

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

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This 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 technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft [Open Specification Promise](#) or the [Community Promise](#). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** 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.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of 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.

Tools. The Open Specifications do 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. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
03/14/2007	1.0		Version 1.0 release
04/10/2007	1.1		Version 1.1 release
05/18/2007	1.2		Version 1.2 release
06/08/2007	1.2.1	Editorial	Revised and edited the technical content.
07/10/2007	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.
03/14/2008	2.1	Minor	Updated the technical content.
06/20/2008	2.2	Minor	Updated the technical content.
07/25/2008	3.0	Major	Updated and revised the technical content.
08/29/2008	4.0	Major	Updated and revised the technical content.
10/24/2008	5.0	Major	Updated and revised the technical content.
12/05/2008	6.0	Major	Updated and revised the technical content.
01/16/2009	7.0	Major	Updated and revised the technical content.
02/27/2009	8.0	Major	Updated and revised the technical content.
04/10/2009	9.0	Major	Updated and revised the technical content.
05/22/2009	10.0	Major	Updated and revised the technical content.
07/02/2009	11.0	Major	Updated and revised the technical content.
08/14/2009	12.0	Major	Updated and revised the technical content.
09/25/2009	13.0	Major	Updated and revised the technical content.
11/06/2009	14.0	Major	Updated and revised the technical content.
12/18/2009	15.0	Major	Updated and revised the technical content.
01/29/2010	16.0	Major	Updated and revised the technical content.
03/12/2010	17.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
04/23/2010	18.0	Major	Updated and revised the technical content.
06/04/2010	19.0	Major	Updated and revised the technical content.
07/16/2010	20.0	Major	Significantly changed the technical content.
08/27/2010	21.0	Major	Significantly changed the technical content.
10/08/2010	22.0	Major	Significantly changed the technical content.
11/19/2010	23.0	Major	Significantly changed the technical content.
01/07/2011	24.0	Major	Significantly changed the technical content.
02/11/2011	24.0	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	25.0	Major	Significantly changed the technical content.
