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 [5.58.1](#).

Note All fields have little-endian byte ordering.

5.58.1 Record

Each 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
RecordLen																															
Flags																															
Timestamp																															
...																															
RecordType										ForestTrustData (variable)																					
...																															

RecordLen: The length, in bytes, of the entire record.

Flags: The individual bit flags that control how the forest trust information in this record can be used.

If RecordType = 0 or 1, the **Flags** field 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
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	X	X	X	X	X
D	D	D																													
N	A	C																													

X: Unused. Must be zero and ignored.

TDN (LSA_TLN_DISABLED_NEW): The entry is not yet enabled.

TDA (LSA_TLN_DISABLED_ADMIN): The entry is disabled by the administrator.

TDC (LSA_TLN_DISABLED_CONFLICT): The entry is disabled due to a conflict with another trusted domain.

If RecordType = 2, the **Flags** field 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
S	S	N	N	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																												
A	C	A	C																												

X: Unused. MUST be zero and ignored.

SDA (LSA_SID_DISABLED_ADMIN): The entry is disabled for SID-based matches by the administrator.

SDC (LSA_SID_DISABLED_CONFLICT): The entry is disabled due to a SID conflict with another trusted domain.

NDA (LSA_NB_DISABLED_ADMIN): The entry is disabled for NetBIOS name-based matches by the administrator.

NDC (LSA_NB_DISABLED_CONFLICT): The entry is disabled due to a 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:** A 64-bit time-stamp value that indicates when this entry was created.
- RecordType:** An 8-bit value that specifies the type of record contained in this specific entry. The allowed values are specified in section [5.59](#).
- ForestTrustData:** A variable length, type-specific record, depending on the RecordType value, that contains the 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, described as follows.

- 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	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
NameLen																															
Name (variable length)...																															

NameLen: The length, in bytes, of the Name field.

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 that none of the following data shown is necessarily aligned to 32-bit boundaries.

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
SidLen																															
Sid (variable length)...																															
DnsNameLen																															
DnsName (variable length)...																															
NetbiosNameLen																															
NetbiosName (variable length)...																															

SidLen: The length, in bytes, of the Sid field.

Sid: The SID of a domain in the trusted forest, specified as a [SID](#) structure, which is defined in [\[MS-DTYP\]](#) section 2.3.

DnsNameLen: The length, in bytes, of the DnsName field.

DnsName: The DNS name of a domain in the trusted forest, in UTF-8 format.

NetbiosNameLen: The length, in bytes, of the NetbiosName field.

NetbiosName: The NetBIOS name of a domain in the trusted forest, in UTF-8 format.

- If the RecordType is not one of the preceding values, 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: The length, in bytes, of the BinaryData field.

BinaryData: The trusted forest data.

5.58.2 Determining If a Name Is In a Trusted Forest

This section describes procedures that use the forest trust information contained in the [msDS-TrustForestTrustInfo](#) attribute to determine if a given domain is in a trusted forest.

The procedures described in this subsection use the following data structures.

```
struct {
    ULONG RecordCount;
    PX_FOREST_TRUST_RECORD *Entries;
} X_FOREST_TRUST_INFORMATION;

struct {
    ULONG Flags;
    FOREST_TRUST_RECORD_TYPE ForestTrustType;
    LARGE_INTEGER Time;
    union {
        LPWSTR TopLevelName;
        X_FOREST_TRUST_DOMAIN_INFO DomainInfo;
        X_FOREST_TRUST_BINARY_DATA Data;
    } ForestTrustData;
} X_FOREST_TRUST_RECORD, *PX_FOREST_TRUST_RECORD;

struct {
    SID *Sid;
    LPWSTR DnsName;
    LPWSTR NetbiosName;
} X_FOREST_TRUST_DOMAIN_INFO;

struct {
    ULONG Length;
    BYTE *Buffer;
} X_FOREST_TRUST_BINARY_DATA;
```

The X_FOREST_TRUST_INFORMATION structure previously defined is used by the procedure to determine if a given domain is in a trusted forest. To unmarshal the content of the [msDS-TrustForestTrustInfo](#) attribute into this structure, the UnmarsallForestTrustInfo procedure described below can be used.

```
procedure ExtractString(
    buffer: sequence of BYTE,
    index: DWORD, size: DWORD): unicodestring;
```

The sequence [index .. index + size] of bytes in the buffer is interpreted as a UTF-8 string, and a corresponding *unicodestring* (as described in section [3.4.3](#)) is returned.

```
procedure ExtractSid(
    buffer: sequence of BYTE,
    index: DWORD, size: DWORD): SID;
```

The sequence [index .. index + size] of bytes in a buffer is converted into a [SID](#) structure and returned.

```

procedure ExtractBinary(
    buffer: sequence of BYTE,
    index: DWORD, size: DWORD): sequence of BYTE;

```

The sequence [index .. index + size] of bytes in a buffer is returned.

```

procedure UnmarshalForestTrustInfo
    (inputBuffer: sequence of BYTE,
    var forestTrustInfo: X_FOREST_TRUST_INFORMATION): boolean

```

Informative summary of behavior: The UnmarshalForestTrustInfo procedure unmarshals the byte stream inputBuffer, which holds the content of a [msDS-TrustForestTrustInfo](#) attribute that contains forest trust information, as described in [FOREST TRUST INFORMATION](#), into the forestTrustInfo structure.

```

index: DWORD
pdwVersion: ADDRESS OF DWORD
pdwRecordCount: ADDRESS OF DWORD
i: DWORD
pwdRecordLength: ADDRESS OF DWORD
pTrustRecord: ADDRESS OF X_FOREST_TRUST_RECORD
pulTime: ADDRESS OF ULONGLONG
pType: ADDRESS OF BYTE
pSid: ADDRESS OF SID
pString: ADDRESS OF unicodestring
pdwSize: ADDRESS OF DWORD

index := 0

pdwVersion := ADR(inputBuffer[index])
if pdwVersion^ ≠ 1 then
    return false
endif

index := index + 4

pdwRecordCount := ADR(inputBuffer[index])
forestTrustInfo.RecordCount := pdwRecordCount^
index := index + 4

/* Extract each record */
for i:= 0 to pdwRecordCount^

    /* First 4 bytes of the record is the length */
    pwdRecordLength := ADR(inputBuffer[index])
    index := index + 4

    pTrustRecord := forestTrustInfo.Entries[i]

    /* Next 4 bytes of the record are the flags */
    pdwFlags := ADR(inputBuffer[index])
    pTrustRecord^.Flags := pdwFlags^
    index := index + 4

    /* Next 8 bytes of the record represent the Time field */

```

```

pulTime := ADR(inputBuffer[index])
pTrustRecord^.Time := pulTime^
index := index + 8

/* Next byte represents trust type */
pType := ADR(inputBuffer[index])
pTrustRecord^.ForestTrustType := pType^
index := index + 1

if (pTrustRecord^.ForestTrustType = ForestTrustTopLevelName or
    pTrustRecord^.ForestTrustType = ForestTrustTopLevelNameEx)
    then

    /* Next 4 bytes represent the size of the top level name */
    pdwSize := ADR(inputBuffer[index])
    index := index + 4

    /* Extract the top level name; index is at the start of name */
    pTrustRecord^.TopLevelName :=
        ExtractString(inputBuffer, index, pdwSize^)
    index := index + pdwSize^
else
    if (pTrustRecord^.ForestTrustType = ForestTrustDomainInfo)
        then
            /* Next 4 bytes represent the size of the sid */
            pdwSize := ADR(inputBuffer[index])
            index := index + 4

            /* Extract the sid; index is at the start of sid */
            pTrustRecord^.DomainInfo.Sid :=
                ExtractSid(inputBuffer, index, pdwSize^)
            index := index + pdwSize^

            /* Next 4 bytes represent the size of the dns domain name */
            pdwSize := ADR(inputBuffer[index])
            index := index + 4

            /* Extract the dns domain name; index is at start of name */
            pTrustRecord^.DomainInfo.DnsName :=
                ExtractString(inputBuffer, index, pdwSize^)
            index := index + pdwSize^

            /* Next 4 bytes represent the size of the netbios
             * domain name */
            pdwSize := ADR(inputBuffer[index])
            index := index + 4

            /* Extract the netbios domain name; index is at the start
             * of name */
            pTrustRecord^.DomainInfo.NetbiosName :=
                ExtractString(inputBuffer, index, pdwSize^)
            index := index + pdwSize^
        else
            /* Next 4 bytes represent the size of the binary data */
            pdwSize := ADR(inputBuffer[index])
            pTrustRecord^.Data.Length := pdwSize^
            index := index + 4

```

```

        /* Extract the binary data; index is at the start of data */
        pTrustRecord^.Data.Buffer :=
            ExtractBinaryData(inputbuffer, index, pdwSize^)
        index := index + pdwSize^
    endif

endif

/* index is now at the beginning of the next record */
endfor

return true

```

The following procedure is used to determine if a given domain is in a trusted forest. Since it makes use of forest trust information data stored in objects in the NC replica of the forest root domain (see FOREST_TRUST_INFORMATION), this function only works on GC servers or DCs in the forest root domain.

```

procedure IsDomainNameInTrustedForest(name: unicodestring): boolean

```

Informative summary of behavior: The IsDomainNameInTrustedForest procedure determines if the domain with the name given by name is in a trusted forest. The input name may be a DNS or a NetBIOS name.

```

    if IsDomainDnsNameInTrustedForest(name) then
        return true
    endif

    if IsDomainNetbiosNameInTrustedForest(name) then
        return true
    endif

    return false

```

The IsDomainNameInTrustedForest procedure uses the following helper procedures to determine if a domain is in a trusted forest.

```

procedure ForestTrustOwnsName(o: DSName, name: unicodestring): boolean

    f: X_FOREST_TRUST_INFORMATION

    /* o is the DSName identifying the object containing the forest
     * trust information being evaluated to determine if the input
     * name is owned by the trusted forest represented by o. */

    if not UnmarshalForestTrustInfo(o!msDS-TrustForestTrustInfo, f)
        /* unable to extract forest trust information from binary form */
        return false
    endif

    /* if a suffix of the name is in the exclusion list, the
     * forest does not own this name */
    foreach e in f.Entries

```



```

        if (e.ForestTrustType = ForestTrustTopLevelNameEx and
            e.TopLevelName is a suffix for name) then
            return false
        endif
    endfor

    /* if a suffix of the name is in the inclusion list and is
     * not disabled, the forest owns this name */
    foreach e in f.Entries
        if (e.ForestTrustType = ForestTrustTopLevelName and
            LSA_FTRECORD_DISABLED_REASONS not in e.Flags and
            e.TopLevelName is a suffix for name) then
            return true
        endif
    endfor

    return false
endprocedure

procedure IsDomainDnsNameInTrustedForest(name: unicodestring)
: boolean

    tdos: set of DSName
    f: X_FOREST_TRUST_INFORMATION

    /* Get all the objects that represent trusted domains */
    tdos := select all o in Children ForestRootDomainNC() where
        trustedDomain in o!objectClass and
        o!trustAttributes & 0x00000008 ≠ 0 and
        o!msDS-TrustForestTrustInfo ≠ null

    foreach o in tdos
        if not UnmarshalForestTrustInfo(o!msDS-TrustForestTrustInfo, f)
            then
            return false
        else
            foreach e in f.Entries
                if (e.ForestTrustType = ForestTrustDomainInfo and
                    e.DomainInfo.DnsName = name and
                    LSA_SID_DISABLED_ADMIN not in e.Flags and
                    LSA_SID_DISABLED_CONFLICT not in e.Flags and
                    ForestTrustOwnsName(o, e.DomainInfo.DnsName) then
                    return true
                endif
            endfor
        endif
    endfor

    return false
endprocedure

procedure IsDomainNetbiosNameInTrustedForest
(name: unicodestring): boolean

    tdos: set of DSName
    f: X_FOREST_TRUST_INFORMATION

    /* Get all the objects that represent trusted domains */
    tdos := select all o in Children ForestRootDomainNC() where
        trustedDomain in o!objectClass and

```

```

        o!trustAttributes & 0x00000008 ≠ 0 and
        o!msDS-TrustForestTrustInfo ≠ null

foreach o in tdos
    if not UnmarshalForestTrustInfo(o!msDS-TrustForestTrustInfo, f)
        then
            return false
        else
            foreach e in f.Entries
                if (e.ForestTrustType = ForestTrustDomainInfo and
                    e.DomainInfo.NetbiosName = name and
                    NETBIOS_DISABLED_MASK not in e.Flags and
                    ForestTrustOwnsName(o, e.DomainInfo.DnsName) then
                    return true
                endif
            endfor
        endif
    endfor
endfor

return false

```

5.59 FOREST_TRUST_RECORD_TYPE

FOREST_TRUST_RECORD_TYPE is a concrete type for specifying the type of record contained in a forest trust information ([FOREST_TRUST_INFORMATION](#)) entry. The allowed values are specified by the following enumerated list.

```

typedef enum
{
    ForestTrustTopLevelName = 0,
    ForestTrustTopLevelNameEx = 1,
    ForestTrustDomainInfo = 2,
    ForestTrustDomainInfo = ForestTrustDomainInfo
} FOREST_TRUST_RECORD_TYPE;

```

5.60 ForestRootDomainNC

```

procedure ForestRootDomainNC(): DSName

```

The ForestRootDomainNC procedure returns the [DSName](#) of the forest root domain NC.

5.61 FullReplicaExists

```

procedure FullReplicaExists(nc : DSName) : boolean

```

The FullReplicaExists procedure returns true if the NC replica with root nc is a full replica.

```

if not ObjExists(nc) then
    return false
endif
return nc in (DSAObj()!msDS-hasMasterNCs +

```

```
DSAObj() !msDS-hasFullReplicaNCs)
```

5.62 GCPAS

```
procedure GCPAS() : PARTIAL_ATTR_VECTOR_V1_EXT
```

Informative summary of behavior: The GCPAS procedure returns a reference to an instance of the [PARTIAL_ATTR_VECTOR_V1_EXT](#) structure, which contains the list of attributes that may be present, based on the schema, on a GC NC replica.

```
partialAttrSetSeq: sequence of DSName
pPartialAttrVector: PARTIAL_ATTR_VECTOR_V1_EXT^

partialAttrSetSeq := select o from subtree SchemaNC() where
    (o!isMemberOfPartialAttributeSet = true)
pPartialAttrVector = new PARTIAL_ATTR_VECTOR_V1_EXT sized to hold
    partialAttrSetSeq.length entries in its rgPartialAttr
    field
pPartialAttrVector^.dwVersion := 1
pPartialAttrVector^.cAttrs := partialAttrSetSeq.length
for i := 0 to partialAttrSetSeq.length-1
    pPartialAttrVector^.rgPartialAttr[i] :=
        AttrtypFromSchemaObj(partialAttrSetSeq[i])
endfor

return pPartialAttrVector
```

5.63 GetFilteredAttributeSet

```
procedure GetFilteredAttributeSet() : sequence of ATTRTYP
```

Informative summary of behavior: The GetFilteredAttributeSet procedure returns a sequence of [ATTRTYP](#) that represents the list of attributes that may not be present on a filtered NC replica

```
filteredAttrSet: sequence of ATTRTYP
filteredAttrSetObjSeq: sequence of DSName
i: int

filteredAttrSetObjSeq := select o from subtree SchemaNC() where
    (fRODCFilteredAttribute in o!searchFlags) and
    (not FLAG_ATTR_REQ_PARTIAL_SET_MEMBER in
        o!systemFlags) and
    (not o!systemOnly = true) and
    (not AttrtypFromSchemaObj(o) in
        {currentValue, dbcsPwd, unicodePwd,
          ntPwdHistory, priorValue,
          supplementalCredentials, trustAuthIncoming,
          trustAuthOutgoing, lmPwdHistory,
          initialAuthIncoming, initialAuthOutgoing,
          msDS-ExecuteScriptPassword, displayName,
          codePage, creationTime, lockoutDuration,
          lockoutObservationWindow, logonHours,
```

```

        lockoutThreshold, maxPwdAge, minPwdAge,
        minPwdLength, netbiosName, pwdProperties,
        pwdHistoryLength, pwdLastSet,
        securityIdentifier, trustDirection,
        trustPartner, trustPosixOffset, trustType,
        rid, domainReplica, accountExpires,
        ntMixedDomain, OperatingSystem,
        OperationSystemVersion,
        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))

for i := 0 to filteredAttrSetObjSeq.length-1
    filteredAttrSet[i] :=
        AttrtypFromSchemaObj(filteredAttrSetObjSeq [i])
endfor

return filteredAttrSet

```

5.64 GetNCType

```

procedure GetNCType(nc: DSName) : ULONG

```

Informative summary of behavior: The GetNCType procedure returns the type of the NC replica.

```

ncType: ULONG

ncType = 0;

if not AmIRODC() then
    if not nc = ConfigNC() and
        not nc = SchemaNC() and
        not nc = DefaultNC() and
        IsApplicationNC(nc) = false then
        /* the NC replica correspond to a GC partition */
        ncType := ncType + {NCT_GC_PARTIAL}
    endif
else if
    if nc = ConfigNC() or
        nc = DefaultNC() or
        ApplicationNC(nc) = true then
        ncType := ncType + {NCT_FILTERED_ATTRIBUTE_SET,
                            NCT_SPECIAL_SECRET_PROCESSING }
    else if nc = SchemaNC() then
        ncType := 0
    else
        ncType := ncType + {NCT_FILTERED_ATTRIBUTE_SET,
                            NCT_GC_PARTIAL}
    endif
endif

```

```

endif
endif
return ncType

```

5.65 GetAttrVals

```

procedure GetAttrVals(
    o: DSName,
    att: ATTRTYP,
    includeDeletedLinks: boolean): set of attribute value

```

The GetAttrVals procedure constructs a set V that contains each value of the attribute att from object o.

If att is not a link attribute, the value of includeDeletedLinks is ignored. If att is a link attribute and includeDeletedLinks = false, the set includes only those values v of att such that [LinkStamp](#)(o, att, v).timeDeleted = 0. If att is a link attribute and includeDeletedLinks = true, the set contains all values v of att, even those such that [LinkStamp](#)(o, att, v).timeDeleted ≠ 0.

If the V is empty, null is returned. Otherwise, V is returned.

5.66 GetCallerAuthzInfo

```

procedure GetCallerAuthorizationInfo(): ClientAuthorizationInfo

```

The GetCallerAuthzInfo procedure returns the [ClientAuthorizationInfo](#) (a security token) of the current caller. For more information, see [\[MS-DTYP\]](#) section 2.5.2.

5.67 GetDefaultObjectCategory

```

procedure GetDefaultObjectCategory(class: ATTRTYP): DSName

```

The GetDefaultObjectCategory procedure returns the [defaultObjectCategory](#) value for the object class *class*.

```

classObj: DSName
classObj := SchemaObj(class)
return classObj!defaultObjectCategory

```

5.68 GetDSNameFromDN

```

procedure GetDSNameFromDN(dn: unicodestring): DSName

```

The GetDSNameFromDN procedure produces a [DSName](#) from the DN dn. Let d represent the returned [DSName](#). It is the case that d.dn = dn and d.guid = dn![objectGUID](#). Furthermore, if dn![objectSid](#) ≠ null, then d.sid = dn![objectSid](#), otherwise d.sid has no value.

5.69 GetForestFunctionalLevel

```
procedure GetForestFunctionalLevel(): integer
```

The GetForestFunctionalLevel procedure returns the forest functional level (see [\[MS-ADTS\]](#) section 7.1.4.4).

```
partitionsContainer: DSName
partitionsContainer:= DescendantObject(ConfigNC(), "CN=Partitions,")
if partitionsContainer!msDS-Behavior-Version = null then
    return DS_BEHAVIOR_WIN2000
else
    return partitionsContainer!msDS-Behavior-Version
endif
```

5.70 GetFSMORoleOwner

```
procedure GetFSMORoleOwner(role: integer): DSName
```

The GetFSMORoleOwner procedure returns the [DSName](#) of the [nTDSDSA](#) object of the DC that owns the FSMO role specified by role. The following table lists the valid values for role.