05/06/2011	25.0	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	25.1	Minor	Clarified the meaning of the technical content.

Contents

1	Introduction	22
1.1	Glossary	23
1.2	References.....	35
1.2.1	Normative References.....	35
1.2.2	Informative References	37
1.3	Overview	37
1.3.1	Methods Categorized by Function	37
1.3.2	Sequencing Issues.....	38
1.3.3	Most Frequently Used Types	40
1.4	Relationship to Other Protocols.....	41
1.5	Prerequisites/Preconditions	42
1.6	Applicability Statement.....	42
1.7	Versioning and Capability Negotiation.....	42
1.8	Vendor-Extensible Fields.....	42
1.9	Standards Assignments	42
2	Message Transport	43
2.1	RPC Transport	43
2.2	Protocol Security	43
2.2.1	General Background	43
2.2.2	Service Principal Names for Domain Controllers	43
2.2.3	DC-to-DC Operations	44
2.2.3.1	Security Provider	44
2.2.3.2	SPN for a Target DC in AD DS	44
2.2.3.3	SPN for a Target DC in AD LDS.....	45
2.2.4	Client-to-DC Operations	45
2.2.4.1	Security Provider	46
2.2.4.2	SPN for a Target DC in AD DS	46
2.2.4.3	SPN for a Target DC in AD LDS.....	47
2.3	Directory Service Schema Elements	48
3	Background to Behavior Specifications	49
3.1	Document Organization	49
3.2	Typographical Conventions	49
3.3	State Model	50
3.3.1	Preliminaries.....	50
3.3.2	Transactions	50
3.3.3	Concrete and Abstract Types	50
3.4	Pseudocode Language	51
3.4.1	Naming Conventions.....	51
3.4.2	Language Constructs for Concrete Types	52
3.4.3	Language Constructs for Abstract Types.....	52
3.4.4	Common Language Constructs.....	54
3.4.5	Access to Objects and Their Attributes	56
3.4.6	Asynchronous Processing	58
3.5	Conventions for Protocol Examples.....	59
3.5.1	Common Configuration	59
3.5.2	Data Display Conventions.....	60
3.6	Server and Client Initialization.....	61
3.6.1	AD LDS Specifics	61

4	RPC Methods and Their Behavior	62
4.1	drsuapi RPC Interface	62
4.1.1	IDL_DRSAddEntry (Opnum 17)	64
4.1.1.1	Method-Specific Concrete Types	65
4.1.1.1.1	DRS_MSG_ADDENTRYREQ	65
4.1.1.1.2	DRS_MSG_ADDENTRYREQ_V1	65
4.1.1.1.3	DRS_MSG_ADDENTRYREQ_V2	65
4.1.1.1.4	DRS_MSG_ADDENTRYREQ_V3	66
4.1.1.1.5	DRS_MSG_ADDENTRYREPLY	66
4.1.1.1.6	DRS_MSG_ADDENTRYREPLY_V1	66
4.1.1.1.7	DRS_MSG_ADDENTRYREPLY_V2	67
4.1.1.1.8	DRS_MSG_ADDENTRYREPLY_V3	67
4.1.1.1.9	ADDENTRY_REPLY_INFO	68
4.1.1.1.10	DIRERR_DRS_WIRE_V1	68
4.1.1.1.11	ATRERR_DRS_WIRE_V1	69
4.1.1.1.12	PROBLEMLIST_DRS_WIRE_V1	69
4.1.1.1.13	INTFORMPROB_DRS_WIRE_V1	69
4.1.1.1.14	NAMERR_DRS_WIRE_V1	70
4.1.1.1.15	REFERR_DRS_WIRE_V1	70
4.1.1.1.16	NAMERESOP_DRS_WIRE_V1	71