Symbolic constant	Value
FSMO_SCHEMA	1
FSMO_DOMAIN_NAMING	2
FSMO_PDC	3
FSMO_RID	4
FSMO_INFRASTRUCTURE	5

5.71 GetInstanceNameFromSPN

```
procedure GetInstanceNameFromSPN(spn: unicodestring): unicodestring
```

The GetInstanceNameFromSPN procedure syntactically extracts and returns the instance name from a two-part or three-part SPN. The instance name is the second part of the SPN. For example, dc-01.fabrikam.com is the instance name in the two-part SPN "ldap/dc-01.fabrikam.com" and in the three-part SPN "ldap/dc-01.fabrikam.com/fabrikam.com".

5.72 GetObjectNC

```
procedure GetObjectNC(o: DSName): DSName
```

The GetObjectNC procedure returns the [DSName](#) of the NC in which the object whose [DSName](#) is o is located.

5.73 GetProxyEpoch

```
procedure GetProxyEpoch(dnbinValue: DNBinary): DWORD
```

The GetProxyEpoch procedure returns the decoded proxy epoch field from the dnbinValue, which is a [proxiedObjectName](#) value.

5.74 GetProxyType

```
procedure GetProxyType(dnbinValue: DNBinary): DWORD
```

The GetProxyType procedure returns the decoded proxy type field from the dnbinValue, which is a [proxiedObjectName](#) value.

5.75 GetServiceClassFromSPN

```
procedure GetServiceClassFromSPN(spn: unicodestring): unicodestring
```

The GetServiceClassFromSPN procedure syntactically extracts and returns the service class from a two-part or three-part SPN. The service class is the first part of the SPN. For example, "ldap" is the service class in the two-part SPN "ldap/dc-01.fabrikam.com" and in the three-part SPN "ldap/dc-01.fabrikam.com/fabrikam.com".

5.76 GetServiceNameFromSPN

```
procedure GetServiceNameFromSPN(spn: unicodestring): unicodestring
```

The GetServiceNameFromSPN procedure syntactically extracts and returns the service name from a three-part SPN. If the supplied SPN is a two-part SPN, it will return null. The service name is the third part of the SPN. For example, "fabrikam.com" is the service name in the three-part SPN "ldap/dc-01.fabrikam.com/fabrikam.com".

5.77 groupType Bit Flags

The groupType bit flags may appear in values of the [groupType](#) attribute that define a group type.

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	A	R	U	B	Q	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
	G	G	G	G	G																										E

X: Unused. MUST be zero and ignored.

AG (GROUP_TYPE_ACCOUNT_GROUP): The account group type.

RG (GROUP_TYPE_RESOURCE_GROUP): The resource group type.

UG (GROUP_TYPE_UNIVERSAL_GROUP): The universal group type.

BG (GROUP_TYPE_APP_BASIC_GROUP): The application basic group type.

QG (GROUP_TYPE_APP_QUERY_GROUP): The application query group type.

SE (GROUP_TYPE_SECURITY_ENABLED): The group is security-enabled.

5.78 GUID

A concrete type, as specified in [\[C706\]](#) and [\[MS-DTYP\]](#) section **2.3.2**.

The type GUID has a well-defined null value, which is all zeros. The constant [NULLGUID](#) is equal to this value.

When comparing two GUID values, each GUID value is treated as an octet string in little-endian byte order.

Two GUID values g1 and g2 are equal if they are octet-for-octet identical.

Value g1 is less than value g2 only if there exists an N (where N is less than the size of the GUID type in octets) such that octets 0...N-1 of g1 and g2 are identical, and octet N of g1 is less than octet N of g2.

Value g1 is greater than value g2 only if there exists an N (where N is less than the size of the GUID type in octets) such that octets 0...N-1 of g1 and g2 are identical, and octet N of g1 is greater than octet N of g2.

5.79 GuidFromString

```
procedure GuidFromString(strGuid: unicodestring): GUID
```

The GuidFromString procedure converts the string representation of a GUID specified in strGuid (for example, "{12AA5F43-C776-4D63-B347-1175DF806200}" or "12aa5f43-c776-4d63-b347-1175df806200") to a binary GUID. The string representation is that defined in [\[RFC4122\]](#) section 3, or such a representation prefixed with a brace ({) and suffixed with a brace (}). If strGuid is not a valid string representation of a GUID, null is returned.

5.80 GuidToString

```
procedure GuidToString(guid: GUID): unicodestring
```

The GuidToString procedure converts the guid to the concatenation of "{", the string representation defined in [\[RFC4122\]](#) section 3, and "}" for example, {12aa5f43-c776-4d63-b347-1175df806200}.

5.81 handle_t

The handle_t is a concrete type for an RPC binding handle, as specified in [\[C706\]](#) section [4.2.9.7](#) and [\[MS-DTYP\]](#) section **2.1.2**.

5.82 instanceType Bit Flags

The [instanceType](#) bit flags are bits that may appear in values of the [instanceType](#) attribute.

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
N	U	W	N	N	N	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
H	I	R	A	C	G																										

X: Unused. MUST be zero and ignored.

NH (IT_NC_HEAD): The object is the root of an NC.

UI (IT_UNINSTANT): The NC replica has not yet been instantiated.

WR (IT_WRITE): The object is writable.

NA (IT_NC_ABOVE): The DC hosts the NC prior to this one. IT_NC_HEAD must also be set.

NC (IT_NC_COMING): The NC replica is in the process of being constructed for the first time via replication. IT_NC_HEAD must also be set.

NG (IT_NC_GOING): The NC replica is in the process of being removed from the DC. IT_NC_HEAD must also be set.

5.83 Is2PartSPN

```
procedure Is2PartSPN(spn: unicodestring): boolean
```

The Is2PartSPN procedure returns true if *spn* is an SPN consisting of two parts, and false otherwise.

5.84 Is3PartSPN

```
procedure Is3PartSPN(spn: unicodestring): boolean
```

The Is3PartSPN procedure returns true if *spn* is an SPN consisting of three parts, and false otherwise.

5.85 IsBuiltinPrincipal

```
procedure IsBuiltinPrincipal(sid: SID): boolean
```

The IsBuiltinPrincipal procedure returns true if *sid* is the SID of a built-in security principal, and returns false if it is not.

5.86 IsDomainNameInTrustedForest

```
procedure IsDomainNameInTrustedForest(name: unicodestring) : boolean
```

The IsDomainNameInTrustedForest procedure returns true if the domain identified by *name* is in a forest trusted by the caller's forest, as determined by the [FOREST_TRUST_INFORMATION](#) state of the caller's forest, and false otherwise. The *name* parameter can be either a DNS name or a NetBIOS name of a domain.

See section [5.58](#) for the specification of this procedure.

5.87 IsDCAccount

```
procedure IsDCAccount(o: DSName): Boolean
```

The IsDCAccount procedure returns true if the object represents the computer account of a DC.

5.88 IsGC

```
procedure IsGC(): boolean
```

The IsGC procedure returns true if the DC on which it is called is a global catalog server, and false if it is not.

5.89 IsGetNCChangesPermissionGranted

```
procedure IsGetNCChangesPermissionGranted(  
    msgIn: DRS_MSG_GETCHG_REQ_V8) : boolean
```

Informative summary of behavior: The IsGetNCChangesPermissionGranted procedure returns true if the source DC has permission to replicate objects and its attributes from the NC replica, as defined in msgIn.

```
ncRoot: DSName  
clientDsaObj: DSName  
serverObj: DSName  
cachedAt: DSName  
cachedUser: DSName  
fRevealSecret: boolean  
fRevealFilteredSet: boolean  
  
ncRoot := GetObjectNC(msgIn.pNC^)  
if not AccessCheckCAR(ncRoot, Ds-Replication-Get-Changes) then  
    return false  
endif  
  
fRevealSecret := true  
  
if IsRevealSecretRequest(msgIn) then  
    if AccessCheckCAR(ncRoot, Ds-Replication-Get-Changes-All) = false  
        then  
        if (msgIn.ulExtendedOp = EXOP_REPL_SECRETS) then  
            clientDsaObj := select one o from ConfigNC() where  
                             o!objectGUID = msgIn.uuidDsaObjDest  
            serverObj := clientDsaObj!parent  
            cachedAt := serverObj!serverReference  
            cachedUser := msgIn.pNC^  
            fRevealSecret := RevealSecretsForUserAllowed(  
                             cachedAt, cachedUser)  
        else  
            fRevealSecret := false  
        endif  
    endif
```

```

    endif
endif

fRevealFilteredSet := true
if IsRevealFilteredAttr(msgIn) then
    if (AccessCheckCAR(ncRoot, Ds-Replication-Get-Changes-All) = false
        and
        AccessCheckCAR(ncRoot, Ds-Replication-Get-Changes-In-Filtered-Set)
        = false) then
        fRevealFilteredSet := false
    endif
endif

if (fRevealSecret = false) or (fRevealFilteredSet = false)
    return false
else
    return true
endif

```

5.90 IsGUIDBasedDNSName

```

procedure IsGUIDBasedDNSName(o: DSName, instanceName: uncodestring):
    boolean

```

The IsGUIDBasedDNSName procedure returns true if instanceName is the DNS host name of the DC, identified by o, constructed in the form "<DSA GUID>._msdcs.<DNS forest name>".

5.91 IsMemberOfBuiltinAdminGroup

```

procedure IsMemberOfBuiltinAdminGroup(): boolean

```

The IsMemberOfBuiltinAdminGroup function returns true if the client security context is a member of the BUILTIN\Administrators group, and false if it is not. The BUILTIN\Administrators group is the group with the SID S-1-5-32-544.

5.92 IsRevealFilteredAttribute

```

procedure IsRevealFilteredAttribute(
    DRS_MSG_GETCHG_REQ_V8 msgIn) : boolean

```

Informative summary of behavior: The IsRevealFilteredAttribute procedure returns true if the source DC is requesting attributes in the filtered set.

```

filteredAttrSet: sequence of ATTRTYP
i: int

filteredAttrSet := GetFilteredAttributeSet()

for i := 0 to msgIn.pPartialAttrSet.cAttrs - 1
    if msgIn.pPartialAttrSet.rgPartialAttr[i] in
        filteredAttrSet then

```

```

        return true;
    endif
endfor

for i := 0 to msgIn.pPartialAttrSetEx.cAttrs - 1
    if msgIn.pPartialAttrSetEx.rgPartialAttr[i] in
        filteredAttrSet then
        return true;
    endif
endfor

return false;

```

5.93 IsRevealSecretRequest

```

procedure IsRevealSecretRequest (DRS_MSG_GETCHG_REQ_V8 msgIn)
    : boolean

```

Informative summary of behavior: The IsRevealSecretRequest procedure returns true if the source DC is requesting secret attributes.

```

if AmILHDC() = false then
    if (DRS_WRITE_REP in msgIn.ulFlags) then
        return true
    else
        return false
    endif
endif

/* if source DC is requesting FSMO related operation then it is same
   as a reveal secret request */
if (msgIn.ulExtendedOp = EXOP_FSMO_REQ_ROLE      or
    msgIn.ulExtendedOp = EXOP_FSMO_REQ_RID_ALLOC or
    msgIn.ulExtendedOp = EXOP_FSMO_REQ_RID_REQ_ROLE or
    msgIn.ulExtendedOp = EXOP_FSMO_REQ_PDC      or
    msgIn.ulExtendedOp = EXOP_FSMO_ABANDON_ROLE) then
    return true
endif

/* if source DC is requesting for special secrets processing then it
   implies that it is not requesting for secrets */
if ({DRS_SPECIAL_SECRET_PROCESSING} ∩ msgIn.ulFlags) then
    return false
endif

if (msgIn.ulExtendedOp = EXOP_REPL_SECRETS or
    msgIn.pAttributeSet = null then /* requesting all attributes that
                                     includes secrets*/
    return true
endif

for i := 0 to msgIn.pPartialAttrSet.cAttrs - 1
    if IsSecretAttribute(msgIn.pPartialAttrSet.rgPartialAttr[i]) then
        return true;
    endif
endfor

```

```

for i := 0 to msgIn.pPartialAttrSetEx.cAttrs - 1
    if IsSecretAttribute(msgIn.pPartialAttrSetEx.rgPartialAttr[i]) then
        return true;
    endif
endfor

return false;

```

5.94 IsValidServiceName

```

procedure IsValidServiceName(o: DSName, serviceName: unicodestring):
    boolean

```

The IsValidServiceName procedure returns true if the name serviceName is a valid service name in an SPN for the DC represented by [computer](#) object o.

A valid service name can be one of the following:

- For GC SPNs, the service name must be the DNS forest name.
- For other classes of SPNs, the service name must be either the DNS domain name of the DC's default domain or the DNS domain name of an application NC hosted by the DC.

5.95 KCCFailedConnections

KCCFailedConnections is an abstract type consisting of a sequence of tuples, one tuple for each DC for which the connection attempt failed. Each tuple contains the following fields:

- **DsaDN:** A *unicodestring* (see section [3.4.3](#)) that contains the DN of the [nTDSDSA](#) object that corresponds to the DC.
- **UUIDDsa:** A [GUID](#) that contains the DSA GUID of the DC.
- **TimeFirstFailure:** A [FILETIME](#) that contains the time when the Knowledge Consistency Checker (KCC) noticed the first failure while contacting the DC.
- **FailureCount:** An integer that contains the total number of failures KCC encountered while contacting the DC.
- **LastResult:** A [DWORD](#) that contains a Windows error code that indicates the reason for the last failure.

The global variable [dc](#) for a DC has an associated field [dc.kccFailedConnections](#), which maintains the DC's KCCFailedConnections state.

5.96 KCCFailedLinks

KCCFailedLinks is an abstract type that consists of a sequence of tuples, one tuple for each neighboring DC for which a connection attempt failed.

The fields of the tuple are the same as the fields of the [KCCFailedConnections](#) tuple.

The global variable [dc](#) for a DC has an associated field [dc.kccFailedLinks](#), which maintains the DC's KCCFailedLinks state.

5.97 LARGE_INTEGER

LARGE_INTEGER is a concrete type for a 64-bit signed integer, as specified in [\[MS-DTYP\]](#) section 2.3.3.

5.98 LDAP_CONN_PROPERTIES

LDAP_CONN_PROPERTIES is a concrete type that contains bit flags that identify properties of an LDAP connection.

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
B	S	U	G	G	N	S	M	S	S	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
N	S	D	C	S	G	P	D	G	L																						
D	L	P		S	O	L	S	N																							

X: Unused. MUST be zero and ignored.

BND: A bind has been performed on this LDAP connection.

SSL: The LDAP connection corresponds to a Secure Sockets Layer (SSL) connection.

UDP: The LDAP connection corresponds to a User Datagram Protocol (UDP) connection.

GC: The LDAP connection was made through the GC port.

GSS: The Generic Security Services Application Programming Interface (GSS-API) security package was used for authentication.

NGO: The SPNEGO security package was used for authentication.

SPL: The LDAP bind corresponds to LDAP simple bind.

MD5: The Digest-MD5 security package was used for authentication.

SGN: Signing is enabled on the LDAP connection.

SL: Sealing is enabled on the LDAP connection.

5.99 LDAPConnections

LDAPConnections is an abstract type for the LDAP connections associated with a DC. It is a sequence of tuples, one tuple per LDAP connection currently open. Each tuple contains the following fields:

- **iPAddress:** A [DWORD](#) that contains the IPv4 address of the client machine that established the connection.
- **notificationCount:** An integer that contains the number of LDAP notifications enabled on the connection.
- **secTimeConnected:** An integer that contains the time, in seconds, that the connection has been open.

- **flags:** A [DWORD](#) that contains the [LDAP_CONN_PROPERTIES](#) bit flags that identify the properties of the connection.
- **totalRequests:** An integer that contains the total number of LDAP requests processed on the connection.
- **userName:** A *unicodestring* (see section [3.4.3](#)) that contains the name of the security principal that opened the connection.

The global variable [dc](#) for a DC has an associated field [dc.ldapConnections](#), which maintains the DC's [LDAPConnections](#) state.

5.100 LinkStamp

```
procedure LinkStamp(
  o: DSName;
  att: ATTRTYP;
  val: attribute value): LinkValueStamp
```

The [LinkStamp](#) procedure returns the [LinkValueStamp](#) associated with the last update to add or remove value *val* from the forward link attribute *att* of object *o*. If *val* was last updated when the forest functional level was [DS_BEHAVIOR_WIN2000](#) (see [\[MS-ADTS\]](#) section 7.1.4.4), no [LinkValueStamp](#) is associated with *val*, and [LinkStamp](#) returns null.

5.101 LinkValueStamp

[LinkValueStamp](#) is an abstract type that denotes information about the last update to a link value of an object. It is a tuple that consists of all the fields in [AttributeStamp](#), plus the following additional fields:

- **timeCreated:** The date and time at which the first originating update was made.
- **timeDeleted:** The date and time at which the last replicated update was made that deleted the value, or 0 if the value is not currently deleted.

Comparisons

Values of [LinkValueStamp](#) are partially ordered. Let *d* be the result of *x.dwVersion* - *y.dwVersion*, cast as a 32-bit integer. Then given two stamps *x* and *y*, *x* is said to be greater than *y* if any of the following is true:

- *x* is not null and *y* is null
- *x.timeCreated* > *y.timeCreated*
- *x.timeCreated* = *y.timeCreated* and *d* > 0
- *x.timeCreated* = *y.timeCreated* and *d* = 0 and *x.timeChanged* > *y.timeChanged*
- *x.timeCreated* = *y.timeCreated* and *d* = 0 and *x.timeChanged* = *y.timeChanged* and *x.uuidOriginating* > *y.uuidOriginating*

Conversions

A value *x* of type [LinkValueStamp](#) may be converted to and from its wire format *y* of type [PROPERTY_META_DATA_EXT](#) by associating the values of fields in *x* with the values of the like-

named fields in y and y.MetaData.**Note** The value of timeDeleted does not appear in the wire format. On the wire, the [PROPERTY META DATA EXT](#) value always appears as a value of the MetaData field of a [REPLVALINE](#) structure. Given value x of type LinkValueStamp and value z of type [REPLVALINE](#), z.IsPresent is TRUE if x.timeDeleted is 0 and FALSE if x.timeDeleted is nonzero.

5.102 LocalAttidFromRemoteAttid

```
procedure LocalAttidFromRemoteAttid(  
    remotePT: PrefixTable,  
    remoteAttid : ATTRTYP) : ATTRTYP
```

Informative summary of behavior: LocalAttidFromRemoteAttid converts the attribute ID remoteAttid based on the prefix table remotePT to an attribute ID based on [dc.prefixTable](#).

```
oid : OID  
oid := OidFromAttid(remotePT, remoteAttid)  
return MakeAttid(dc.prefixTable, oid)
```

5.103 LONG

LONG is a concrete type for a 32-bit, signed integer, as specified in [\[MS-DTYP\]](#) section 2.2.

5.104 LONGLONG

LONGLONG is a concrete type for a 64-bit, signed integer, as specified in [\[MS-DTYP\]](#) section 2.2.

5.105 LPWSTR

LPWSTR is a concrete type for a pointer to a string of double-byte Unicode characters, as specified in [\[MS-DTYP\]](#) section 2.2.

5.106 MakeAttid

```
procedure MakeAttid(var t: PrefixTable, o: OID) : ATTRTYP
```

The MakeAttid procedure translates an abstract [OID](#) o to a concrete [ATTRTYP](#), using the prefix table specified by t. This procedure may mutate the supplied prefix table. See section [5.15.4](#) for the specification of this procedure.

5.107 MakeProxyValue

```
procedure MakeProxyValue(  
    dnPart: DSName,  
    proxyType: DWORD,  
    proxyEpoch: DWORD): DNBinary
```

The MakeProxyValue procedure constructs and returns a value in the [proxiedObjectName](#) format (see section [5.139](#)) from the provided parts. Let d be the returned [DNBinary](#). d.dn equals dnPart and d.binary is constructed from proxyType and proxyEpoch.

5.108 MasterReplicaExists

```
procedure MasterReplicaExists(nc : DSName) : boolean
```

The MasterReplicaExists procedure returns true only if the NC replica with root nc is a writable NC replica.

```
  If not ObjExists(nc) then
    return false
  endif
  return nc in DSAObj()!msDS-hasMasterNCs
```

5.109 MD5_CTX

MD5_CTX is an abstract type defined in [\[RFC1321\]](#).

5.110 MD5Final

```
procedure MD5Final(var context: MD5_CTX)
```

The MD5Final procedure performs the MD5Final algorithm, as specified in [\[RFC1321\]](#).

5.111 MD5Init

```
procedure MD5Init(var context: MD5_CTX)
```

The MD5Init procedure performs the MD5Init algorithm, as specified in [\[RFC1321\]](#).

5.112 MD5Update

```
procedure MD5Update(
  var context: MD5_CTX,
  input: sequence of BYTE,
  inputLen: integer)
```

The MD5Update procedure performs the MD5Update algorithm, as specified in [\[RFC1321\]](#).

5.113 MergeUTD

```
procedure MergeUTD(
  utd1: UPTODATE_VECTOR_V1_EXT,
  utd2: UPTODATE_VECTOR_V1_EXT): UPTODATE_VECTOR_V1_EXT
```

Informative summary of behavior: The client does not want to include objects in the inconsistency-detection process that have not yet replicated. To meet this goal, it uses the MergeUTD procedure to compute an [UPTODATE VECTOR V1 EXT](#) that has minimal pairwise values for each uuidDsa.

MergeUTD is specified by the following normative semantics:

For every *uuidDsa* that is in both *utd1* and *utd2*, add a *uuidDsa* to the returned [UPTODATE VECTOR V1 EXT](#) with a corresponding **USN** value such that the USN is smaller than the USNs corresponding to the *uuidDsa* in *utd1* and *utd2*.

5.114 MTX_ADDR

The **MTX_ADDR** structure defines a concrete type for the network name of a DC.

```
typedef struct {
    [range(1,256)] unsigned long mtx_namelen;
    [size_is(mtx_namelen)] char mtx_name[];
} MTX_ADDR;
```

mtx_namelen: A 32-bit, unsigned integer that specifies the number of bytes in *mtx_name*, including a null terminating character.

mtx_name: The UTF-8 encoding of a [NetworkAddress](#).

The following table shows an alternative representation of this 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
mtx_namelen																															
mtx_name (variable length) ...																															

5.115 NCType Bits

Bit flags describing NCType.

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
S	G	F	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
P	P	P																													

X: Unused. MUST be zero and ignored.

SP (NCT_SPECIAL_SECRET_PROCESSING): The NC replica requests special secret processing while replicating objects.

GP (NCT_GC_PARTIAL): Objects in the NC replica can only have attributes that are specified in the GC partial attribute set.

FP (NCT_FILTERED_ATTRIBUTE_SET): Objects in the NC replica do not have attributes defined in the filtered attribute set.

5.116 NetworkAddress

NetworkAddress is an abstract type for the transport-specific address of a DC represented as a UTF-8 string. For the RPC transport, the address is a DNS name (see [\[MS-ADTS\]](#) section 7.3.2.1). For the SMTP transport, the address is an SMTP address (as specified in [\[RFC2821\]](#) and [\[MS-SRPL\]](#)).

In AD/LDS, the NetworkAddress also includes the LDAP port of the destination AD/LDS DC. This data is used as RPC endpoint annotation to select the endpoint corresponding to the selected AD/LDS DC, because a machine can hold multiple AD/LDS DCs.

A NetworkAddress is stored as an `mtx_name` within an [MTX_ADDR](#) structure, which in turn is stored within a [REPS_TO](#) structure.

5.117 NewPrefixTable

```
procedure NewPrefixTable() : PrefixTable
```

The `NewPrefixTable` procedure creates a new [PrefixTable](#) that contains a set of default prefixes. See section [5.15.4](#) for the specification of this procedure.

5.118 Nt4ReplicationState

Nt4ReplicationState is an abstract type for the replication state for Windows NT 4.0 BDCs. It is a tuple that contains the following fields:

- **SamNT4ReplicationUSN:** A [USN](#) that records the replication update sequence number for SAM database updates that are relevant to the Windows NT 4.0 replication protocol. Relevant updates are described in [\[MS-ADTS\]](#) section 3.1.1.7.2.2.
- **SamCreationTime:** A [FILETIME](#) at which the Windows NT 4.0 replication update sequence number for the SAM database was set to 1.
- **BuiltinNT4ReplicationUSN:** A [USN](#) that records the replication update sequence number for built-in database updates that are relevant to the Windows NT 4.0 replication protocol. The built-in database contains the objects for built-in principals.
- **BuiltinCreationTime:** A [FILETIME](#) at which the replication update sequence number for the built-in database was set to 1.

The global variable [dc](#) for a DC has an associated field [dc.nt4ReplicationState](#). When a DC owns the PDC FSMO role, this field contains its Nt4ReplicationState. [\[MS-ADTS\]](#) section 3.1.1.7.1.1 describes how the components of [dc.nt4ReplicationState](#) are maintained and used to support replication via the Windows NT 4.0 replication mechanism. As an implementation-specific behavior, other DCs may maintain the [dc.nt4ReplicationState](#) field as well.

5.119 NT4SID

The **NT4SID** structure defines a concrete type for an SID.

```
typedef struct {  
    char Data[28];  
} NT4SID;
```

Data: Bytes that make up a SID structure, as specified in [\[MS-DTYP\]](#) section **2.4.2**, in little-endian byte order.

5.120 NTDSTRANSPORT_OPT Values

The valid system flags used on directory objects are specified in [\[MS-ADTS\]](#) section 7.1.1.2.2.3.1.

5.121 NULLGUID

NULLGUID is a value of type [GUID](#) that is entirely zero, that is, {00000000-0000-0000-0000-000000000000}.

5.122 ObjExists

```
procedure ObjExists(dsName: DSName): boolean
```

The ObjExists procedure returns true if *dsName* identifies an object in some NC replica hosted by the server.

5.123 OID

OID is an abstract type for representing values of type [String\(Object-Identifier\)](#). Values of this type are a dotted decimal *unicodestring* (section [3.4.3](#)), for example, "1.2.840.113556.1.4.159".

5.124 OID_t

The **OID_t** structure defines a concrete type for an [OID](#) or a prefix of an OID; it is a component of type [PrefixTableEntry](#).

```
typedef struct {  
    [range(0,10000)] unsigned int length;  
    [size_is(length)] BYTE* elements;  
} OID_t;
```

length: The size, in bytes, of the elements array.

elements: The array of bytes that constitute an OID or a prefix of an OID.

5.125 OidFromAttid

```
procedure OidFromAttid(t: PrefixTable, attr: ATTRTYP) : OID
```

The OidFromAttid procedure translates a concrete [ATTRTYP](#) attr to an abstract [OID](#), using the prefix table specified by t. See section [5.15.4](#) for the specification of this procedure.

5.126 parent

parent is an abstract attribute that is present on every object, as specified in [\[MS-ADTS\]](#) section 3.1.1.1.3. This attribute is part of the state model but is not exposed in the Active Directory schema.

5.127 PARTIAL_ATTR_VECTOR_V1_EXT

The **PARTIAL_ATTR_VECTOR_V1_EXT** structure defines a concrete type for a set of attributes to be replicated to a given partial replica.

```
typedef struct {
    DWORD dwVersion;
    DWORD dwReserved1;
    [range(1,1048576)] DWORD cAttrs;
    [size_is(cAttrs)] ATTRTYP rgPartialAttr[];
} PARTIAL_ATTR_VECTOR_V1_EXT;
```

dwVersion: The version of this structure; MUST be 1.

dwReserved1: Reserved; MUST be 0 and ignored.

cAttrs: The count of attributes in the rgPartialAttr array.

rgPartialAttr: The attributes in the set.

5.128 partialAttributeSet

The abstract, non-replicated, single-valued attribute [partialAttributeSet](#) is an optional attribute on the NC root of every partial replica.

The abstract type set of [ATTRTYP](#) simplifies the specification of methods that read and write the attribute [partialAttributeSet](#). Reading the attribute [partialAttributeSet](#) returns a single value, which is of type set of [ATTRTYP](#). Each element in the set is an attribute that is in the subset of attributes replicated to the partial replica.

5.129 PartialGCReplicaExists

```
procedure PartialGCReplicaExists(nc : DSName) : boolean
```

The PartialGCReplicaExists procedure returns true if the NC replica with root nc is a partial NC replica.

```
if not ObjExists(nc) then
    return false
endif
return nc in DSAObj()!hasPartialReplicaNCs
```

5.130 PAS_DATA

PAS_DATA is a concrete type for a list of attributes in a partial attribute set.

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																															
size																															
flag																															
pas (variable)																															
...																															

version: The version of the structure; MUST be 1.

size: The size of the entire structure.

flag: Not used; MUST be 0.

pas: A [PARTIAL ATTR VECTOR V1 EXT](#) structure, which specifies the additional attributes being requested as part of a PAS cycle.

5.131 PdcChangeLog

PdcChangeLog is an abstract type for a sequence of [CHANGELOG ENTRY](#) objects. The change log is used to support replication of certain Active Directory changes to Windows NT 4.0 BDCs, and is specified in [\[MS-ADTS\]](#) section 3.1.1.7.

The global variable [dc](#) for a DC has an associated field [dc.pdcChangeLog](#). When a DC owns the PDC FSMO role, this field contains its change log. As an implementation-specific behavior, other DCs may maintain the [dc.pdcChangeLog](#) field as well.

5.132 PerformAddOperation

```
procedure PerformAddOperation(
  data: ENTINF,
  prefixTable: PrefixTable,
  var newObjectName: DSName): integer
```

The PerformAddOperation procedure performs an add operation with the given [ENTINF](#) to create a new object in the directory. For more information, see [\[MS-ADTS\]](#) section 3.1.1.5.2.

The resulting object has the [DSNAME](#) data.pName.

For each [ATTR attr](#) in data.AttrBlock, let attribute be the [ATTRTYP](#) returned by [LocalAttrIdFromRemoteAttrId](#)(prefixTable, attr.attrType). Then the object created by PerformAddOperation has an attribute whose [ATTRTYP](#) is attribute and that has the values attr.AttrVal.pAVal[0... attr.AttrVal.valCount].

If $\text{data.ulFlags} \cap \{\text{ENTINF_DYNAMIC_OBJECT}\} = \{\text{ENTINF_DYNAMIC_OBJECT}\}$, the resulting object is created as a dynamic object.

If the add operation succeeds, the [DSName](#) of the created object is returned in newObjectName and the procedure returns 0. If the add operation fails, the procedure returns a Windows error code.

5.133 PerformAddOperationAsSystem

```
procedure PerformAddOperationAsSystem(  
    data: ENTINF,  
    prefixTable: PrefixTable,  
    var newObjectName: DSNAME): integer
```

The PerformAddOperationAsSystem operation is identical to [PerformAddOperation](#), except that the add operation is performed as the system. When an operation is performed as the system, all access checks are bypassed and schema constraints are not enforced.

5.134 PerformModifyOperation

```
procedure PerformModifyOperation(  
    data: ENTINF,  
    prefixTable: PrefixTable): integer
```

The PerformModifyOperation procedure performs a modify operation with the given [ENTINF](#) to modify an existing object in the directory. For more information, see [\[MS-ADTS\]](#) section 3.1.1.5.3.

This operation modifies the object whose [DSNAME](#) is data.pName.

For each [ATTR attr](#) in data.AttrBlock, let attribute be the [ATTRTYP](#) returned by [LocalAttrIdFromRemoteAttrId](#)(prefixTable, attr.attrType). Then on the object modified by PerformModifyOperation, if the attribute whose [ATTRTYP](#) is attribute is present, all previous values are removed and replaced with the values attr.AttrVal.pAVal[0... attr.AttrVal.valCount]. If the attribute whose [ATTRTYP](#) is attribute is not present, it is added with the values attr.AttrVal.pAVal[0... attr.AttrVal.valCount].

If the modify operation succeeds, the procedure returns 0. If the modify operation fails, the procedure returns a Windows error code.

5.135 PrefixTable

PrefixTable is an abstract type for a prefix table. See section [5.15.4](#) for the specification of this type.

5.136 PrefixTableEntry

The **PrefixTableEntry** structure defines a concrete type for mapping a range of [ATTRTYP](#) values to and from [OIDs](#). It is a component of the type [SCHEMA_PREFIX_TABLE](#).

```
typedef struct {  
    unsigned long ndx;  
    OID_t prefix;  
} PrefixTableEntry;
```

ndx: The index assigned to the prefix.

prefix: An OID or a prefix of an OID.

5.137 PROPERTY_META_DATA_EXT

The **PROPERTY_META_DATA_EXT** structure defines a concrete type for the stamp of the last originating update to an attribute.

```
typedef struct {
    DWORD dwVersion;
    DSTIME timeChanged;
    UUID uuidDsaOriginating;
    USN usnOriginating;
} PROPERTY_META_DATA_EXT;
```

dwVersion: The version of the attribute values, starting at 1 and increasing by one with each originating update.

timeChanged: The time at which the originating update was performed.

uuidDsaOriginating: The invocationId of the DC that performed the originating update.

usnOriginating: The [USN](#) of the DC assigned to the originating update.

5.138 PROPERTY_META_DATA_EXT_VECTOR

The **PROPERTY_META_DATA_EXT_VECTOR** structure defines a concrete type for a sequence of attribute stamps.

```
typedef struct {
    [range(0,1048576)] DWORD cNumProps;
    [size_is(cNumProps)] PROPERTY_META_DATA_EXT rgMetaData[];
} PROPERTY_META_DATA_EXT_VECTOR;
```

cNumProps: The count of items in the **rgMetaData** array.

rgMetaData: The array of attribute stamps.

5.139 proxiedObjectName Value Format

Values of the [proxiedObjectName](#) attribute are of [DNBinary](#) type. The binary portion is composed of two [DWORDs](#), which are stored in big-endian format. The first DWORD contains the proxy type value. The following table lists the valid values.

Symbolic name	Value	Meaning
PROXY_TYPE_MOVED_OBJECT	0x0	An object that was moved by a cross-NC.
PROXY_TYPE_PROXY	0x1	Used by the reference update task. For more information, see [MS-ADTS] section 3.1.1.6.2.

The second DWORD is the proxy epoch value, which is a DWORD counter value that is used by the cross-NC move operation.

5.140 RDN

RDN is an abstract type for representing the relative distinguished name (RDN) (as specified in [\[RFC2253\]](#)).

5.141 rdnType

The `rdnType` attribute is an abstract attribute present on every object. The `rdnType` of an object is the RDN attribute of the object, that is, an [ATTRTYP](#) that identifies the [attributeSchema](#) object of the RDN attribute. `rdnType` is not represented in the schema and does not replicate in the normal way.

On an originating Add, the new object's `rdnType` is derived from the most specific structural object class of the new object.

On a replicated Add, `rdnType` is derived as follows:

- If the forest functional level is less than DS_BEHAVIOR_WIN2003, `rdnType` is derived from the [objectClass](#) of the object, which replicates.
- If the forest functional level is DS_BEHAVIOR_WIN2003 or greater, `rdnType` is derived from the DN of the object, which replicates.

5.142 RemoveObj

```
procedure RemoveObj(o: DSName)
```

`RemoveObj` deletes from the directory (into a tombstone) the object whose [DSName](#) is `o`.

5.143 REPLENTINFLIST

The **REPLENTINFLIST** structure defines a concrete type for updates to one or more attributes of a given object.

```
typedef struct REPLENTINFLIST {  
    struct REPLENTINFLIST* pNextEntInf;  
    ENTINF Entinf;  
    BOOL fIsNCPrefix;  
    UUID* pParentGuid;  
    PROPERTY_META_DATA_EXT_VECTOR* pMetaDataExt;  
} REPLENTINFLIST;
```

pNextEntInf: The next **REPLENTINFLIST** in the sequence, or NULL.

Entinf: The object and its updated attributes.

fIsNCPrefix: TRUE only if the object is an NC root.

pParentGuid: The value of the [objectGUID](#) attribute of the parent of the object, or NULL if not known or not specified.

pMetaDataExt: The stamps for the attributes specified in `Entinf.AttrBlock`. `Entinf.AttrBlock` and `pMetaDataExt.rgMetaData` are parallel arrays. For a given integer *i* in [0 ..

Entinf.AttrBlock.attrCount], the stamp for the attribute described by Entinf.AttrBlock.pAttr^[i] is pMetaDataExt^.rgMetaData[i].

5.144 ReplicationQueue

ReplicationQueue is an abstract type for queued pending replication operations. It is a sequence of tuples, one tuple for each queued replication operation that is pending. Each tuple contains the following fields:

- **TimeEnqueued:** A [FILETIME](#) that contains the time when the operation was enqueued.
- **SerialNumber:** A [ULONG](#) that contains a unique identifier associated with the operation.
- **Priority:** A [ULONG](#) that contains the priority of the operation.
- **OperationType:** An integer that indicates the type of operation, as defined in [DS_REPL_OP_TYPE](#).
- **Options:** A [DRS_OPTIONS](#) that contains options associated with the replication operation.
- **NamingContext:** A *unicodestring* (section [3.4.3](#)) that contains the NC root of the NC replica associated with the operation.
- **DsaDN:** A *unicodestring* (section [3.4.3](#)) that contains the DN of the [nTDSDSA](#) object of the DC associated with the operation.
- **DsaAddress:** A *unicodestring* (section [3.4.3](#)) that contains the network address of the DC associated with the operation.
- **UUIDNC:** A [GUID](#) that contains the [objectGUID](#) of the NC root of the NC replica associated with the operation.
- **UUIDDsa:** A [GUID](#) that contains the DSA GUID of the DC associated with the operation.

The global variable [dc](#) for a DC has an associated field [dc.replicationQueue](#), which maintains the DC's ReplicationQueue state.

5.145 REPLTIMES

The **REPLTIMES** structure defines a concrete type for when periodic replication occurs.

```
typedef struct {
    UCHAR rgTimes[84];
} REPLTIMES;
```

rgTimes: An 84-byte structure that is used to set periodic replication times. Each bit in this structure represents a 15-minute period for which replication can be scheduled within a one-week period. The replication schedule begins on Sunday at midnight, UTC.

The following diagram shows an alternative representation of this structure.

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
rgTimes...																															
...																															
...rgTimes																															

5.146 replUpToDateVector, ReplUpToDateVector

The abstract nonreplicated multivalued attribute replUpToDateVector is an optional attribute on the NC root of every NC replica.

The abstract type ReplUpToDateVector simplifies the specification of methods that read and write the attribute **replUpToDateVector**. Reading the attribute **replUpToDateVector** produces one or more ReplUpToDateVector values.

The type ReplUpToDateVector is a tuple with the following fields:

- **uuidDsa**: the invocation ID of the DC that assigned usnHighPropUpdate.
- **usnHighPropUpdate**: A [USN](#) at which an update was applied on the DC identified by uuidDsa.
- **timeLastSyncSuccess**: The time at which the last successful replication occurred from the DC identified by uuidDsa; for replication latency reporting only.

Given an NC replica *r*, if *c* is an element of *r*![replUpToDateVector](#), then all updates made by *c*.uuidDsa with USN ≤ *c*.usnHighPropUpdate have been applied to *r*.

5.147 REPLVALINF

The **REPLVALINF** structure defines a concrete type for the identity and stamp of a link value.

```
typedef struct {
    DSNAME* pObject;
    ATTRTYP attrTyp;
    ATTRVAL Aval;
    BOOL fIsPresent;
    VALUE_META_DATA_EXT_V1 MetaData;
} REPLVALINF;
```

pObject: Identifies the object with the attribute that contains the link value.

attrTyp: An attribute that contains the link value.

Aval: The link value.

fIsPresent: FALSE only if the link value has been removed from the attribute.

MetaData: The stamp associated with the link value.

5.148 REPS_FROM

REPS_FROM is a concrete type. The nonreplicated, multivalued attribute [repsFrom](#) is an optional attribute on the root object of every NC replica. It is stored with the structure REPS_FROM, which is represented below as a packet diagram.

Note In the following field descriptions, the source DC refers to the DC identified by the uuidDsaObj.

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
cb																															
cConsecutiveFailures																															
timeLastSuccess																															
...																															
timeLastAttempt																															
...																															
ulResultLastAttempt																															
cbOtherDraOffset																															
cbOtherDra																															
ulReplicaFlags																															
rtSchedule																															
...																															
...																															
...																															
...																															
...																															