4.1.1.1.17	DSA_ADDRESS_LIST_DRS_WIRE_V1	71
4.1.1.1.18	CONTREF_DRS_WIRE_V1	71
4.1.1.1.19	SECERR_DRS_WIRE_V1	72
4.1.1.1.20	SVCERR_DRS_WIRE_V1	73
4.1.1.1.21	UPDERR_DRS_WIRE_V1	73
4.1.1.1.22	SYSERR_DRS_WIRE_V1	73
4.1.1.1.23	DRS_ERROR_DATA	74
4.1.1.1.24	DRS_ERROR_DATA_V1	74
4.1.1.1.25	DIRERR Codes	74
4.1.1.1.26	PROBLEM Error Codes	75
4.1.1.2	Method-Specific Abstract Types and Procedures	77
4.1.1.2.1	ConstructReplSpn	77
4.1.1.2.2	CreateCrossRef	77
4.1.1.2.3	CreateNtdsDsa	79
4.1.1.2.4	UseCredsForAccessCheck	81
4.1.1.2.5	IsDomainToBeCreated	81
4.1.1.2.6	GetDomainNameFromEntinf	81
4.1.1.2.7	ENTINF_GetAttribute	82
4.1.1.2.8	SetErrorData	82
4.1.1.2.9	ClientIpMatch	83
4.1.1.2.10	PerformModifyEntInf	83
4.1.1.3	Server Behavior of the IDL_DRSAddEntry Method	83
4.1.2	IDL_DRSAddSidHistory (Opnum 20)	87
4.1.2.1	Method-Specific Concrete Types	87
4.1.2.1.1	DRS_MSG_ADDSIDREQ	87
4.1.2.1.2	DRS_MSG_ADDSIDREQ_V1	88
4.1.2.1.3	DRS_MSG_ADDSIDREPLY	89
4.1.2.1.4	DRS_MSG_ADDSIDREPLY_V1	89
4.1.2.1.5	DRS_ADDSID_FLAGS	89
4.1.2.2	Method-Specific Abstract Types and Procedures	90
4.1.2.2.1	ConnectionCtx	90
4.1.2.2.2	ConnectToDC	90
4.1.2.2.3	ConnectToDCWithCreds	90

4.1.2.2.4	GenerateFailureAudit	90
4.1.2.2.5	GenerateSuccessAudit	91
4.1.2.2.6	GenerateSuccessAuditRemotely	91
4.1.2.2.7	GetKeyLength	91
4.1.2.2.8	GetPDC	91
4.1.2.2.9	HasAdminRights	91
4.1.2.2.10	IsAuditingEnabled	91
4.1.2.2.11	IsLocalRpcCall	92
4.1.2.2.12	IsNT4SP4OrBetter	92
4.1.2.2.13	IsAuditingGroupPresent	92
4.1.2.2.14	IsWellKnownDomainRelativeSid	92
4.1.2.2.15	LastRID	92
4.1.2.2.16	RemoteQuery	92
4.1.2.3	Server Behavior of the IDL_DRSAddSidHistory Method	93
4.1.2.4	Examples of the IDL_DRSAddSidHistory Method	100
4.1.2.4.1	Calling IDL_DRSAddSidHistory with DS_ADDSID_FLAG_PRIVATE_CHK_SECURE Flags	100
4.1.2.4.1.1	Client Request	100
4.1.2.4.1.2	Server Response	101
4.1.2.4.1.3	Final State	101
4.1.2.4.2	Calling IDL_DRSAddSidHistory with DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ Flags	101
4.1.2.4.2.1	Initial State	101
4.1.2.4.2.2	Client Request	102
4.1.2.4.2.3	Server Response	102
4.1.2.4.2.4	Final State	102
4.1.2.4.3	Calling IDL_DRSAddSidHistory with 0 in Flags	103
4.1.2.4.3.1	Initial State	103
4.1.2.4.3.2	Client Request	104
4.1.2.4.3.3	Server Response	104
4.1.2.4.3.4	Final State	105
4.1.3	IDL_DRSBind (Opnum 0)	105
4.1.3.1	Client Behavior When Sending the IDL_DRSBind Request	106
4.1.3.2	Server Behavior of the IDL_DRSBind Method	110
4.1.3.3	Client Behavior When Receiving the IDL_DRSBind Response	114
4.1.3.4	Examples of the IDL_DRSBind Method	115
4.1.3.4.1	Initial State	115