...
...
(rtSchedule cont'd for 13 rows)
usnVec
...
...
...
...
...
...
uuidDsaObj
...
...
...
uuidInvocId
...
...
...
uuidTransportObj
...
...
...

dwReserved
cbPasDataOffset
data (variable)
...

cb: The total number of bytes in the REPS_FROM structure.

cConsecutiveFailures: An unsigned long that contains the number of consecutive failures that have occurred while replicating from the source DC.

timeLastSuccess: A [DSTIME](#) that contains the time of the last successful replication cycle with the source DC.

timeLastAttempt: A **DSTIME**. Contains the time of the last replication attempt with the source DC.

ulResultLastAttempt: An unsigned long that contains the result of the last replication attempt with the source DC.

cbOtherDraOffset: The offset from the start of the structure to a location in the data field, specifying the start of an [MTX_ADDR](#) structure that contains a [NetworkAddress](#) for the source DC.

cbOtherDra: The size of the **MTX_ADDR** structure pointed to by **cbOtherDraOffset**.

ulReplicaFlags: A [ULONG](#). This field contains a set of [DRS_OPTIONS](#) that are applicable when replicating from the source DC.

rtSchedule: A [REPLTIMES](#) structure. If periodic replication is enabled (ulReplicaFlags contains DRS_PER_SYNC), this field identifies the 15-minute intervals within each week when a replication cycle is initiated with the source DC.

usnVec: A [USN_VECTOR](#) structure. This holds 0 or the usnvecTo field from the response to the last [IDL_DRSGetNCChanges](#) replication request sent to the source DC.

uuidDsaObj: A [GUID](#) that is the DSA GUID of the source DC.

uuidInvocId: A [GUID](#) that contains the invocation ID of the source DC.

uuidTransportObj: A [GUID](#) that contains the [objectGUID](#) of the [interSiteTransport](#) object that corresponds to the transport used for communication with the source DC.

dwReserved: Not used; MUST be 0.

cbPasDataOffset: The offset from the start of the structure to a location in the data field, specifying the start of a [PAS_DATA](#) structure.

data: The storage for the rest of the structure. The **MTX_ADDR** structure and the **PAS_DATA** structure pointed to by **cbOtherDraOffset** and **cbPasDataOffset** are packed into this field and can be referenced using the offsets.

5.149 REPS_TO

REPS_TO is a concrete type. The nonreplicated, multivalued attribute [repsTo](#) is an optional attribute on the root object of every NC replica. It is stored as the structure REPS_TO, which is represented below as a packet diagram.

This structure is used for both [repsTo](#) values and [repsFrom](#) values. Many of the fields are unused in [repsTo](#) values, and some of the field names are misleading (for example, **timeLastSuccess**).

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
cb																															
cConsecutiveFailures																															
timeLastSuccess																															
...																															
timeLastAttempt																															
...																															
ulResultLastAttempt																															
cbOtherDraOffset																															
cbOtherDra																															
ulReplicaFlags																															
rtSchedule																															
...																															
...																															
...																															
...																															
...																															

...
...
(rtSchedule cont'd for 13 rows)
usnVec
...
...
...
...
...
...
uuidDsaObj
...
...
...
uuidInvocId
...
...
...
uuidTransportObj
...
...
...

dwReserved
cbPasDataOffset
data (variable)
...

cb: The total number of bytes in the REPS_TO structure.

cConsecutiveFailures: An unsigned long that contains the number of unsuccessful consecutive attempts to send a replication notification to the DC identified by uuidDsaObj.

timeLastSuccess: A [DSTIME](#) structure that contains the time when the last successful replication notification to the DC identified by uuidDsaObj was sent, or 0 if no replication notification has been sent successfully.

timeLastAttempt: A **DSTIME** structure that contains the last time when an attempt was made to send a replication notification to the DC identified by uuidDsaObj, or 0 if no attempt has been made.

ulResultLastAttempt: An unsigned long that contains the result of the last attempt to send a replication notification to the DC identified by uuidDsaObj. It has a value 0 if the last notification was sent successfully, or a Windows error code otherwise.

cbOtherDraOffset: The offset from the start of the structure to a location in the data field, specifying the start of an [MTX_ADDR](#) structure that contains a [NetworkAddress](#).

cbOtherDra: The size of the **MTX_ADDR** structure pointed to by **cbOtherDraOffset**.

ulReplicaFlags: A ULONG. This set contains DRS_WRIT_REP if this replica is writable. This set never contains any other elements.

rtSchedule: A [REPLTIMES](#) structure. Not used.

usnVec: A [USN_VECTOR](#) structure. Not used.

uuidDsaObj: A [GUID](#). A DSA GUID that identifies a DC.

uuidInvocId: A [GUID](#). Not used.

uuidTransportObj: A [GUID](#). Not used.

dwReserved: Not used; MUST be 0.

cbPasDataOffset: Not used.

data: The storage for the rest of the structure. The **MTX_ADDR** structure pointed to by **cbOtherDraOffset** is packed into this field and can be referenced using the offset.

5.150 repsFrom, RepsFrom

The nonreplicated, multivalued attribute [repsFrom](#) is an optional attribute on the root object of every NC replica. It is stored with the structure [REPS_FROM](#).

The abstract type RepsFrom simplifies the specification of methods that read and write the attribute [repsFrom](#). Reading the attribute [repsFrom](#) produces one or more RepsFrom values using the conversions from [REPS_FROM](#) specified below. Writing a RepsFrom value to the attribute [repsFrom](#) stores a [REPS_FROM](#) using the reverse conversion.

The type RepsFrom is a tuple with the following fields:

- **naDsa:** A [NetworkAddress](#) of the DC that corresponds to cbOtherDraOffset and cbOtherDra in [REPS_FROM](#).
- **uuidDsa:** A [GUID](#) that corresponds to uuidDsaObj in [REPS_FROM](#). This is the DSA GUID of the DC.
- **options:** A [ULONG](#) that corresponds to ulReplicaFlags in [REPS_FROM](#). This set contains one or more of the following values chosen from [DRS_OPTIONS](#):
 - DRS_WRIT_REP: The replica is a full (read/write) replica of the NC.
 - DRS_INIT_SYNC: The replica must be replicated from the DC identified by uuidDsa when the DC hosting this replica is started.
 - DRS_PER_SYNC: Periodically replicates the NC replica from the DC identified by uuidDsa, as defined by the periodic replication schedule.
 - DRS_MAIL_REP: Replicates the NC replica from the DC identified by uuidDsa via SMTP (see [\[MS-SRPL\]](#)).
 - DRS_DISABLE_AUTO_SYNC: Disables notification-based replication of the NC replica from the DC identified by uuidDsa.
 - DRS_DISABLE_PERIODIC_SYNC: Disables periodic replication of the NC replica from the DC identified by uuidDsa.
 - DRS_USE_COMPRESSION: Replication response messages sent along this communication path should be compressed.
 - DRS_TWOWAY_SYNC: At the end of a replication cycle, replication should be triggered in the opposite direction.
- **schedule:** A [REPLTIMES](#) that corresponds to rtSchedule in [REPS_FROM](#). This contains the periodic replication schedule.
- **uuidInvocId:** A [GUID](#). Contains the invocation ID of the source DC.
- **usnVec:** A [USN_VECTOR](#) that corresponds to the usnVec in [REPS_FROM](#). This holds 0 or the usnvecTo field from the response to the last [IDL_DRSGetNCChanges](#) replication request sent to the DC identified by uuidDsa.
- **uuidTransport:** A [GUID](#) that corresponds to uuidTransportObj in [REPS_FROM](#). This is the [objectGUID](#) of the [interSiteTransport](#) object that corresponds to the transport used for communication with the DC identified by uuidDsa.
- **consecutiveFailures:** A [DWORD](#) that corresponds to cbConsecutiveFailures in [REPS_FROM](#). It is the number of consecutive failures during replication from the DC identified by uuidDsa.
- **timeLastSuccess:** A [DWORD](#) that corresponds to timeLastSuccess in [REPS_FROM](#). It is the time of the last successful replication from the DC identified by uuidDsa.

- **timeLastAttempt:** A [DWORD](#) that corresponds to timeLastAttempt in [REPS_FROM](#). It is the time of the last replication attempt with the DC identified by uuidDsa.
- **resultLastAttempt:** The result of the last replication attempt with the DC identified by uuidDsa.
- **pasData:** A [PAS_DATA](#) value that corresponds to **cbPasDataOffset** in [REPS_FROM](#). Contains the list of attributes (being added to the partial attribute set for the NC on this DC) that are being requested from the DC identified by uuidDsa as part of a PAS replication cycle.

When converting a RepsFrom to a [REPS_FROM](#), assign zeros to all unused fields of [REPS_FROM](#). If naDsa is an empty string, set cbOtherDra to 0 and cbOtherDraOffset to 0. If pasData is not present, set cbPasDataOffset to 0.

5.151 repsTo, RepsTo

The nonreplicated, multivalued attribute [repsTo](#) is an optional attribute on the root object of every NC replica. It is stored as the structure [REPS_TO](#).

The abstract type RepsTo simplifies the specification of methods that read and write the attribute [repsTo](#). Reading the attribute [repsTo](#) produces one or more RepsTo values using the following specified conversions from [REPS_TO](#). Writing a RepsTo value to the attribute [repsTo](#) stores a [REPS_TO](#) using the reverse conversion.

The type RepsTo is a tuple with the following fields:

- **naDsa:** A [NetworkAddress](#) of a DC that corresponds to cbOtherDraOffset and cbOtherDra in [REPS_TO](#).
- **uuidDsa:** A [GUID](#) that corresponds to uuidDsaObj in [REPS_TO](#). This is the DSA GUID of the target DC.
- **options:** Bit flags chosen from [DRS_OPTIONS](#) that correspond to ulReplicaFlags in [REPS_TO](#). This set contains the DRS_WRIT_REP value if this replica is writable. This set never contains any other elements.
- **resultLastAttempt:** A [DWORD](#) that corresponds to ulResultLastAttempt in [REPS_TO](#). Contains the result of the last attempt to send a replication notification to the DC identified by uuidDsa. It has a value of 0 if the last notification was sent successfully and a Windows error code otherwise.
- **consecutiveFailures:** A [DWORD](#) that corresponds to cConsecutiveFailures in [REPS_TO](#). Contains the number of unsuccessful consecutive attempts to send a replication notification to the DC identified by uuidDsa.
- **timeLastAttempt:** A [DSTIME](#) that corresponds to timeLastAttempt in [REPS_TO](#). Contains the last time an attempt was made to send a replication notification to the DC identified by uuidDsa, or 0 if no attempt was made.
- **timeLastSuccess:** A [DSTIME](#) that corresponds to timeLastSuccess in [REPS_TO](#). Contains the time the last successful replication notification to the DC identified by uuidDsa was sent, or 0 if no replication notification was successfully sent.

When converting a RepsTo to a [REPS_TO](#), assign zeros to all unused fields of [REPS_TO](#). If naDsa is an empty string, set cbOtherDra to 0 and cbOtherDraOffset to 0.

5.152 Rid

Rid is an abstract type that consists of an integer that represents the relative identifier (RID) component of a SID, as specified in [\[MS-DTYP\]](#) section 2.3 and [\[MS-SECO\]](#) section 2.1.2.

5.153 Right

Right is an abstract type that represents an access right (for example, `RIGHT_DS_WRITE_PROPERTY`) or a control access right (for example, DS-Replication-Management-Topology) on an object. The complete set of access right values is specified in [\[MS-ADTS\]](#) section 5.1.3.2, and the complete set of control access right values is specified in [\[MS-ADTS\]](#) section 5.1.3.2.1. **Note** Since access rights and control access rights are nonoverlapping sets, there is no ambiguity in having one type represent rights of both kinds.

5.154 RIGHT Values

The valid access rights used in ACLs in security descriptors are defined in [\[MS-ADTS\]](#) section 5.1.3.2.

5.155 RPCClientContexts

RPCClientContexts is an abstract type that is a sequence of tuples, one tuple per RPC context for an incoming RPC session to the DC. Each tuple contains the following fields:

- **BindingContext:** A [ULONGLONG](#) that contains a unique identifier for the context.
- **RefCount:** An integer that is used to reference count the number of references to the context.
- **IsBound:** A Boolean value that is true if [IDL_DRSUnbind](#) has not yet been called on the RPC context represented by this tuple, and false otherwise.
- **UUIDClient:** A [GUID](#) that contains the value that was passed in as the `puuidClientDsa` argument of [IDL_DRSBind](#) while establishing the context.
- **TimeLastUsed:** A [FILETIME](#) that contains the last time a session corresponding to the context was used in an RPC method call.
- **IPAddress:** A [DWORD](#) that contains the IPv4 address of the client associated with the context.
- **PID:** An integer that contains the process ID passed in by the client as the `pextClient` argument of [IDL_DRSBind](#) while establishing the context.

The global variable [dc](#) for a DC has an associated field [dc.rpcClientContexts](#), which maintains the DC's RPCClientContexts state.

5.156 RPCOutgoingContexts

RPCOutgoingContexts is an abstract type that is a sequence of tuples, one tuple per RPC context for an outgoing RPC session from the DC. Each tuple contains the following fields:

- **ServerName:** A *unicodestring* (section [3.4.3](#)) that contains the host name of the server.
- **IsBound:** A Boolean value that is true if [IDL_DRSUnbind](#) has not yet been called on the RPC context represented by this tuple, and false otherwise.
- **HandleFromCache:** A Boolean value that is true if the context handle was retrieved from the cache, and false otherwise.

- **HandleInCache:** A Boolean value that is true if the context handle is still in the cache, and false otherwise.
- **ThreadId:** An integer that contains the thread ID of the thread that is using the context.
- **BindingTimeOut:** An integer. If the context is set to be canceled, then this field contains the time-out, in minutes.
- **CreateTime:** A [DSTIME](#) value that contains the time when the context was created.
- **CallType:** An integer that indicates the type of RPC call that the DC is waiting on. See [DS_REPL_SERVER_OUTGOING_CALL](#) for possible values.

The global variable [dc](#) for a DC has an associated field [dc.rpcOutgoingContexts](#), which maintains the DC's [RPCOutgoingContexts](#) state.

5.157 sAMAccountType Values

sAMAccountType values describe information about various account type objects.

Symbolic name	Value
SAM_GROUP_OBJECT	0x10000000
SAM_NON_SECURITY_GROUP_OBJECT	0x10000001
SAM_ALIAS_OBJECT	0x20000000
SAM_NON_SECURITY_ALIAS_OBJECT	0x20000001
SAM_USER_OBJECT	0x30000000
SAM_NORMAL_USER_ACCOUNT	0x30000000
SAM_MACHINE_ACCOUNT	0x30000001
SAM_TRUST_ACCOUNT	0x30000002

5.158 SCHEMA_PREFIX_TABLE

The **SCHEMA_PREFIX_TABLE** structure defines the concrete type for a table to map [ATTRTYP](#) values to and from [OIDs](#).

```
typedef struct {
    [range(0,1048576)] DWORD PrefixCount;
    [size_is(PrefixCount)] PrefixTableEntry* pPrefixEntry;
} SCHEMA_PREFIX_TABLE;
```

PrefixCount: The count of items in the pPrefixEntry array.

pPrefixEntry: The array of [PrefixTableEntry](#) items in the table.

5.159 SchemaNC

```
procedure SchemaNC(): DSName
```

The SchemaNC procedure returns the [DSName](#) of [dc.schemaNC](#).

5.160 SchemaObj

```
procedure SchemaObj(att: ATTRTYP): DSName
```

Given the [ATTRTYP](#) att of an [attributeSchema](#) or [classSchema](#) object on this DC, the SchemaObj procedure returns the dsname of the [attributeSchema](#) or the [classSchema](#) object.

```
return select one o from children SchemaNC()  
where AttrtypFromSchemaObj(o) = att
```

5.161 ServerExtensions

```
procedure ServerExtensions(hDrs: DRS_HANDLE): DRS_EXTENSIONS_INT
```

The ServerExtensions procedure returns the server extensions presented in the [IDL_DRSBind](#) call that created hDrs. Any fields not specified by the server in the *ppextServer*[^] parameter of [IDL_DRSBind](#) are set to 0.

5.162 SID

SID is a concrete type for the Windows NT **SID** structure, as specified in [\[MS-DTYP\]](#) section **2.4.2**.

5.163 SidFromStringSid

```
procedure SidFromStringSid(stringSID: unicodestring): SID
```

The SidFromStringSid procedure converts the string representation of a SID specified in stringSID (for example, S-1-5-3) to the [SID](#) type, as specified in [\[MS-DTYP\]](#) section **2.4.2**. See [\[MS-SECO\]](#) section 2.1.2 for the string representation of a SID.

5.164 StampLessThanOrEqualUTD

```
procedure StampLessThanOrEqualUTD(  
    stamp: AttributeStamp,  
    utd: UPTODATE_VECTOR_V1_EXT) : boolean
```

Informative summary of behavior: The StampLessThanOrEqualUTD procedure is used to determine if an attribute has already replicated (or should have already replicated).

```
i: integer  
  
for i := 0 to utd.cNumCursors - 1
```

```

    if utd.rgCursors[i].uuidDsa = stamp.uuidOriginating) and
      (utd.rgCursors[i].usn >= stamp.usnOriginating) then
      return true
    endif
  endfor
return false

```

5.165 StartsWith

```

procedure StartsWith(s: unicodestring, p: unicodestring): boolean

```

The StartsWith procedure returns true if the string p is a prefix of string s, and returns false otherwise.

5.166 STATUS Codes

Windows status code. Other nonzero codes are fatal errors.

Symbolic name	Value
STATUS_ACCESS_DENIED	0xC0000022
STATUS_MORE_ENTRIES	0x00000105
STATUS_TOO_MANY_CONTEXT_IDS	0xC000015A
STATUS_INSUFFICIENT_RESOURCES	0xC000009A
STATUS_BUFFER_TOO_SMALL	0xC0000023
STATUS_INVALID_PARAMETER	0xC000000D

5.167 StringSidFromSid

```

procedure StringSidFromSid(sid: SID): unicodestring

```

The StringSidFromSid procedure converts a binary [SID](#) specified in sid to the string representation of a SID (for example, S-1-5-3). See [\[MS-SECO\]](#) section 2.1.2 for the string representation of a SID.

5.168 SubString

```

procedure SubString(
  s: unicodestring, start: integer, length: integer): unicodestring

```

The SubString procedure returns the portion of s beginning at the zero-based index start and containing length characters. If start is less than zero or greater than s.length-1, returns null. If length + start is greater than s.length, then length is treated as if it equals s.length - start.

5.169 Syntax

```
procedure Syntax(attr: ATTRTYP): AttributeSyntax
```

The Syntax procedure returns the syntax of the attribute attr.

5.170 SYNTAX_ADDRESS

The SYNTAX_ADDRESS packet is the concrete type for a sequence of bytes or Unicode characters.

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
dataLen																															
byteVal (variable)																															
...																															

dataLen: The size of the entire structure (including this field), in bytes.

byteVal: The byte or character data.

The following structure definition shows an alternative representation of this data type.

```
typedef struct {  
    DWORD dataLen;  
    union {  
        BYTE byteVal[];  
        wchar_t uVal[];  
    };  
} SYNTAX_ADDRESS;
```

5.171 SYNTAX_DISTNAME_BINARY

The SYNTAX_DISTNAME_BINARY packet is the concrete type for a combination of a [DSNAME](#) and a binary or character data buffer.

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
structLen																															
SidLen																															
Guid																															

...
...
...
Sid
...
...
...
...
...
...
...
NameLen
StringName (variable)
...
Padding (variable)
...
dataLen
byteVal (variable)
...

structLen: The length of the structure, in bytes, up to and including the field StringName.

SidLen: The number of bytes in the Sid field used to represent the object's [objectSid](#) attribute value. Zero indicates that the SYNTAX_DISTNAME_BINARY does not identify the objectSid value of the directory object.

Guid: The value of the object's [objectGUID](#) attribute specified as a GUID structure, which is defined in [\[MS-DTYP\]](#) section **2.3.2**. Zero values for all fields in the GUID structure indicate that the SYNTAX_DISTNAME_BINARY does not identify the [objectGUID](#) value of the directory object.

Sid: The value of the object's [objectSid](#) attribute, its security identifier (see [\[MS-SECO\]](#) section 2.1.2), specified as a SID structure, which is defined in [\[MS-DTYP\]](#) section **2.4.2**. The size of this field is exactly 28 bytes, regardless of the value of SidLen, which specifies how many bytes in this field are used.

NameLen: Number of characters in the StringName field, not including the null terminator, used to represent the object's [distinguishedName](#) attribute value. Zero indicates that the SYNTAX_DISTNAME_BINARY does not identify the [distinguishedName](#) value of the directory object.