4.1.3.4.2	Client Request	116
4.1.3.4.3	Server Response	117
4.1.3.4.4	Final State	118
4.1.4	IDL_DRSCrackNames (Opnum 12)	118
4.1.4.1	Method-Specific Concrete Types	119
4.1.4.1.1	DRS_MSG_CRACKREQ	119
4.1.4.1.2	DRS_MSG_CRACKREQ_V1	119
4.1.4.1.3	DS_NAME_FORMAT	121
4.1.4.1.4	DS_NAME_RESULT_ITEMW	122
4.1.4.1.5	DS_NAME_RESULTW	123
4.1.4.1.6	DRS_MSG_CRACKREPLY	123
4.1.4.1.7	DRS_MSG_CRACKREPLY_V1	123
4.1.4.1.8	DS_NAME_ERROR	123
4.1.4.2	Method-Specific Abstract Types and Procedures	125
4.1.4.2.1	CanonicalNameFromCanonicalNameEx	125
4.1.4.2.2	DomainDNSNameFromDomain	125

4.1.4.2.3	DomainFromDomainDNSName.....	126
4.1.4.2.4	DomainNameFromCanonicalName	126
4.1.4.2.5	DomainNameFromSid	126
4.1.4.2.6	DomainNameFromUPN.....	126
4.1.4.2.7	DomainNetBIOSNameFromDomain.....	126
4.1.4.2.8	DomainSidFromSid.....	126
4.1.4.2.9	CrackNames	127
4.1.4.2.10	LookupName.....	131
4.1.4.2.11	LookupAttr	134
4.1.4.2.12	LookupCanonicalName	134
4.1.4.2.13	GetCanonicalName	135
4.1.4.2.14	LookupSPN	135
4.1.4.2.15	LookupSID	136
4.1.4.2.16	LookupUnknownName.....	137
4.1.4.2.17	LookupUPNAndAltSecID	137
4.1.4.2.18	LookupFPO	138
4.1.4.2.19	MapSPN	139
4.1.4.2.20	ParseCanonicalName.....	139
4.1.4.2.21	RetrieveDCSuffixFromDn	139
4.1.4.2.22	UserNameFromUPN	140
4.1.4.2.23	TranslateFPOToName.....	140
4.1.4.2.24	ConstructOutput.....	141
4.1.4.3	Server Behavior of the IDL_DRSCrackNames Method	141
4.1.4.4	Examples of the IDL_DRSCrackNames Method	142
4.1.4.4.1	Initial State	142
4.1.4.4.2	Client Request	143
4.1.4.4.3	Server Response	143
4.1.4.4.4	Final State.....	144
4.1.5	IDL_DRSDomainControllerInfo (Opnum 16).....	144
4.1.5.1	Method-Specific Concrete Types	145
4.1.5.1.1	DRS_MSG_DCINFOREQ.....	145
4.1.5.1.2	DRS_MSG_DCINFOREQ_V1	145
4.1.5.1.3	DRS_MSG_DCINFOREPLY	145
4.1.5.1.4	DRS_MSG_DCINFOREPLY_V1	146
4.1.5.1.5	DRS_MSG_DCINFOREPLY_V2	146
4.1.5.1.6	DRS_MSG_DCINFOREPLY_V3	146
4.1.5.1.7	DRS_MSG_DCINFOREPLY_VFFFFFFF	147
4.1.5.1.8	DS_DOMAIN_CONTROLLER_INFO_1W	147
4.1.5.1.9	DS_DOMAIN_CONTROLLER_INFO_2W	147
4.1.5.1.10	DS_DOMAIN_CONTROLLER_INFO_3W	148
4.1.5.1.11	DS_DOMAIN_CONTROLLER_INFO_FFFFFFFW	149
4.1.5.2	Server Behavior of the IDL_DRSDomainControllerInfo Method.....	150
4.1.5.3	Examples of the IDL_DRSDomainControllerInfo Method.....	155
4.1.5.3.1	Initial State	155
4.1.5.3.2	Client Request	159
4.1.5.3.3	Server Response	159
4.1.5.3.4	Final State.....	160
4.1.6	IDL_DRSExecuteKCC (Opnum 18)	160
4.1.6.1	Method-Specific Concrete Types	160
4.1.6.1.1	DRS_MSG_KCC_EXECUTE	160
4.1.6.1.2	DRS_MSG_KCC_EXECUTE_V1	161
4.1.6.2	Method-Specific Abstract Types and Procedures	161
4.1.6.2.1	ExecuteKCCTasks.....	161

4.1.6.3	Server Behavior of the IDL_DRSExecuteKCC Method	161
4.1.7	IDL_DRSFinishDemotion (Opnum 27)	162
4.1.7.1	Method-Specific Concrete Types	163
4.1.7.1.1	DRS_MSG_FINISH_DEMOTIONREQ	163
4.1.7.1.2	DRS_MSG_FINISH_DEMOTIONREQ_V1	163
4.1.7.1.3	DRS_MSG_FINISH_DEMOTIONREPLY	164
4.1.7.1.4	DRS_MSG_FINISH_DEMOTIONREPLY_V1	164
4.1.7.2	Method-Specific Abstract Types and Procedures	165
4.1.7.2.1	RemoveADLDSServer	165
4.1.7.2.2	RemoveADLDSSCP	165
4.1.7.2.3	RemoveADLDSSPNs	165
4.1.7.3	Server Behavior of the IDL_DRSFinishDemotion Method	165
4.1.8	IDL_DRSGetMemberships (Opnum 9)	168
4.1.8.1	Method-Specific Concrete Types	168
4.1.8.1.1	DRS_MSG_REVMEMB_REQ	168
4.1.8.1.2	DRS_MSG_REVMEMB_REQ_V1	169
4.1.8.1.3	REVERSE_MEMBERSHIP_OPERATION_TYPE	169
4.1.8.1.4	DRS_MSG_REVMEMB_REPLY	170
4.1.8.1.5	DRS_MSG_REVMEMB_REPLY_V1	170
4.1.8.1.6	SE_GROUP Values	171
4.1.8.2	Method-Specific Abstract Types and Procedures	171
4.1.8.2.1	Arc and ArcSet	171
4.1.8.2.2	Closure	171
4.1.8.2.3	DomainOf	172
4.1.8.2.4	GetDSNameOfEnterpriseRODCsGroup	172
4.1.8.2.5	GetDSNameFromPrimaryGroupId	172
4.1.8.2.6	IsMatchedGroup	172
4.1.8.2.7	Neighbors	173
4.1.8.3	Server Behavior of the IDL_DRSGetMemberships Method	173
4.1.9	IDL_DRSGetMemberships2 (Opnum 21)	176
4.1.9.1	Method-Specific Concrete Types	176
4.1.9.1.1	DRS_MSG_GETMEMBERSHIPS2_REQ	176
4.1.9.1.2	DRS_MSG_GETMEMBERSHIPS2_REQ_V1	177
4.1.9.1.3	DRS_MSG_GETMEMBERSHIPS2_REPLY	177
4.1.9.1.4	DRS_MSG_GETMEMBERSHIPS2_REPLY_V1	177
4.1.9.2	Server Behavior of the IDL_DRSGetMemberships2 Method	178
4.1.10	IDL_DRSGetNCChanges (Opnum 3)	178
4.1.10.1	Overview	179
4.1.10.1.1	Cycle Start and Finish	179
4.1.10.1.2	Cycle Goal	180
4.1.10.1.3	Extended Operations	181
4.1.10.2	Method-Specific Concrete Types	181
4.1.10.2.1	DRS_MSG_GETCHGREQ	181
4.1.10.2.2	DRS_MSG_GETCHGREQ_V3	182
4.1.10.2.3	DRS_MSG_GETCHGREQ_V4	182
4.1.10.2.4	DRS_MSG_GETCHGREQ_V5	183
4.1.10.2.5	DRS_MSG_GETCHGREQ_V7	183
4.1.10.2.6	DRS_MSG_GETCHGREQ_V8	184
4.1.10.2.7	DRS_MSG_GETCHGREQ_V10	185
4.1.10.2.8	DRS_MSG_GETCHGREPLY	186
4.1.10.2.9	DRS_MSG_GETCHGREPLY_V1	186
4.1.10.2.10	DRS_MSG_GETCHGREPLY_V2	187
4.1.10.2.11	DRS_MSG_GETCHGREPLY_V6	187

4.1.10.2.12	DRS_MSG_GETCHGREPLY_V7	188
4.1.10.2.13	COMPRESSED_DATA	189
4.1.10.2.14	DRS_COMP_ALG_TYPE	189
4.1.10.2.15	DRS_COMPRESSED_BLOB	190
4.1.10.2.16	ENCRYPTED_PAYLOAD	190
4.1.10.2.17	EXOP_ERR Codes.....	191
4.1.10.2.18	EXOP_REQ Codes	191
4.1.10.2.19	PROPERTY_META_DATA.....	191
4.1.10.3	Method-Specific Abstract Types and Procedures.....	192