StringName: the null-terminated Unicode value of the object's [distinguishedName](#) attribute, as specified in [\[MS-ADTS\]](#) section 3.1.1.1.4. This field always contains at least one character: the null terminator. Each Unicode value is encoded as 2 bytes. The byte ordering is little-endian.

Padding: The padding (bytes with value zero) to align the field dataLen at a double word boundary.

dataLen: The length of the remaining structure, including this field, in bytes.

byteVal: The array of bytes.

Note All fields have little-endian byte ordering.

The following structure definition shows an alternative representation of this data type.

```
typedef struct {
    DSNAME Name;
    SYNTAX_ADDRESS Data;
} SYNTAX_DISTNAME_BINARY;
```

5.172 systemFlags Values

The valid system flags used on directory objects are defined in [\[MS-ADTS\]](#) section 2.2.10.

5.173 UCHAR

UCHAR is a concrete type, as defined in [\[MS-DTYP\]](#) section 2.2. A UCHAR is an 8-bit, unsigned quantity.

5.174 ULARGE_INTEGER

ULARGE_INTEGER is a concrete type for a 64-bit, unsigned integer.

```
typedef struct {
    ULONGLONG QuadPart;
} ULARGE_INTEGER;
```

5.175 ULONG

ULONG is a concrete type for a 32-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.

5.176 ULONGLONG

ULONGLONG is a concrete type for a 64-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.

5.177 UNICODE_STRING

UNICODE_STRING is a concrete type for a counted, double-byte Unicode string, as specified in [\[MS-DTYP\]](#) section 2.3.

5.178 UPTODATE_CURSOR_V1

The **UPTODATE_CURSOR_V1** structure is a concrete type for the replication state relative to a given DC.

```
typedef struct {
    UUID uuidDsa;
    USN usnHighPropUpdate;
} UPTODATE_CURSOR_V1;
```

uuidDsa: The invocationId of the DC performing the update.

usnHighPropUpdate: The USN of the update on the updating DC.

A cursor *c* with *c.uuidDsa* = *x* and *c.usnHighPropUpdate* = *y* indicates a replication state that includes all changes originated by DC *x* at USN less than or equal to *y*.

5.179 UPTODATE_CURSOR_V2

The **UPTODATE_CURSOR_V2** structure defines a concrete type for the replication state relative to a given DC.

```
typedef struct {
    UUID uuidDsa;
    USN usnHighPropUpdate;
    DSTIME timeLastSyncSuccess;
} UPTODATE_CURSOR_V2;
```

uuidDsa: The invocationId of the DC performing the update.

usnHighPropUpdate: The USN of the update on the updating DC.

timeLastSyncSuccess: The time at which the last successful replication occurred from the DC identified by **uuidDsa**; for replication latency reporting only.

A cursor *c* with *c.uuidDsa* = *x* and *c.usnHighPropUpdate* = *y* indicates a replication state that includes all changes originated by DC *x* at USN less than or equal to *y*.

5.180 UPTODATE_VECTOR_V1_EXT

The **UPTODATE_VECTOR_V1_EXT** structure defines a concrete type for the replication state relative to a set of DCs.

```
typedef struct {
    DWORD dwVersion;
    DWORD dwReserved1;
    [range(0,1048576)] DWORD cNumCursors;
    DWORD dwReserved2;
    [size_is(cNumCursors)] UPTODATE_CURSOR_V1 rgCursors[];
} UPTODATE_VECTOR_V1_EXT;
```

dwVersion: The version of this structure; MUST be set to 1.

dwReserved1: Reserved; MUST be set to 0 and ignored.

cNumCursors: The count of items in the rgCursors array.

dwReserved2: Reserved; MUST be set to 0 and ignored.

rgCursors: An array of [UPTODATE_CURSOR_V1](#). The items in this field MUST be sorted in increasing order of the uuidDsa field.

5.181 UPTODATE_VECTOR_V2_EXT

The **UPTODATE_VECTOR_V2_EXT** structure defines a concrete type for the replication state relative to a set of DCs.

```
typedef struct {
    DWORD dwVersion;
    DWORD dwReserved1;
    [range(0,1048576)] DWORD cNumCursors;
    DWORD dwReserved2;
    [size_is(cNumCursors)] UPTODATE_CURSOR_V2 rgCursors[];
} UPTODATE_VECTOR_V2_EXT;
```

dwVersion: The version of this structure; MUST be set to 2.

dwReserved1: Reserved; MUST be set to 0 and ignored.

cNumCursors: The count of items in the rgCursors array.

dwReserved2: Reserved; MUST be set to 0 and ignored.

rgCursors: An array of [UPTODATE_CURSOR_V2](#). The items in this field MUST be sorted in increasing order of the uuidDsa field.

5.182 userAccountControl Bits

The userAccountControl bits are bit flags that describe various qualities of a security account.

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	A	X	X	L	X	X	X	D	N	X	I	W	S	X	X	X	X	X	X	X	X	X	X	X	X	X	P	X	X	X	X
	D			O				A	A		D	T	T													S					

X: Unused. MUST be zero and ignored.

AD (ADS_UF_ACCOUNTDISABLE): The account is disabled.

LO (ADS_UF_LOCKOUT): The account is temporarily locked out.

DA (ADS_UF_TEMP_DUPLICATE_ACCOUNT): This is an account for a user whose primary account is in another domain.

NA (ADS_UF_NORMAL_ACCOUNT): The default account type that represents a typical user.

ID (ADS_UF_INTERDOMAIN_TRUST_ACCOUNT): The account for a domain-to-domain trust.

WT (ADS_UF_WORKSTATION_ACCOUNT): The computer account for a computer that is a member of this domain.

ST (ADS_UF_SERVER_TRUST_ACCOUNT): The computer account for a DC.

PS (ADS_UF_PARTIAL_SECRETS_ACCOUNT): The computer account for an RODC.

5.183 UserNameFromNT4AccountName

```
procedure UserNameFromNT4AccountName(
    nt4AccountName: unicodestring): unicodestring
```

If nt4AccountName is a name in Windows NT 4.0 account name format, that is, two components separated by a backslash (for example, "DOMAIN\username"), then the UserNameFromNT4AccountName procedure returns the second component (the user name, or "username" in this example). If the nt4AccountName is not in this format, null is returned.

5.184 USHORT

USHORT is a concrete type for a 16-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section **2.2.57**.

5.185 USN

USN is a concrete type for the variable *usn* specified in [\[MS-ADTS\]](#) section 3.1.1.1.9 and present in the [dc](#) global variable.

This type is declared as follows:

```
typedef LONGLONG USN;
```

5.186 USN_VECTOR

The **USN_VECTOR** structure defines a concrete type for the cookie (section [1.3.2](#)) used to pass state between calls to [IDL_DRSGetNCChanges](#).

```
typedef struct {
    USN usnHighObjUpdate;
    USN usnReserved;
    USN usnHighPropUpdate;
} USN_VECTOR;
```

usnHighObjUpdate: A USN.

usnReserved: A USN.

usnHighPropUpdate: A USN.

The **USN_VECTOR** type, as shown, is used in the DRS IDL. However, only the size of **USN_VECTOR** (24 bytes) and the representation of its null value (24 zero bytes) are standardized for interoperability.

5.187 UUID

The UUID is a type that is equivalent to the [GUID](#) type.

5.188 Value

Value is an abstract type for attribute values used for abstract value representation (see section [5.15.2](#)).

5.189 VALUE_META_DATA_EXT_V1

The **VALUE_META_DATA_EXT_V1** structure defines a concrete type for the stamp of a link value.

```
typedef struct {
    DTIME timeCreated;
    PROPERTY_META_DATA_EXT MetaData;
} VALUE_META_DATA_EXT_V1;
```

timeCreated: The date and time at which the first originating update was made.

MetaData: The remainder of the stamp; has the same [PROPERTY_META_DATA_EXT](#) type as used for the stamp of an attribute.

5.190 ValueFromATTRVAL

```
procedure ValueFromATTRVAL(
    a: ATTRVAL, s: Syntax, t: PrefixTable) : Value
```

The ValueFromATTRVAL procedure converts a value of syntax *s* expressed as a concrete [ATTRVAL](#) into the abstract [Value](#) encoding, using the prefix table represented by *t*.

See section [5.15.3](#) for the specification of this procedure.

5.191 WCHAR

WCHAR is a concrete type, as specified in [\[MS-DTYP\]](#) section **2.2**. A WCHAR is a 16-bit, unsigned integer in little-endian order that is used to store a double-byte Unicode character. A WCHAR * is a pointer to a null-terminated Unicode string.

6 Security

The following sections specify security considerations for implementers.

6.1 Security Considerations for Implementers

General security considerations for this protocol are specified in section [2.2](#). Security considerations for an individual method are specified in the subsection of section [4](#) that describes the behavior of that method.

6.2 Index of Security Parameters

Security parameter	Section
SPNs for DC-to-DC authentication	Section 2.2.3.2
SPNs for client-to-DC authentication	Section 2.2.4.2

7 Appendix A: Full IDL

For ease of implementation, the full IDL is provided below, where "ms-dtyp.idl" refers to the IDL found in [\[MS-DTYP\] Appendix A](#).

```
import "ms-dtyp.idl";
[
    uuid (e3514235-4b06-11d1-ab04-00c04fc2dcd2), version(4.0),
    pointer_default (unique)
]
interface drsuapi
{

typedefet LONGLONG DSTIME;

typedefet [context_handle] void * DRS_HANDLE;

typedefet struct {
    char Data[28];
} NT4SID;

typedefet struct {
    unsigned long structLen;
    unsigned long SidLen;
    GUID Guid;
    NT4SID Sid;
    unsigned long NameLen;
    [size_is(NameLen + 1)] WCHAR StringName[];
} DSNAME;

typedefet LONGLONG USN;

typedefet struct {
    USN usnHighObjUpdate;
    USN usnReserved;
    USN usnHighPropUpdate;
} USN_VECTOR;

typedefet struct {
    UUID uuidDsa;
    USN usnHighPropUpdate;
} UPTODATE_CURSOR_V1;

typedefet struct {
    DWORD dwVersion;
    DWORD dwReserved1;
    [range(0,1048576)] DWORD cNumCursors;
    DWORD dwReserved2;
    [size_is(cNumCursors)] UPTODATE_CURSOR_V1 rgCursors[];
} UPTODATE_VECTOR_V1_EXT;

typedefet struct {
    [range(0,10000)] unsigned int length;
    [size_is(length)] BYTE *elements;
} OID_t;
```

```

typedef struct {
    unsigned long ndx;
    OID_t prefix;
} PrefixTableEntry;

typedef struct {
    [range(0,1048576)] DWORD PrefixCount;
    [size_is(PrefixCount)] PrefixTableEntry *pPrefixEntry;
} SCHEMA_PREFIX_TABLE;

typedef ULONG ATTRTYP;

typedef struct {
    DWORD dwVersion;
    DWORD dwReserved1;
    [range(1,1048576)] DWORD cAttrs;
    [size_is(cAttrs)] ATTRTYP rgPartialAttr[];
} PARTIAL_ATTR_VECTOR_V1_EXT;

typedef struct {
    [range(1,256)] unsigned long mtx_namelen;
    [size_is(mtx_namelen)] char mtx_name[];
} MTX_ADDR;

typedef struct {
    [range(0,10485760)] ULONG valLen;
    [size_is(valLen)] UCHAR *pVal;
} ATTRVAL;

typedef struct {
    [range(0, 10485760)] ULONG valCount;
    [size_is(valCount)] ATTRVAL *pAVal;
} ATTRVALBLOCK;

typedef struct {
    ATTRTYP attrTyp;
    ATTRVALBLOCK AttrVal;
} ATTR;

typedef struct {
    [range(0, 1048576)] ULONG attrCount;
    [size_is(attrCount)] ATTR *pAttr;
} ATTRBLOCK;

typedef struct {
    DSNAME *pName;
    unsigned long ulFlags;
    ATTRBLOCK AttrBlock;
} ENTINF;

typedef struct {
    DWORD dwVersion;
    DSTIME timeChanged;
    UUID uuidDsaOriginating;
    USN usnOriginating;
} PROPERTY_META_DATA_EXT;

```

```

typedef struct {
    [range(0,1048576)] DWORD cNumProps;
    [size_is(cNumProps)] PROPERTY_META_DATA_EXT rgMetaData[];
} PROPERTY_META_DATA_EXT_VECTOR;

typedef struct REPLENTINFLIST {
    struct REPLENTINFLIST * pNextEntInf;
    ENTINF Entinf;
    BOOL fIsNCPrefix;
    UUID* pParentGuid;
    PROPERTY_META_DATA_EXT_VECTOR* pMetaDataExt;
} REPLENTINFLIST;

typedef struct {
    UUID uuidDsa;
    USN usnHighPropUpdate;
    DSTIME timeLastSyncSuccess;
} UPTODATE_CURSOR_V2;

typedef struct {
    DWORD dwVersion;
    DWORD dwReserved1;
    [range(0,1048576)] DWORD cNumCursors;
    DWORD dwReserved2;
    [size_is(cNumCursors)] UPTODATE_CURSOR_V2 rgCursors[];
} UPTODATE_VECTOR_V2_EXT;

typedef struct {
    DSTIME timeCreated;
    PROPERTY_META_DATA_EXT MetaData;
} VALUE_META_DATA_EXT_V1;

typedef struct {
    DSNAME *pObject;
    ATTRTYP attrTyp;
    ATTRVAL Aval;
    BOOL fIsPresent;
    VALUE_META_DATA_EXT_V1 MetaData;
} REPLVALINF;

typedef struct {
    UCHAR rgTimes[84];
} REPLTIMES;

typedef struct {
    DWORD status;
    [string,unique] WCHAR *pDomain;
    [string,unique] WCHAR *pName;
} DS_NAME_RESULT_ITEMW, *PDS_NAME_RESULT_ITEMW;

typedef struct {
    DWORD cItems;
    [size_is(cItems)] PDS_NAME_RESULT_ITEMW rItems;
} DS_NAME_RESULTW, *PDS_NAME_RESULTW;

typedef struct {
    [string,unique] WCHAR *NetbiosName;
    [string,unique] WCHAR *DnsHostName;
}

```

```

        [string,unique] WCHAR *SiteName;
        [string,unique] WCHAR *ComputerObjectName;
        [string,unique] WCHAR *ServerObjectName;
        BOOL fIsPdc;
        BOOL fDsEnabled;
    } DS_DOMAIN_CONTROLLER_INFO_1W;

typedef struct {
    [string,unique] WCHAR *NetbiosName;
    [string,unique] WCHAR *DnsHostName;
    [string,unique] WCHAR *SiteName;
    [string,unique] WCHAR *SiteObjectName;
    [string,unique] WCHAR *ComputerObjectName;
    [string,unique] WCHAR *ServerObjectName;
    [string,unique] WCHAR *NtdsDsaObjectName;
    BOOL fIsPdc;
    BOOL fDsEnabled;
    BOOL fIsGc;
    GUID SiteObjectGuid;
    GUID ComputerObjectGuid;
    GUID ServerObjectGuid;
    GUID NtdsDsaObjectGuid;
} DS_DOMAIN_CONTROLLER_INFO_2W;

typedef struct {
    [string, unique] WCHAR* NetbiosName;
    [string, unique] WCHAR* DnsHostName;
    [string, unique] WCHAR* SiteName;
    [string, unique] WCHAR* SiteObjectName;
    [string, unique] WCHAR* ComputerObjectName;
    [string, unique] WCHAR* ServerObjectName;
    [string, unique] WCHAR* NtdsDsaObjectName;
    BOOL fIsPdc;
    BOOL fDsEnabled;
    BOOL fIsGc;
    BOOL fIsRodc;
    GUID SiteObjectGuid;
    GUID ComputerObjectGuid;
    GUID ServerObjectGuid;
    GUID NtdsDsaObjectGuid;
} DS_DOMAIN_CONTROLLER_INFO_3W;

typedef struct {
    DWORD IPAddress;
    DWORD NotificationCount;
    DWORD secTimeConnected;
    DWORD Flags;
    DWORD TotalRequests;
    DWORD Reserved1;
    [string,unique] WCHAR *UserName;
} DS_DOMAIN_CONTROLLER_INFO_FFFFFFFW;

typedef struct ENTINFLIST {
    struct ENTINFLIST *pNextEntInf;
    ENTINF Entinf;
} ENTINFLIST;

typedef struct {

```

```

        DWORD dsid;
        DWORD extendedErr;
        DWORD extendedData;
        USHORT problem;
        ATTRTYP type;
        BOOL valReturned;
        ATTRVAL Val;
    } INFORMPROB_DRS_WIRE_V1;

typedef struct _PROBLEMLIST_DRS_WIRE_V1 {
    struct _PROBLEMLIST_DRS_WIRE_V1 *pNextProblem;
    INFORMPROB_DRS_WIRE_V1 intprob;
} PROBLEMLIST_DRS_WIRE_V1;

typedef struct {
    DSNAME *pObject;
    ULONG count;
    PROBLEMLIST_DRS_WIRE_V1 FirstProblem;
} ATRERR_DRS_WIRE_V1;

typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
    DSNAME *pMatched;
} NAMERR_DRS_WIRE_V1;

typedef struct {
    UCHAR nameRes;
    UCHAR unusedPad;
    USHORT nextRDN;
} NAMERESOP_DRS_WIRE_V1;

typedef struct _DSA_ADDRESS_LIST_DRS_WIRE_V1 {
    struct _DSA_ADDRESS_LIST_DRS_WIRE_V1 *pNextAddress;
    UNICODE_STRING *pAddress;
} DSA_ADDRESS_LIST_DRS_WIRE_V1;

typedef struct _CONTREF_DRS_WIRE_V1 {
    DSNAME *pTarget;
    NAMERESOP_DRS_WIRE_V1 OpState;
    USHORT aliasRDN;
    USHORT RDNsInternal;
    USHORT refType;
    USHORT count;
    DSA_ADDRESS_LIST_DRS_WIRE_V1 *pDAL;
    struct _CONTREF_DRS_WIRE_V1 *pNextContRef;
    BOOL bNewChoice;
    UCHAR choice;
} CONTREF_DRS_WIRE_V1;

typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    CONTREF_DRS_WIRE_V1 Refer;
} REFERR_DRS_WIRE_V1;

```

```

typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
} SECERR_DRS_WIRE_V1;

typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
} SVCERR_DRS_WIRE_V1;

typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
} UPDERR_DRS_WIRE_V1;

typedef struct {
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
} SYSERR_DRS_WIRE_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] ATRERR_DRS_WIRE_V1 AtrErr;
    [case(2)] NAMERR_DRS_WIRE_V1 NamErr;
    [case(3)] REFERR_DRS_WIRE_V1 RefErr;
    [case(4)] SECERR_DRS_WIRE_V1 SecErr;
    [case(5)] SVCERR_DRS_WIRE_V1 SvcErr;
    [case(6)] UPDERR_DRS_WIRE_V1 UpdErr;
    [case(7)] SYSERR_DRS_WIRE_V1 SysErr;
} DIRERR_DRS_WIRE_V1;

typedef struct {
    [string] LPWSTR pszNamingContext;
    [string] LPWSTR pszSourceDsaDN;
    [string] LPWSTR pszSourceDsaAddress;
    [string] LPWSTR pszAsyncIntersiteTransportDN;
    DWORD dwReplicaFlags;
    DWORD dwReserved;
    UUID uuidNamingContextObjGuid;
    UUID uuidSourceDsaObjGuid;
    UUID uuidSourceDsaInvocationID;
    UUID uuidAsyncIntersiteTransportObjGuid;
    USN usnLastObjChangeSynced;
    USN usnAttributeFilter;
    FILETIME ftimeLastSyncSuccess;
    FILETIME ftimeLastSyncAttempt;
    DWORD dwLastSyncResult;
    DWORD cNumConsecutiveSyncFailures;
} DS_REPL_NEIGHBORW;