4.1.10.3.1	AbstractLinkValStampFromConcreteLinkValStamp	192
4.1.10.3.2	AbstractPASFromConcretePAS.....	192
4.1.10.3.3	AbstractUTDFromConcreteUTD.....	192
4.1.10.3.4	AttributeAndStamp	193
4.1.10.3.5	AttributeStampCompare.....	193
4.1.10.3.6	ConcretePASFromAbstractPAS.....	193
4.1.10.3.7	ConcreteUTDFromAbstractUTD.....	194
4.1.10.3.8	GetNCChangesV6Reply	194
4.1.10.3.9	GetStampsForUpdate.....	195
4.1.10.3.10	GetWellKnownObject.....	195
4.1.10.3.11	IsProtectedObject	196
4.1.10.3.12	IsSecretAttribute	196
4.1.10.3.13	IsUserIncluded	196
4.1.10.3.14	LinkValueStampCompare.....	196
4.1.10.3.15	ObjAtts.....	197
4.1.10.3.16	ObjAttVal.....	197
4.1.10.3.17	PerformModifyDNOperation	197
4.1.10.3.18	RemoveAttrVal	198
4.1.10.3.19	SetAttrStamp	198
4.1.10.3.20	SetAttrVal.....	198
4.1.10.3.21	SetLinkStamp	198
4.1.10.4	Client Behavior When Sending the IDL_DRSGetNCChanges Request.....	198
4.1.10.4.1	ReplicateNCRequestMsg	199
4.1.10.4.2	ReplSingleObjRequestMsg	202
4.1.10.4.3	PerformExtendedOpRequestMsg	204
4.1.10.5	Server Behavior of the IDL_DRSGetNCChanges Method	206
4.1.10.5.1	TransformInput.....	209
4.1.10.5.2	GetReplChanges.....	211
4.1.10.5.3	GetReplScope	214
4.1.10.5.4	ObjectMatchesSearchFilter.....	215
4.1.10.5.5	GetChangesInScope.....	215
4.1.10.5.6	FilterAttribute	217
4.1.10.5.7	GetResponseSubset	218
4.1.10.5.8	AddObjToResponse.....	219
4.1.10.5.9	UpdateRevealedList	221
4.1.10.5.10	AddLinkToResponse	222
4.1.10.5.11	EncryptValuesIfNecessary	223
4.1.10.5.12	ProcessFsmoRoleRequest.....	224
4.1.10.5.13	RevealSecretsPolicy	228
4.1.10.5.14	GetRevealSecretsPolicyForUser.....	228
4.1.10.5.15	RevealSecretsForUserAllowed.....	229
4.1.10.5.16	GetRoleScope.....	230
4.1.10.5.17	SortResponseLinks.....	231
4.1.10.5.18	TransformOutput	232

4.1.10.5.19	CompressOrDecompressWin2k3	234
4.1.10.5.19.1	LZ77 Compression Algorithm	234
4.1.10.5.19.2	DIRECT2 Encoding Algorithm	236
4.1.10.6	Client Behavior When Receiving the IDL_DRSGetNCChanges Response.....	240
4.1.10.6.1	ProcessGetNCChangesReply.....	240
4.1.10.6.2	EnableRecycleBin	243
4.1.10.6.3	PrepareCrossNCMove	243
4.1.10.6.4	AdjustInstanceTypeAttrVal	245
4.1.10.6.5	SetResetInstanceTypeBits	246
4.1.10.6.6	PerformModifyOperation	246
4.1.10.6.7	NameObject.....	247
4.1.10.6.8	AddObject	248
4.1.10.6.9	UpdateObject.....	250
4.1.10.6.10	FindBestParentObject.....	252
4.1.10.6.11	ResolveNameConflict.....	253
4.1.10.6.12	MakeConflictDN	254
4.1.10.6.13	ProcessLinkValue	255
4.1.10.6.14	UpdateRepsFrom	257
4.1.10.6.15	UpdateUTDandPAS.....	258
4.1.10.6.16	DecryptValuesIfNecessary	259
4.1.10.6.17	DecompressReplyMessage	261