```

```

typedef struct {
    DWORD cNumNeighbors;
    DWORD dwReserved;
    [size_is(cNumNeighbors)] DS_REPL_NEIGHBORW rgNeighbor[];
} DS_REPL_NEIGHBORSW;

typedef struct {
    UUID uuidSourceDsaInvocationID;
    USN usnAttributeFilter;
} DS_REPL_CURSOR;

typedef struct {
    DWORD cNumCursors;
    DWORD dwReserved;
    [size_is(cNumCursors)] DS_REPL_CURSOR rgCursor[];
} DS_REPL_CURSORS;

typedef struct {
    [string] LPWSTR pszAttributeName;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
} DS_REPL_ATTR_META_DATA;

typedef struct {
    [string] LPWSTR pszDsaDN;
    UUID uuidDsaObjGuid;
    FILETIME ftimeFirstFailure;
    DWORD cNumFailures;
    DWORD dwLastResult;
} DS_REPL_KCC_DSA_FAILUREW;

typedef struct {
    DWORD cNumEntries;
    DWORD dwReserved;
    [size_is(cNumEntries)] DS_REPL_KCC_DSA_FAILUREW rgDsaFailure[];
} DS_REPL_KCC_DSA_FAILURESW;

typedef struct {
    DWORD cNumEntries;
    DWORD dwReserved;
    [size_is(cNumEntries)] DS_REPL_ATTR_META_DATA rgMetaData[];
} DS_REPL_OBJ_META_DATA;

typedef enum {
    DS_REPL_OP_TYPE_SYNC = 0,
    DS_REPL_OP_TYPE_ADD,
    DS_REPL_OP_TYPE_DELETE,
    DS_REPL_OP_TYPE_MODIFY,
    DS_REPL_OP_TYPE_UPDATE_REFS
} DS_REPL_OP_TYPE;

typedef struct {
    FILETIME ftimeEnqueued;
    ULONG ulSerialNumber;
    ULONG ulPriority;
}

```

```

        DS_REPL_OP_TYPE OpType;
        ULONG ulOptions;
        [string] LPWSTR pszNamingContext;
        [string] LPWSTR pszDsaDN;
        [string] LPWSTR pszDsaAddress;
        UUID uuidNamingContextObjGuid;
        UUID uuidDsaObjGuid;
    } DS_REPL_OPW;

typedef struct {
    FILETIME ftimeCurrentOpStarted;
    DWORD cNumPendingOps;
    [size_is(cNumPendingOps)] DS_REPL_OPW rgPendingOp[];
} DS_REPL_PENDING_OPSW;

typedef struct {
    [string] LPWSTR pszAttributeName;
    [string] LPWSTR pszObjectDn;
    DWORD cbData;
    [size_is(cbData), ptr] BYTE *pbData;
    FILETIME ftimeDeleted;
    FILETIME ftimeCreated;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
} DS_REPL_VALUE_META_DATA;

typedef struct {
    DWORD cNumEntries;
    DWORD dwEnumerationContext;
    [size_is(cNumEntries)] DS_REPL_VALUE_META_DATA rgMetaData[];
} DS_REPL_ATTR_VALUE_META_DATA;

typedef struct {
    UUID uuidSourceDsaInvocationID;
    USN usnAttributeFilter;
    FILETIME ftimeLastSyncSuccess;
} DS_REPL_CURSOR_2;

typedef struct {
    DWORD cNumCursors;
    DWORD dwEnumerationContext;
    [size_is(cNumCursors)] DS_REPL_CURSOR_2 rgCursor[];
} DS_REPL_CURSORS_2;

typedef struct {
    UUID uuidSourceDsaInvocationID;
    USN usnAttributeFilter;
    FILETIME ftimeLastSyncSuccess;
    [string] LPWSTR pszSourceDsaDN;
} DS_REPL_CURSOR_3W;

typedef struct {
    DWORD cNumCursors;
    DWORD dwEnumerationContext;
    [size_is(cNumCursors)] DS_REPL_CURSOR_3W rgCursor[];
}

```



```

} DS_REPL_CURSORS_3W;

typedef struct {
    [string] LPWSTR pszAttributeName;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
    [string] LPWSTR pszLastOriginatingDsaDN;
} DS_REPL_ATTR_META_DATA_2;

typedef struct {
    DWORD cNumEntries;
    DWORD dwReserved;
    [size_is(cNumEntries)] DS_REPL_ATTR_META_DATA_2 rgMetaData[];
} DS_REPL_OBJ_META_DATA_2;

typedef struct {
    [string] LPWSTR pszAttributeName;
    [string] LPWSTR pszObjectDn;
    DWORD cbData;
    [size_is(cbData), ptr] BYTE *pbData;
    FILETIME ftimeDeleted;
    FILETIME ftimeCreated;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
    [string] LPWSTR pszLastOriginatingDsaDN;
} DS_REPL_VALUE_META_DATA_2;

typedef struct {
    DWORD cNumEntries;
    DWORD dwEnumerationContext;
    [size_is(cNumEntries)] DS_REPL_VALUE_META_DATA_2 rgMetaData[];
} DS_REPL_ATTR_VALUE_META_DATA_2;

typedef struct {
    [range(1,10000)] DWORD cb;
    [size_is(cb)] BYTE rgb[];
} DRS_EXTENSIONS;

typedef struct {
    UUID uuidDsaObjDest;
    UUID uuidInvocIdSrc;
    [ref] DSNAME *pNC;
    USN_VECTOR usnvecFrom;
    [unique] UPTODATE_VECTOR_V1_EXT *pUpToDateVecDestV1;
    [unique] PARTIAL_ATTR_VECTOR_V1_EXT *pPartialAttrVecDestV1;
    SCHEMA_PREFIX_TABLE PrefixTableDest;
    ULONG ulFlags;
    ULONG cMaxObjects;
    ULONG cMaxBytes;
    ULONG ulExtendedOp;
} DRS_MSG_GETCHGREQ_V3;

```

```

typedef struct {
    UUID uuidTransportObj;
    [ref] MTX_ADDR *pmtxReturnAddress;
    DRS_MSG_GETCHGREQ_V3 V3;
} DRS_MSG_GETCHGREQ_V4;

typedef struct {
    UUID uuidTransportObj;
    [ref] MTX_ADDR *pmtxReturnAddress;
    DRS_MSG_GETCHGREQ_V3 V3;
    [unique] PARTIAL_ATTR_VECTOR_V1_EXT *pPartialAttrSet;
    [unique] PARTIAL_ATTR_VECTOR_V1_EXT *pPartialAttrSetEx;
    SCHEMA_PREFIX_TABLE PrefixTableDest;
} DRS_MSG_GETCHGREQ_V7;

typedef struct {
    UUID uuidDsaObjSrc;
    UUID uuidInvocIdSrc;
    [ref] DSNAME *pNC;
    USN_VECTOR usnvecFrom;
    USN_VECTOR usnvecTo;
    [unique] UPTODATE_VECTOR_V1_EXT *pUpToDateVecSrcV1;
    SCHEMA_PREFIX_TABLE PrefixTableSrc;
    ULONG ulExtendedRet;
    ULONG cNumObjects;
    ULONG cNumBytes;
    [unique] REPLENTINFLIST* pObjects;
    BOOL fMoreData;
} DRS_MSG_GETCHGREPLY_V1;

typedef struct {
    UUID uuidDsaObjSrc;
    UUID uuidInvocIdSrc;
    [ref] DSNAME *pNC;
    USN_VECTOR usnvecFrom;
    USN_VECTOR usnvecTo;
    [unique] UPTODATE_VECTOR_V2_EXT *pUpToDateVecSrc;
    SCHEMA_PREFIX_TABLE PrefixTableSrc;
    ULONG ulExtendedRet;
    ULONG cNumObjects;
    ULONG cNumBytes;
    [unique] REPLENTINFLIST *pObjects;
    BOOL fMoreData;
    ULONG cNumNcSizeObjects;
    ULONG cNumNcSizeValues;
    [range(0,1048576)] DWORD cNumValues;
    [size_is(cNumValues)] REPLVALINF *rgValues;
    DWORD dwDRSError;
} DRS_MSG_GETCHGREPLY_V6;

typedef struct {
    DWORD cbUncompressedSize;
    [range(1, 10485760)] DWORD cbCompressedSize;
    [size_is(cbCompressedSize)] BYTE *pbCompressedData;
} DRS_COMPRESSED_BLOB;

typedef struct {
    UUID uuidDsaObjDest;

```

```

        UUID uuidInvocIdSrc;
        [ref] DSNAME *pNC;
        USN_VECTOR usnvecFrom;
        [unique] UPTODATE_VECTOR_V1_EXT *pUpToDateVecDestV1;
        ULONG ulFlags;
        ULONG cMaxObjects;
        ULONG cMaxBytes;
        ULONG ulExtendedOp;
        ULARGE_INTEGER liFsmoInfo;
    } DRS_MSG_GETCHGREQ_V5;

typedef struct {
    UUID uuidDsaObjDest;
    UUID uuidInvocIdSrc;
    [ref] DSNAME *pNC;
    USN_VECTOR usnvecFrom;
    [unique] UPTODATE_VECTOR_V1_EXT *pUpToDateVecDest;
    ULONG ulFlags;
    ULONG cMaxObjects;
    ULONG cMaxBytes;
    ULONG ulExtendedOp;
    ULARGE_INTEGER liFsmoInfo;
    [unique] PARTIAL_ATTR_VECTOR_V1_EXT *pPartialAttrSet;
    [unique] PARTIAL_ATTR_VECTOR_V1_EXT *pPartialAttrSetEx;
    SCHEMA_PREFIX_TABLE PrefixTableDest;
} DRS_MSG_GETCHGREQ_V8;

typedef [switch_type(DWORD)] union {
    [case(4)] DRS_MSG_GETCHGREQ_V4 V4;
    [case(5)] DRS_MSG_GETCHGREQ_V5 V5;
    [case(7)] DRS_MSG_GETCHGREQ_V7 V7;
    [case(8)] DRS_MSG_GETCHGREQ_V8 V8;
} DRS_MSG_GETCHGREQ;

typedef struct {
    DRS_COMPRESSED_BLOB CompressedV1;
} DRS_MSG_GETCHGREPLY_V2;

typedef enum {
    DRS_COMP_ALG_NONE = 0,
    DRS_COMP_ALG_MSZIP = 2,
    DRS_COMP_ALG_WIN2K3 = 3
} DRS_COMP_ALG_TYPE;

typedef struct {
    DWORD dwCompressedVersion;
    DRS_COMP_ALG_TYPE CompressionAlg;
    DRS_COMPRESSED_BLOB CompressedAny;
} DRS_MSG_GETCHGREPLY_V7;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_GETCHGREPLY_V1 V1;
    [case(2)] DRS_MSG_GETCHGREPLY_V2 V2;
    [case(6)] DRS_MSG_GETCHGREPLY_V6 V6;
    [case(7)] DRS_MSG_GETCHGREPLY_V7 V7;
} DRS_MSG_GETCHGREPLY;

typedef struct {

```

```

        [ref] DSNAME *pNC;
        UUID uuidDsaSrc;
        [unique] [string] char *pszDsaSrc;
        ULONG ulOptions;
    } DRS_MSG_REPSYNC_V1;

typedef [switch_type(DWORD)] union
{
    [case(1)] DRS_MSG_REPSYNC_V1 V1;
} DRS_MSG_REPSYNC;

typedef struct {
    [ref] DSNAME *pNC;
    [ref] [string] char *pszDsaDest;
    UUID uuidDsaObjDest;
    ULONG ulOptions;
} DRS_MSG_UPDREFS_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_UPDREFS_V1 V1;
} DRS_MSG_UPDREFS;

typedef struct {
    [ref] DSNAME *pNC;
    [ref] [string] char *pszDsaSrc;
    REPLTIMES rtSchedule;
    ULONG ulOptions;
} DRS_MSG_REPADD_V1;

typedef struct {
    [ref] DSNAME *pNC;
    [unique] DSNAME *pSourceDsaDN;
    [unique] DSNAME *pTransportDN;
    [ref] [string] char *pszSourceDsaAddress;
    REPLTIMES rtSchedule;
    ULONG ulOptions;
} DRS_MSG_REPADD_V2;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_REPADD_V1 V1;
    [case(2)] DRS_MSG_REPADD_V2 V2;
} DRS_MSG_REPADD;

typedef struct {
    [ref] DSNAME *pNC;
    [string] char *pszDsaSrc;
    ULONG ulOptions;
} DRS_MSG_REPDEL_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_REPDEL_V1 V1;
} DRS_MSG_REPDEL;

typedef struct {
    [ref] DSNAME *pNC;
    UUID uuidSourceDRA;
    [unique, string] char *pszSourceDRA;
    REPLTIMES rtSchedule;

```

```

        ULONG ulReplicaFlags;
        ULONG ulModifyFields;
        ULONG ulOptions;
    } DRS_MSG_REPMOD_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_REPMOD_V1 V1;
} DRS_MSG_REPMOD;

typedef struct {
    DWORD dwFlags;
    [range(1,10000)] DWORD cNames;
    [size_is(cNames)] DSNAME **rpNames;
    ATTRBLOCK RequiredAttrs;
    SCHEMA_PREFIX_TABLE PrefixTable;
} DRS_MSG_VERIFYREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_VERIFYREQ_V1 V1;
} DRS_MSG_VERIFYREQ;

typedef struct {
    DWORD error;
    [range(0,10000)] DWORD cNames;
    [size_is(cNames)] ENTINF *rpEntInf;
    SCHEMA_PREFIX_TABLE PrefixTable;
} DRS_MSG_VERIFYREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_VERIFYREPLY_V1 V1;
} DRS_MSG_VERIFYREPLY;

typedef enum {
    RevMembGetGroupsForUser=1,
    RevMembGetAliasMembership,
    RevMembGetAccountGroups,
    RevMembGetResourceGroups,
    RevMembGetUniversalGroups,
    GroupMembersTransitive,
    RevMembGlobalGroupsNonTransitive
} REVERSE_MEMBERSHIP_OPERATION_TYPE;

typedef struct {
    [range(1,10000)] ULONG cDsNames;
    [size_is(cDsNames)] DSNAME **ppDsNames;
    DWORD dwFlags;
    [range(1,7)] REVERSE_MEMBERSHIP_OPERATION_TYPE OperationType;
    DSNAME *pLimitingDomain;
} DRS_MSG_REVMEMB_REQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_REVMEMB_REQ_V1 V1;
} DRS_MSG_REVMEMB_REQ;

typedef struct {
    ULONG errCode;
    [range(0,10000)] ULONG cDsNames;
    [range(0,10000)] ULONG cSidHistory;

```

```

        [size_is(cDsNames)] DSNAME **ppDsNames;
        [size_is(cDsNames)] DWORD *pAttributes;
        [size_is(cSidHistory)] NT4SID **ppSidHistory;
    } DRS_MSG_REVMEMB_REPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_REVMEMB_REPLY_V1 V1;
} DRS_MSG_REVMEMB_REPLY;

typedef struct {
    char *pSourceDSA;
    ENTINF *pObject;
    UUID *pParentUUID;
    SCHEMA_PREFIX_TABLE PrefixTable;
    ULONG ulFlags;
} DRS_MSG_MOVEREQ_V1;

typedef struct {
    [range(0,10000)] unsigned long cbBuffer;
    unsigned long BufferType;
    [size_is(cbBuffer)] BYTE *pvBuffer;
} DRS_SecBuffer;

typedef struct {
    unsigned long ulVersion;
    [range(0,10000)] unsigned long cBuffers;
    [size_is(cBuffers)] DRS_SecBuffer *Buffers;
} DRS_SecBufferDesc;

typedef struct {
    DSNAME *pSrcDSA;
    ENTINF *pSrcObject;
    DSNAME *pDstName;
    DSNAME *pExpectedTargetNC;
    DRS_SecBufferDesc *pClientCreds;
    SCHEMA_PREFIX_TABLE PrefixTable;
    ULONG ulFlags;
} DRS_MSG_MOVEREQ_V2;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_MOVEREQ_V1 V1;
    [case(2)] DRS_MSG_MOVEREQ_V2 V2;
} DRS_MSG_MOVEREQ;

typedef struct {
    ENTINF **ppResult;
    SCHEMA_PREFIX_TABLE PrefixTable;
    ULONG *pError;
} DRS_MSG_MOVEREPLY_V1;

typedef struct {
    ULONG win32Error;
    [unique] DSNAME *pAddedName;
} DRS_MSG_MOVEREPLY_V2;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_MOVEREPLY_V1 V1;
    [case(2)] DRS_MSG_MOVEREPLY_V2 V2;
}

```

```

} DRS_MSG_MOVEREPLY;

typedef struct {
    ULONG CodePage;
    ULONG LocaleId;
    DWORD dwFlags;
    DWORD formatOffered;
    DWORD formatDesired;
    [range(1,10000)] DWORD cNames;
    [string, size_is(cNames)] WCHAR **rpNames;
} DRS_MSG_CRACKREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_CRACKREQ_V1 V1;
} DRS_MSG_CRACKREQ;

typedef struct {
    DS_NAME_RESULTW *pResult;
} DRS_MSG_CRACKREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_CRACKREPLY_V1 V1;
} DRS_MSG_CRACKREPLY;

typedef struct {
    DWORD dwFlags;
    DWORD PreferredMaximumLength;
    [range(0,10485760)] DWORD cbRestart;
    [size_is(cbRestart)] BYTE *pRestart;
} DRS_MSG_NT4_CHGLOG_REQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_NT4_CHGLOG_REQ_V1 V1;
} DRS_MSG_NT4_CHGLOG_REQ;

typedef struct {
    LARGE_INTEGER SamSerialNumber;
    LARGE_INTEGER SamCreationTime;
    LARGE_INTEGER BuiltinSerialNumber;
    LARGE_INTEGER BuiltinCreationTime;
    LARGE_INTEGER LsaSerialNumber;
    LARGE_INTEGER LsaCreationTime;
} NT4_REPLICATION_STATE;

typedef struct {
    [range(0,10485760)] DWORD cbRestart;
    [range(0,10485760)] DWORD cbLog;
    NT4_REPLICATION_STATE ReplicationState;
    DWORD ActualNtStatus;
    [size_is(cbRestart)] BYTE *pRestart;
    [size_is(cbLog)] BYTE *pLog;
} DRS_MSG_NT4_CHGLOG_REPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_NT4_CHGLOG_REPLY_V1 V1;
} DRS_MSG_NT4_CHGLOG_REPLY;

typedef struct {

```

```

    DWORD operation;
    DWORD flags;
    [string] const WCHAR *pwszAccount;
    [range(0,10000)] DWORD cSPN;
    [string, size_is(cSPN)] const WCHAR **rpwszSPN;
} DRS_MSG_SPNREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_SPNREQ_V1 V1;
} DRS_MSG_SPNREQ;

typedef struct {
    DWORD retVal;
} DRS_MSG_SPNREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_SPNREPLY_V1 V1;
} DRS_MSG_SPNREPLY;

typedef struct {
    [string] LPWSTR ServerDN;
    [string] LPWSTR DomainDN;
    BOOL fCommit;
} DRS_MSG_RMSVRREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_RMSVRREQ_V1 V1;
} DRS_MSG_RMSVRREQ;

typedef struct {
    BOOL fLastDcInDomain;
} DRS_MSG_RMSVRREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_RMSVRREPLY_V1 V1;
} DRS_MSG_RMSVRREPLY;

typedef struct {
    [string] LPWSTR DomainDN;
} DRS_MSG_RMDMNREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_RMDMNREQ_V1 V1;
} DRS_MSG_RMDMNREQ;

typedef struct {
    DWORD Reserved;
} DRS_MSG_RMDMNREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_RMDMNREPLY_V1 V1;
} DRS_MSG_RMDMNREPLY;

typedef struct {
    [string] WCHAR *Domain;
    DWORD InfoLevel;
} DRS_MSG_DCINFOREQ_V1;

```



```

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_DCINFOREQ_V1 V1;
} DRS_MSG_DCINFOREQ, *PDRS_MSG_DCINFOREQ;

typedef struct {
    [range(0,10000)] DWORD cItems;
    [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_1W *rItems;
} DRS_MSG_DCINFOREPLY_V1;

typedef struct {
    [range(0,10000)] DWORD cItems;
    [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_2W *rItems;
} DRS_MSG_DCINFOREPLY_V2;

typedef struct {
    [range(0,10000)] DWORD cItems;
    [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_3W* rItems;
} DRS_MSG_DCINFOREPLY_V3;

typedef struct {
    [range(0,10000)] DWORD cItems;
    [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_FFFFFFFFW *rItems;
} DRS_MSG_DCINFOREPLY_VFFFFFFFF;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_DCINFOREPLY_V1 V1;
    [case(2)] DRS_MSG_DCINFOREPLY_V2 V2;
    [case(3)] DRS_MSG_DCINFOREPLY_V3 V3;
    [case(0xFFFFFFFF)] DRS_MSG_DCINFOREPLY_VFFFFFFFF VFFFFFFFF;
} DRS_MSG_DCINFOREPLY;

typedef struct {
    [ref] DSNAME *pObject;
    ATTRBLOCK AttrBlock;
} DRS_MSG_ADDENTRYREQ_V1;

typedef struct {
    ENTINFLIST EntInfList;
} DRS_MSG_ADDENTRYREQ_V2;

typedef struct {
    ENTINFLIST EntInfList;
    DRS_SecBufferDesc *pClientCreds;
} DRS_MSG_ADDENTRYREQ_V3;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_ADDENTRYREQ_V1 V1;
    [case(2)] DRS_MSG_ADDENTRYREQ_V2 V2;
    [case(3)] DRS_MSG_ADDENTRYREQ_V3 V3;
} DRS_MSG_ADDENTRYREQ;

typedef struct {
    GUID Guid;
    NT4SID Sid;
    DWORD errCode;
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
}

```

```

        USHORT problem;
    } DRS_MSG_ADDENTRYREPLY_V1;

typedef struct {
    GUID objGuid;
    NT4SID objSid;
} ADDENTRY_REPLY_INFO;

typedef struct {
    [unique] DSNAME *pErrorObject;
    DWORD errCode;
    DWORD dsid;
    DWORD extendedErr;
    DWORD extendedData;
    USHORT problem;
    [range(0,10000)] ULONG cObjectsAdded;
    [size_is(cObjectsAdded)] ADDENTRY_REPLY_INFO *infoList;
} DRS_MSG_ADDENTRYREPLY_V2;

typedef struct {
    DWORD dwRepError;
    DWORD errCode;
    [switch_is(errCode)] DIRERR_DRS_WIRE_V1 *pErrInfo;
} DRS_ERROR_DATA_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_ERROR_DATA_V1 V1;
} DRS_ERROR_DATA;

typedef struct {
    DSNAME *pdsErrObject;
    DWORD dwErrVer;
    [switch_is(dwErrVer)] DRS_ERROR_DATA *pErrData;
    [range(0,10000)] ULONG cObjectsAdded;
    [size_is(cObjectsAdded)] ADDENTRY_REPLY_INFO *infoList;
} DRS_MSG_ADDENTRYREPLY_V3;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_ADDENTRYREPLY_V1 V1;
    [case(2)] DRS_MSG_ADDENTRYREPLY_V2 V2;
    [case(3)] DRS_MSG_ADDENTRYREPLY_V3 V3;
} DRS_MSG_ADDENTRYREPLY;

typedef struct {
    DWORD dwTaskID;
    DWORD dwFlags;
} DRS_MSG_KCC_EXECUTE_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_KCC_EXECUTE_V1 V1;
} DRS_MSG_KCC_EXECUTE;

typedef struct {
    ULONGLONG hCtx;
    LONG lReferenceCount;
    BOOL fIsBound;
    UUID uuidClient;
    DSTMIME timeLastUsed;

```

```

        ULONG IPAddr;
        int pid;
    } DS_REPL_CLIENT_CONTEXT;

typedef struct {
    [range(0,10000)] DWORD cNumContexts;
    DWORD dwReserved;
    [size_is(cNumContexts)] DS_REPL_CLIENT_CONTEXT rgContext[];
} DS_REPL_CLIENT_CONTEXTS;

typedef struct {
    [string] LPWSTR pszServerName;
    BOOL fIsHandleBound;
    BOOL fIsHandleFromCache;
    BOOL fIsHandleInCache;
    DWORD dwThreadId;
    DWORD dwBindingTimeoutMins;
    DSTIME dstimeCreated;
    DWORD dwCallType;
} DS_REPL_SERVER_OUTGOING_CALL;

typedef struct {
    [range(0, 256)] DWORD cNumCalls;
    DWORD dwReserved;
    [size_is(cNumCalls)] DS_REPL_SERVER_OUTGOING_CALL rgCall[];
} DS_REPL_SERVER_OUTGOING_CALLS;

typedef struct {
    DWORD InfoType;
    [string] LPWSTR pszObjectDN;
    UUID uuidSourceDsaObjGuid;
} DRS_MSG_GETREPLINFO_REQ_V1;

typedef struct {
    DWORD InfoType;
    [string] LPWSTR pszObjectDN;
    UUID uuidSourceDsaObjGuid;
    DWORD ulFlags;
    [string] LPWSTR pszAttributeName;
    [string] LPWSTR pszValueDN;
    DWORD dwEnumerationContext;
} DRS_MSG_GETREPLINFO_REQ_V2;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_GETREPLINFO_REQ_V1 V1;
    [case(2)] DRS_MSG_GETREPLINFO_REQ_V2 V2;
} DRS_MSG_GETREPLINFO_REQ;

typedef [switch_type(DWORD)] union {
    [case(0)] DS_REPL_NEIGHBORSW *pNeighbors;
    [case(1)] DS_REPL_CURSORS *pCursors;
    [case(2)] DS_REPL_OBJ_META_DATA *pObjMetaData;
    [case(3)] DS_REPL_KCC_DSA_FAILURESW *pConnectFailures;
    [case(4)] DS_REPL_KCC_DSA_FAILURESW *pLinkFailures;
    [case(5)] DS_REPL_PENDING_OPSW *pPendingOps;
    [case(6)] DS_REPL_ATTR_VALUE_META_DATA *pAttrValueMetaData;
    [case(7)] DS_REPL_CURSORS_2 *pCursors2;
    [case(8)] DS_REPL_CURSORS_3W *pCursors3;
}

```

```

    [case(9)] DS_REPL_OBJ_META_DATA_2 *pObjMetaData2;
    [case(10)] DS_REPL_ATTR_VALUE_META_DATA_2 *pAttrValueMetaData2;
    [case(0xFFFFFFFF)]
        DS_REPL_SERVER_OUTGOING_CALLS *pServerOutgoingCalls;
    [case(0xFFFFFFFFB)] UPTODATE_VECTOR_V1_EXT *pUpToDateVec;
    [case(0xFFFFFFFFC)] DS_REPL_CLIENT_CONTEXTS *pClientContexts;
    [case(0xFFFFFFFFE)] DS_REPL_NEIGHBORSW *pRepsTo;
} DRS_MSG_GETREPLINFO_REPLY;

typedef struct {
    DWORD Flags;
    [string] WCHAR *SrcDomain;
    [string] WCHAR *SrcPrincipal;
    [string, ptr] WCHAR *SrcDomainController;
    [range(0,256)] DWORD SrcCredsUserLength;
    [size_is(SrcCredsUserLength)] WCHAR *SrcCredsUser;
    [range(0,256)] DWORD SrcCredsDomainLength;
    [size_is(SrcCredsDomainLength)] WCHAR *SrcCredsDomain;
    [range(0,256)] DWORD SrcCredsPasswordLength;
    [size_is(SrcCredsPasswordLength)] WCHAR *SrcCredsPassword;
    [string] WCHAR *DstDomain;
    [string] WCHAR *DstPrincipal;
} DRS_MSG_ADDSIDREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_ADDSIDREQ_V1 V1;
} DRS_MSG_ADDSIDREQ;

typedef struct {
    DWORD dwWin32Error;
} DRS_MSG_ADDSIDREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_ADDSIDREPLY_V1 V1;
} DRS_MSG_ADDSIDREPLY;

typedef struct {
    [range(1, 10000)] ULONG Count;
    [size_is(Count)] DRS_MSG_REVMEMB_REQ_V1 *Requests;
} DRS_MSG_GETMEMBERSHIPS2_REQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_GETMEMBERSHIPS2_REQ_V1 V1;
} DRS_MSG_GETMEMBERSHIPS2_REQ;

typedef struct {
    [range(0, 10000)] ULONG Count;
    [size_is(Count)] DRS_MSG_REVMEMB_REPLY_V1 *Replies;
} DRS_MSG_GETMEMBERSHIPS2_REPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_GETMEMBERSHIPS2_REPLY_V1 V1;
} DRS_MSG_GETMEMBERSHIPS2_REPLY;

typedef struct {
    [ref] DSNAME *pNC;
    UUID uuidDsaSrc;
    ULONG ulOptions;
}

```

```

} DRS_MSG_REPVERIFYOBJ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_REPVERIFYOBJ_V1 V1;
} DRS_MSG_REPVERIFYOBJ;

typedef struct {
    UUID guidStart;
    DWORD cGuids;
    DSNAME *pNC;
    UPTODATE_VECTOR_V1_EXT *pUpToDateVecCommonV1;
    UCHAR Md5Digest[16];
} DRS_MSG_EXISTREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_EXISTREQ_V1 V1;
} DRS_MSG_EXISTREQ;

typedef struct {
    DWORD dwStatusFlags;
    [range(0,10485760)] DWORD cNumGuids;
    [size_is(cNumGuids)] UUID *rgGuids;
} DRS_MSG_EXISTREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_EXISTREPLY_V1 V1;
} DRS_MSG_EXISTREPLY;

typedef struct {
    [string] const WCHAR *pwszFromSite;
    [range(1,10000)] DWORD cToSites;
    [string, size_is(cToSites)] WCHAR **rgszToSites;
    DWORD dwFlags;
} DRS_MSG_QUERYsitesREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_QUERYsitesREQ_V1 V1;
} DRS_MSG_QUERYsitesREQ;

typedef struct {
    DWORD dwErrorCode;
    DWORD dwCost;
} DRS_MSG_QUERYsitesREPLYELEMENT_V1;

typedef struct {
    [range(0,10000)] DWORD cToSites;
    [size_is(cToSites)] DRS_MSG_QUERYsitesREPLYELEMENT_V1 *rgCostInfo;
    DWORD dwFlags;
} DRS_MSG_QUERYsitesREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_QUERYsitesREPLY_V1 V1;
} DRS_MSG_QUERYsitesREPLY;

typedef struct {
    DWORD dwReserved;
} DRS_MSG_INIT_DEMOTIONREQ_V1;

```

```

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_INIT_DEMOTIONREQ_V1 V1;
} DRS_MSG_INIT_DEMOTIONREQ;

typedef struct {
    DWORD dwOpError;
} DRS_MSG_INIT_DEMOTIONREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_INIT_DEMOTIONREPLY_V1 V1;
} DRS_MSG_INIT_DEMOTIONREPLY;

typedef struct {
    DWORD dwFlags;
    UUID uuidHelperDest;
    DSNAME* pNC;
} DRS_MSG_REPLICA_DEMOTIONREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_REPLICA_DEMOTIONREQ_V1 V1;
} DRS_MSG_REPLICA_DEMOTIONREQ;

typedef struct {
    DWORD dwOpError;
} DRS_MSG_REPLICA_DEMOTIONREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_REPLICA_DEMOTIONREPLY_V1 V1;
} DRS_MSG_REPLICA_DEMOTIONREPLY;

typedef struct {
    DWORD dwOperations;
    UUID uuidHelperDest;
    LPWSTR szScriptBase;
} DRS_MSG_FINISH_DEMOTIONREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_FINISH_DEMOTIONREQ_V1 V1;
} DRS_MSG_FINISH_DEMOTIONREQ;

typedef struct {
    DWORD dwOperationsDone;
    DWORD dwOpFailed;
    DWORD dwOpError;
} DRS_MSG_FINISH_DEMOTIONREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_FINISH_DEMOTIONREPLY_V1 V1;
} DRS_MSG_FINISH_DEMOTIONREPLY;

// opnum 0
ULONG
IDL_DRSBind(
    [in] handle_t rpc_handle,
    [in, unique] UUID *puuidClientDsa,
    [in, unique] DRS_EXTENSIONS *pextClient,
    [out] DRS_EXTENSIONS **ppextServer,

```

```

        [out, ref] DRS_HANDLE *phDrs);

// opnum 1
ULONG
IDL_DRSUnbind(
    [in, out, ref] DRS_HANDLE *phDrs);

// opnum 2
ULONG
IDL_DRSReplicaSync(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPSYNC *pmsgSync);

// opnum 3
ULONG
IDL_DRSGetNCChanges(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_GETCHGREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_GETCHGREPLY *pmsgOut);

// opnum 4
ULONG
IDL_DRSUpdateRefs(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_UPDREFS *pmsgUpdRefs);

// opnum 5
ULONG
IDL_DRSReplicaAdd(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPADD *pmsgAdd);

// opnum 6
ULONG
IDL_DRSReplicaDel(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPDEL *pmsgDel);

// opnum 7
ULONG
IDL_DRSReplicaModify(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPMOD *pmsgMod);

// opnum 8
ULONG
IDL_DRSVerifyNames(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_VERIFYREQ *pmsgIn,

```

```

        [out, ref] DWORD *pdwOutVersion,
        [out, ref, switch_is(*pdwOutVersion)]
            DRS_MSG_VERIFYREPLY *pmsgOut);

// opnum 9
ULONG
IDL_DRSGetMemberships(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_REVMEMB_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_REVMEMB_REPLY *pmsgOut);

// opnum 10
ULONG
IDL_DRSInterDomainMove(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_MOVEREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)] DRS_MSG_MOVEREPLY *pmsgOut);

// opnum 11
ULONG
IDL_DRSGetNT4ChangeLog(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_NT4_CHGLOG_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_NT4_CHGLOG_REPLY *pmsgOut);

// opnum 12
ULONG
IDL_DRSCrackNames(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_CRACKREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_CRACKREPLY *pmsgOut);

// opnum 13
ULONG
IDL_DRSWriteSPN(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_SPNREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)] DRS_MSG_SPNREPLY *pmsgOut);

// opnum 14
ULONG
IDL_DRSRemoveDsServer(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_RMSVRREQ *pmsgIn,

```



```

        [out, ref] DWORD *pdwOutVersion,
        [out, ref, switch_is(*pdwOutVersion)]
            DRS_MSG_RMSVRREPLY *pmsgOut);

// opnum 15
ULONG
IDL_DRSSRemovedsDomain(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_RMDMNREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_RMDMNREPLY *pmsgOut);

// opnum 16
ULONG
IDL_DRSDomainControllerInfo(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_DCINFOREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_DCINFOREPLY *pmsgOut);

// opnum 17
ULONG
IDL_DRSSAddEntry(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_ADDENTRYREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_ADDENTRYREPLY *pmsgOut);

// opnum 18
ULONG
IDL_DRSExecuteKCC(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_KCC_EXECUTE *pmsgIn);

// opnum 19
ULONG
IDL_DRSSGetReplInfo(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_GETREPLINFO_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_GETREPLINFO_REPLY *pmsgOut);

// opnum 20
ULONG
IDL_DRSSAddSidHistory(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_ADDSIDREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,

```

```

        [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_ADDSIDREPLY *pmsgOut);

// opnum 21
ULONG
IDL_DRSGetMemberships2(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_GETMEMBERSHIPS2_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_GETMEMBERSHIPS2_REPLY *pmsgOut);

// opnum 22
ULONG
IDL_DRSReplicaVerifyObjects(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPVERIFYOBJ *pmsgVerify);

// opnum 23
ULONG
IDL_DRSGetObjectExistence (
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_EXISTREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_EXISTREPLY *pmsgOut);

// opnum 24
ULONG
IDL_DRSQuerySitesByCost(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_QUERYSITESREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_QUERYSITESREPLY *pmsgOut);

// opnum 25
ULONG
IDL_DRSInitDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_INIT_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_INIT_DEMOTIONREPLY* pmsgOut);

// opnum 26
ULONG
IDL_DRSReplicaDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]

```

```

        DRS_MSG_REPLICA_DEMOTIONREQ* pmsgIn,
[out, ref] DWORD* pdwOutVersion,
[out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_REPLICA_DEMOTIONREPLY* pmsgOut);

// opnum 27
ULONG
IDL_DRSFinishDemotion(
[in, ref] DRS_HANDLE hDrs,
[in] DWORD dwInVersion,
[in, ref, switch_is(dwInVersion)]
        DRS_MSG_FINISH_DEMOTIONREQ* pmsgIn,
[out, ref] DWORD* pdwOutVersion,
[out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_FINISH_DEMOTIONREPLY* pmsgOut);

}

// This is the "real" ntdscript interface.
[
    uuid(7c44d7d4-31d5-424c-bd5e-2b3e1f323d22), version(1.0),
    pointer_default (unique)
]
interface dsaop
{

typedef struct {
    DWORD Flags;
    [range(1,1024)] DWORD cbPassword;
    [size_is(cbPassword)] BYTE *pbPassword;
} DSA_MSG_EXECUTE_SCRIPT_REQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DSA_MSG_EXECUTE_SCRIPT_REQ_V1 V1;
} DSA_MSG_EXECUTE_SCRIPT_REQ;

typedef struct {
    DWORD dwOperationStatus;
    [string] LPWSTR pwErrMessage;
} DSA_MSG_EXECUTE_SCRIPT_REPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DSA_MSG_EXECUTE_SCRIPT_REPLY_V1 V1;
} DSA_MSG_EXECUTE_SCRIPT_REPLY;

typedef struct {
    DWORD Reserved;
} DSA_MSG_PREPARE_SCRIPT_REQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DSA_MSG_PREPARE_SCRIPT_REQ_V1 V1;
} DSA_MSG_PREPARE_SCRIPT_REQ;

typedef struct {
    DWORD dwOperationStatus;
    [string] LPWSTR pwErrMessage;
    [range(0,1024)] DWORD cbPassword;

```

```

        [size_is(cbPassword)] BYTE *pbPassword;
        [range(0,10485760)] DWORD cbHashBody;
        [size_is(cbHashBody)] BYTE *pbHashBody;
        [range(0,10485760)] DWORD cbHashSignature;
        [size_is(cbHashSignature)] BYTE *pbHashSignature;
    } DSA_MSG_PREPARE_SCRIPT_REPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DSA_MSG_PREPARE_SCRIPT_REPLY_V1 V1;
} DSA_MSG_PREPARE_SCRIPT_REPLY;

// opnum 0
ULONG
IDL_DSAPrepareScript(
    [in] handle_t hRpc,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DSA_MSG_PREPARE_SCRIPT_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DSA_MSG_PREPARE_SCRIPT_REPLY *pmsgOut);

// opnum 1
ULONG
IDL_DSASExecuteScript(
    [in] handle_t hRpc,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DSA_MSG_EXECUTE_SCRIPT_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DSA_MSG_EXECUTE_SCRIPT_REPLY *pmsgOut);
}

```

8 Appendix B: Windows Behavior

All IDL methods and their associated concrete types have existed in the [drsuapi RPC interface](#) since Windows 2000 except those listed in the following table.

Data type or IDL method	Section	Windows version introduced
DRS_MSG_GETCHGREQ_V7 support	4.1.10.2.5	Windows Server 2003
DRS_MSG_GETCHGREQ_V8 support	4.1.10.2.6	Windows Server 2003
DRS_MSG_GETCHGREPLY_V6 support	4.1.10.2.10	Windows Server 2003
DRS_MSG_GETCHGREPLY_V7 support	4.1.10.2.11	Windows Server 2003
DRS_MSG_ADDENTRYREQ_V3 support	4.1.1.1.4	Windows Server 2003
DRS_MSG_ADDENTRYREPLY_V3 support	4.1.1.1.8	Windows Server 2003
DRS_MSG_GETREPLINFO_REQ_V2	4.1.13.1.3	Windows Server 2003
IDL_DRSGetObjectExistence	4.1.12	Windows Server 2003
IDL_DRSReplicaVerifyObjects	4.1.24	Windows Server 2003
IDL_DRSFinishDemotion	4.1.7	Windows Server 2008
IDL_DRSInitDemotion	4.1.14	Windows Server 2008
IDL_DRSReplicaDemotion	4.1.21	Windows Server 2008

The information in this specification is applicable to the following versions of Windows:

- Windows 2000
- Windows Server 2003
- Windows XP
- Windows Vista
- Windows Server 2008

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

[<1> Section 2.1:](#) Windows servers listen only on the RPC-over-TCP protocol sequence. Windows clients attempt to connect only through the RPC-over-TCP protocol sequence.

[<2> Section 2.2.3:](#) Windows implements DC-to-DC interaction with an SPN with service class "E3514235-4B06-11D1-AB04-00C04FC2DCD2". See [DRS SPN CLASS](#).

[<3> Section 3.6:](#) The client reuses an association for multiple invocations.

[<4> Section 4.1.1.1.2:](#) Though this request version appears in the IDL, Windows DCs do not support it. It was never supported in a released version of Windows Server.

<5> [Section 4.1.1.1.6:](#) Though this response version appears in the IDL, Windows DCs do not support it.

<6> [Section 4.1.2.2.6:](#) The function determines whether auditing is enabled on the server by querying the LSA information policy on the server associated with *ctx* and by confirming that the information policy is set to generate both success and failure audits for the "account management" audit category. To achieve this, the *LsarOpenPolicy2*, *LsarQueryInformationPolicy*, and *LsarClose* messages in [\[MS-LSAD\]](#) are used ([\[MS-LSAD\]](#) sections [3.1.4.4.1](#), [3.1.4.4.4](#), and [3.1.4.9.5](#)). The *SystemName* parameter to *LsarOpenPolicy2* is generated by using the **srcDomainController** variable in the [IDL DRSAddSidHistory](#) method and the *DesiredAccess* parameter to *LsarOpenPolicy2* is set to (*POLICY_VIEW_AUDIT_INFORMATION* | *POLICY_VIEW_LOCAL_INFORMATION*). On success, the *PolicyHandle* acquired from the *LsarOpenPolicy2* message is passed to the *LsarQueryInformationPolicy* message with *PolicyAuditEventsInformation* as the information class. The check to determine whether success and failure audits are enabled for "account management" is achieved by performing the following evaluation:

```
PolicyInformation^.PolicyAuditEventsInfo.EventAuditingOptions[6] ∩  
  { POLICY_AUDIT_EVENT_SUCCESS, POLICY_AUDIT_EVENT_FAILURE } =  
  { POLICY_AUDIT_EVENT_SUCCESS, POLICY_AUDIT_EVENT_FAILURE }
```

where *PolicyInformation* is the result from the *LsarQueryInformationPolicy* message. *PolicyHandle* is then closed by using the *LsarClose* message.

The function generates an audit on the DC associated with *ctx* by adding the source principal (*pmsgIn*^.*V1.SrcPrincipal*, where *pmsgIn* is a parameter to the [IDL DRSAddSidHistory](#) function, which in turn calls this method) to the group *srcDomainFlatName*\$\$\$ on the DC associated with *ctx*, where *srcDomainFlatName* is the NetBIOS name of the domain to which the source principal belongs. After adding the principal to the group, it then removes the principal from the group, leaving the group in its original state but having generated an audit event as a side effect of manipulating the group's membership.

<7> [Section 4.1.2.2.12:](#) This test is implemented in two steps. First, it is determined if the DC associated with *ctx* is running at least Windows 2000. This is determined by whether a *SamrConnect5* or *SamrConnect4* API call (as specified in [\[MS-SAMR\]](#)) to the DC is successful. If it is, the DC is running at least Windows 2000, and the function returns true.

Otherwise, the DC is considered to be running Windows NT 4.0. The function then connects to the registry service on the DC named in *ctx* and queries the value of the "CSDVersion" registry value on the "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion" registry key. If the value is not equal to any of the following strings, the function returns true; otherwise, it returns false:

- Service Pack 0
- Service Pack 1
- Service Pack 2
- Service Pack 3

If the registry service on the DC could not be contacted, or if the registry key or registry value does not exist, the function returns false.

<8> [Section 4.1.3.1:](#) Windows non-DC client callers always use the following UUID for puuidClientDsa: e24d201a-4fd6-11d1-a3da-0000f875ae0d.

<9> [Section 4.1.3.1:](#) Windows non-DC client callers always set the dwFlags field of the [DRS_EXTENSIONS_INT](#) structure to zero. Windows non-DC client callers always set the SiteObjGuid field of the [DRS_EXTENSIONS_INT](#) structure to the NULL GUID value. Windows non-DC client callers always set the Pid field of the [DRS_EXTENSIONS_INT](#) structure to an implementation-specific, client-local process identifier (PID).

<10> [Section 4.1.6.3:](#) The KCC is an internal component that runs on all DCs. The KCC generates and maintains the replication topology between DCs within sites and between sites. The KCC performs two major tasks:

- Configures replication connections (nTDSConnection objects) between DCs. Each nTDSConnection object defines a replication link between the local DC and a replication partner DC. The local DC sends [IDL DRSGetNCChanges](#) requests to the replication partner DC.
- Converts the connection objects that represent replication to the local DC into an [repsFrom](#) values used by DC replication.

These tasks are specified in [\[MS-ADTS\]](#) section 7.2. Upon receiving this message, the server triggers the KCC to perform the preceding two tasks.

<11> [Section 4.1.10.1.1:](#) The Windows implementation never declares msgIn.uuidInvocIdSrc and msgIn.usnvecFrom that are otherwise valid to be stale.

<12> [Section 4.1.10.1.1:](#) The internal format of [USN_VECTOR](#) identifies the start of a cycle.

<13> [Section 4.1.10.1.2:](#) The goal is advanced on each request of a cycle.

<14> [Section 4.1.10.1.2:](#) c.usnHighPropUpdate is never set to 0.

<15> [Section 4.1.10.2.2:](#) Though this request version appears in the IDL, Windows DCs never send this request version by means of RPC. It exists solely to support SMTP replication (see [\[MS-SRPL\]](#)).

<16> [Section 4.1.10.2.3:](#) Although this request version appears in the IDL, Windows DCs never send this request version using RPC. It exists solely to support SMTP replication ([\[MS-SRPL\]](#)).

<17> [Section 4.1.10.2.5:](#) Though this request version appears in the IDL, Windows DCs never send this request version using RPC. It exists solely to support SMTP replication ([\[MS-SRPL\]](#)).

<18> [Section 4.1.10.5.6:](#) The server tests the response against the limits after adding each object and link to the response, unless the object is a parent object that is being included because of the ancestor's predicate (see the [GetReplChanges](#) method). If the test shows that the response has exceeded one of the limits, the server stops adding to the response. The server may return more objects or bytes than the limits.

<19> [Section 4.1.10.6.4:](#) Windows DCs assign the remainder of the bits in values of o!instanceType for a given object o as follows:

- IT_UNINSTANT: Set if and only if o is an NC root and its NC replica is not present on the DC.
- IT_NC_ABOVE: Set if and only if o is an NC root and the DC has an NC replica with NC root p such that p is the parent of o.
- IT_NC_COMING: Set if and only if o is an NC root and the DC has not yet completed the first replication cycle for that NC replica.

- IT_NC_GOING: Set if and only if o is an NC root and the DC is in the process of removing its replica of the NC.

<20> [Section 4.1.12.3:](#) Windows uses count = 1000.

<21> [Section 4.1.15.1.2:](#) Though this request version appears in the IDL, Windows DCs do not support it. It was never supported in a released version of Windows Server.

<22> [Section 4.1.15.1.5:](#) Though this response version appears in the IDL, WindowsDCs do not support it.

<23> [Section 4.1.16.3:](#) The server returns the error ERROR_DS_GENERIC_ERROR if the Intersite Messaging Service is not running on the server.

<24> [Section 4.1.24.3:](#) The Windows implementation of the for loop uses [IDL DRSGetObjectExistence](#) to determine if object o exists at refDsa, and logs a message to the Windows Event Log.

<25> [Section 5.15.3.13:](#) Only [Object\(OR-Name\)](#) values that consist of only a DN are supported and can be converted to the [SYNTAX DISTNAME BINARY](#) syntax. Values with an OR_Name portion cannot be converted and are rejected by the DC.

<26> [Section 5.35:](#) Client callers set **dwFlags** to zero.

<27> [Section 5.35:](#) This field contains the process ID of the client.

9 Index

A

[ADDENTRY_REPLY_INFO structure](#)
[Applicability](#)
[ATRERR_DRS_WIRE_V1 structure](#)
[ATTR structure](#)
[ATTRBLOCK structure](#)
[ATTRVAL structure](#)
[ATTRVALBLOCK structure](#)

B

[Background - behavior specifications](#)

C

[Capability negotiation](#)
[Common data types](#)
[COMPRESSED_DATA structure](#)
[CONTREF_DRS_WIRE_V1 structure](#)
[Conventions](#)
[Conventions - examples](#)

D

[Data types](#)
[Document conventions](#)
[DRS_COMP_ALG_TYPE enumeration](#)
[DRS_COMPRESSED_BLOB structure](#)
[DRS_ERROR_DATA_V1 structure](#)
[DRS_EXTENSIONS structure](#)
[DRS_EXTENSIONS_INT packet](#)
[DRS_MSG_ADDENTRYREPLY_V1 structure](#)
[DRS_MSG_ADDENTRYREPLY_V2 structure](#)
[DRS_MSG_ADDENTRYREPLY_V3 structure](#)
[DRS_MSG_ADDENTRYREQ_V1 structure](#)
[DRS_MSG_ADDENTRYREQ_V2 structure](#)
[DRS_MSG_ADDENTRYREQ_V3 structure](#)
[DRS_MSG_ADDSIDREPLY_V1 structure](#)
[DRS_MSG_ADDSIDREQ_V1 structure](#)
[DRS_MSG_CRACKREPLY_V1 structure](#)
[DRS_MSG_CRACKREQ_V1 structure](#)
[DRS_MSG_DCINFOREPLY_V1 structure](#)
[DRS_MSG_DCINFOREPLY_V2 structure](#)
[DRS_MSG_DCINFOREPLY_V3 structure](#)
[DRS_MSG_DCINFOREPLY_VFFFFFFFF structure](#)
[DRS_MSG_DCINFOREQ_V1 structure](#)
[DRS_MSG_EXISTREPLY_V1 structure](#)
[DRS_MSG_EXISTREQ_V1 structure](#)
[DRS_MSG_FINISH_DEMOTIONREPLY_V1 structure](#)
[DRS_MSG_FINISH_DEMOTIONREQ_V1 structure](#)
[DRS_MSG_GETCHGREPLY_V1 structure](#)
[DRS_MSG_GETCHGREPLY_V2 structure](#)
[DRS_MSG_GETCHGREPLY_V6 structure](#)
[DRS_MSG_GETCHGREPLY_V7 structure](#)
[DRS_MSG_GETCHGREQ_V3 structure](#)
[DRS_MSG_GETCHGREQ_V4 structure](#)
[DRS_MSG_GETCHGREQ_V5 structure](#)
[DRS_MSG_GETCHGREQ_V7 structure](#)

[DRS_MSG_GETCHGREQ_V8 structure](#)
[DRS_MSG_GETMEMBERSHIPS2_REPLY_V1 structure](#)
[DRS_MSG_GETMEMBERSHIPS2_REQ_V1 structure](#)
[DRS_MSG_GETREPLINFO_REQ_V1 structure](#)
[DRS_MSG_GETREPLINFO_REQ_V2 structure](#)
[DRS_MSG_INIT_DEMOTIONREPLY_V1 structure](#)
[DRS_MSG_INIT_DEMOTIONREQ_V1 structure](#)
[DRS_MSG_KCC_EXECUTE_V1 structure](#)
[DRS_MSG_MOVEREPLY_V1 structure](#)
[DRS_MSG_MOVEREPLY_V2 structure](#)
[DRS_MSG_MOVEREQ_V1 structure](#)
[DRS_MSG_MOVEREQ_V2 structure](#)
[DRS_MSG_NT4_CHGLOG_REPLY_V1 structure](#)
[DRS_MSG_NT4_CHGLOG_REQ_V1 structure](#)
[DRS_MSG_QUERYsitesREPLY_V1 structure](#)
[DRS_MSG_QUERYsitesREPLYELEMENT_V1 structure](#)
[DRS_MSG_QUERYsitesREQ_V1 structure](#)
[DRS_MSG_REPADD_V1 structure](#)
[DRS_MSG_REPADD_V2 structure](#)
[DRS_MSG_REPDEL_V1 structure](#)
[DRS_MSG_REPLICA_DEMOTIONREPLY_V1 structure](#)
[DRS_MSG_REPLICA_DEMOTIONREQ_V1 structure](#)
[DRS_MSG_REPMOD_V1 structure](#)
[DRS_MSG_REPSYNC_V1 structure](#)
[DRS_MSG_REPVERIFYOBJ_V1 structure](#)
[DRS_MSG_REVMEMB_REPLY_V1 structure](#)
[DRS_MSG_REVMEMB_REQ_V1 structure](#)
[DRS_MSG_RMDMNRREPLY_V1 structure](#)
[DRS_MSG_RMDMNRREQ_V1 structure](#)
[DRS_MSG_RMSVRREPLY_V1 structure](#)
[DRS_MSG_RMSVRREQ_V1 structure](#)
[DRS_MSG_SPNREPLY_V1 structure](#)
[DRS_MSG_SPNREQ_V1 structure](#)
[DRS_MSG_UPDREFS_V1 structure](#)
[DRS_MSG_VERIFYREPLY_V1 structure](#)
[DRS_MSG_VERIFYREQ_V1 structure](#)
[DRS_SecBuffer structure](#)
[DRS_SecBufferDesc structure](#)
[DS_DOMAIN_CONTROLLER_INFO_1W structure](#)
[DS_DOMAIN_CONTROLLER_INFO_2W structure](#)
[DS_DOMAIN_CONTROLLER_INFO_3W structure](#)
[DS_DOMAIN_CONTROLLER_INFO_FFFFFFFFW structure](#)
[DS_NAME_FORMAT enumeration](#)
[DS_NAME_RESULT_ITEMW structure](#)
[DS_NAME_RESULTW structure](#)
[DS_REPL_ATTR_META_DATA structure](#)
[DS_REPL_ATTR_META_DATA_2 structure](#)
[DS_REPL_ATTR_VALUE_META_DATA structure](#)
[DS_REPL_ATTR_VALUE_META_DATA_2 structure](#)
[DS_REPL_CLIENT_CONTEXT structure](#)
[DS_REPL_CLIENT_CONTEXTS structure](#)
[DS_REPL_CURSOR structure](#)
[DS_REPL_CURSOR_2 structure](#)
[DS_REPL_CURSOR_3W structure](#)
[DS_REPL_CURSORS structure](#)
[DS_REPL_CURSORS_2 structure](#)
[DS_REPL_CURSORS_3W structure](#)
[DS_REPL_KCC_DSA_FAILURESW structure](#)
[DS_REPL_KCC_DSA_FAILUREW structure](#)

[DS_REPL_NEIGHBORSW structure](#)
[DS_REPL_NEIGHBORW structure](#)
[DS_REPL_OBJ_META_DATA structure](#)
[DS_REPL_OBJ_META_DATA_2 structure](#)
[DS_REPL_OP_TYPE; enumeration](#)
[DS_REPL_OPW structure](#)
[DS_REPL_PENDING_OPSW structure](#)
[DS_REPL_SERVER_OUTGOING_CALL structure](#)
[DS_REPL_SERVER_OUTGOING_CALLS structure](#)
[DS_REPL_VALUE_META_DATA structure](#)
[DS_REPL_VALUE_META_DATA_2 structure](#)
[DSA_ADDRESS_LIST_DRS_WIRE_V1 structure](#)
[DSA_MSG_EXECUTE_SCRIPT_REPLY_V1 structure](#)
[DSA_MSG_EXECUTE_SCRIPT_REQ_V1 structure](#)
[DSA_MSG_PREPARE_SCRIPT_REPLY_V1 structure](#)
[DSA_MSG_PREPARE_SCRIPT_REQ_V1 structure](#)
[DSNAME structure](#)

E

[ENCRYPTED_PAYLOAD packet](#)
[ENTINF structure](#)
[ENTINFLIST structure](#)

F

[Fields - vendor-extensible](#)
[FOREST_TRUST_INFORMATION packet](#)
[FOREST_TRUST_RECORD_TYPE enumeration](#)
[Full IDL](#)

G

[Glossary](#)

I

[IDL](#)
[IDL_DRSAddEntry method](#)
[IDL_DRSAddSidHistory method](#)
[IDL_DRSBind method](#)
[IDL_DRSCrackNames method](#)
[IDL_DRSDomainControllerInfo method](#)
[IDL_DRSExecuteKCC method](#)
[IDL_DRSFinishDemotion method](#)
[IDL_DRSGetMemberships method](#)
[IDL_DRSGetMemberships2 method](#)
[IDL_DRSGetNCChanges method](#)
[IDL_DRSGetNT4ChangeLog method](#)
[IDL_DRSGetObjectExistence method](#)
[IDL_DRSGetReplInfo method](#)
[IDL_DRSInitDemotion method](#)
[IDL_DRSInterDomainMove method](#)
[IDL_DRSQuerySitesByCost method](#)
[IDL_DRSRemoveDsDomain method](#)
[IDL_DRSRemoveDsServer method](#)
[IDL_DRSReplicaAdd method](#)
[IDL_DRSReplicaDel method](#)
[IDL_DRSReplicaDemotion method](#)
[IDL_DRSReplicaModify method](#)
[IDL_DRSReplicaSync method](#)

[IDL_DRSReplicaVerifyObjects method](#)
[IDL_DRSUnbind method](#)
[IDL_DRSUpdateRefs method](#)
[IDL_DRSVerifyNames method](#)
[IDL_DRSWriteSPN method](#)
[IDL_DSAScriptExecute method](#)
[IDL_DSAPrepareScript method](#)
[Implementer - security considerations](#)
[Index of security parameters](#)
[Informative references](#)
[Initialization](#)
[INT32 packet](#)
[INT64 packet](#)
[INTFORMPROB_DRS_WIRE_V1 structure](#)
[Introduction](#)

M

[Messages](#)
[RPC transport](#)
[transport](#)
[MTX_ADDR structure](#)

N

[NAMERESOP_DRS_WIRE_V1 structure](#)
[NAMERR_DRS_WIRE_V1 structure](#)
[Normative references](#)
[NT4_REPLICATION_STATE structure](#)
[NT4SID structure](#)

O

[OID_t structure](#)
[Organization](#)
[Overview \(synopsis\)](#)

P

[packet](#)
[Parameters - security index](#)
[PARTIAL_ATTR_VECTOR_V1_EXT structure](#)
[PAS_DATA packet](#)
[PDS_NAME_RESULT_ITEMW](#)
[PDS_NAME_RESULTW](#)
[Preconditions](#)
[PrefixTableEntry structure](#)
[Prerequisites](#)
[PROBLEMLIST_DRS_WIRE_V1 structure](#)
[Procedures](#)
[PROPERTY_META_DATA structure](#)
[PROPERTY_META_DATA_EXT structure](#)
[PROPERTY_META_DATA_EXT_VECTOR structure](#)
[Pseudocode](#)

R

[Record packet](#)
[References](#)
[informative](#)
[normative](#)

[overview](#)
[REFERR_DRS_WIRE_V1 structure](#)
[Relationship to other protocols](#)
[REPLENTINFLIST structure](#)
[REPLTIMES structure](#)
[REPLVALINF structure](#)
[REPS_FROM packet](#)
[REPS_TO packet](#)
[REVERSE_MEMBERSHIP_OPERATION_TYPE enumeration](#)
[RPC methods - behavior](#)
[RPC transport](#)

S

[SCHEMA_PREFIX_TABLE structure](#)
[SECERR_DRS_WIRE_V1 structure](#)
Security
 [implementer considerations](#)
 [overview](#) ([section 2.2](#), [section 6](#))
 [parameter index](#)
[Standards assignments](#)
[State model](#)
[SVCERR_DRS_WIRE_V1 structure](#)
[SYNTAX_ADDRESS packet](#)
[SYSERR_DRS_WIRE_V1 structure](#)

T

[Transport](#)
[Typographical conventions](#)

U

[ULARGE_INTEGER structure](#)
[UPDERR_DRS_WIRE_V1 structure](#)
[UPTODATE_CURSOR_V1 structure](#)
[UPTODATE_CURSOR_V2 structure](#)
[UPTODATE_VECTOR_V1_EXT structure](#)
[UPTODATE_VECTOR_V2_EXT structure](#)
[USN_VECTOR structure](#)

V

[VALUE_META_DATA_EXT_V1 structure](#)
[Variables](#)
[Vendor-extensible fields](#)
[Versioning](#)
[Version-specific behavior](#)