4.1.10.6.18	DecompressMessage	262
4.1.10.7	Examples of the IDL_DRSGetNCChanges Method - Add User	263
4.1.10.7.1	Initial State	263
4.1.10.7.2	Client Request.....	265
4.1.10.7.3	Server Response	266
4.1.10.7.4	Final State	268
4.1.10.8	Examples of the IDL_DRSGetNCChanges Method - Add User to a Group	269
4.1.10.8.1	Initial State	269
4.1.10.8.2	Client Request.....	271
4.1.10.8.3	Server Response	272
4.1.10.8.4	Final State.....	273
4.1.10.9	Examples of the IDL_DRSGetNCChanges Method - Change User Password	274
4.1.10.9.1	Initial State	274
4.1.10.9.2	Client Request.....	277
4.1.10.9.3	Server Response	278
4.1.10.9.4	Final State.....	279
4.1.11	IDL_DRSGetNT4ChangeLog (Opnum 11)	280
4.1.11.1	Method-Specific Concrete Types	281
4.1.11.1.1	DRS_MSG_NT4_CHGLOG_REQ.....	281
4.1.11.1.2	DRS_MSG_NT4_CHGLOG_REQ_V1	281
4.1.11.1.3	DRS_MSG_NT4_CHGLOG_REPLY	282
4.1.11.1.4	DRS_MSG_NT4_CHGLOG_REPLY_V1.....	282
4.1.11.1.5	NT4_REPLICATION_STATE.....	283
4.1.11.2	Method-Specific Abstract Types and Procedures.....	283
4.1.11.2.1	IsPDC	283
4.1.11.2.2	GetWindowsErrorCode	283
4.1.11.3	Server Behavior of the IDL_DRSGetNT4ChangeLog Method	284
4.1.11.4	Examples of the IDL_DRSGetNT4ChangeLog Method	287
4.1.11.4.1	Initial State	287
4.1.11.4.2	Client Request.....	287
4.1.11.4.3	Server Response	288
4.1.11.4.4	Final State.....	288

4.1.12	IDL_DRSGetObjectExistence (Opnum 23)	288
4.1.12.1	Method-Specific Concrete Types	289
4.1.12.1.1	DRS_MSG_EXISTREQ	289
4.1.12.1.2	DRS_MSG_EXISTREQ_V1	289
4.1.12.1.3	DRS_MSG_EXISTREPLY	289
4.1.12.1.4	DRS_MSG_EXISTREPLY_V1	290
4.1.12.2	Method-Specific Abstract Types and Procedures	290
4.1.12.2.1	GuidSequence	290
4.1.12.3	Client Behavior When Sending the IDL_DRSGetObjectExistence Request	291
4.1.12.4	Server Behavior of the IDL_DRSGetObjectExistence Method	292
4.1.12.5	Client Behavior When Receiving the IDL_DRSGetObjectExistence Response	293
4.1.13	IDL_DRSGetReplInfo (Opnum 19)	293
4.1.13.1	Method-Specific Concrete Types	294
4.1.13.1.1	DRS_MSG_GETREPLINFO_REQ	294
4.1.13.1.2	DRS_MSG_GETREPLINFO_REQ_V1	294
4.1.13.1.3	DRS_MSG_GETREPLINFO_REQ_V2	295
4.1.13.1.4	DS_REPL_INFO Codes	295
4.1.13.1.5	DRS_MSG_GETREPLINFO_REPLY	296
4.1.13.1.6	DS_REPL_NEIGHBORSW	298
4.1.13.1.7	DS_REPL_NEIGHBORW	298
4.1.13.1.8	DS_REPL_CURSORS	299
4.1.13.1.9	DS_REPL_CURSOR	299
4.1.13.1.10	DS_REPL_CURSORS_2	300
4.1.13.1.11	DS_REPL_CURSOR_2	300
4.1.13.1.12	DS_REPL_CURSORS_3W	300
4.1.13.1.13	DS_REPL_CURSOR_3W	301
4.1.13.1.14	DS_REPL_OBJ_META_DATA	301
4.1.13.1.15	DS_REPL_ATTR_META_DATA	302
4.1.13.1.16	DS_REPL_OBJ_META_DATA_2	302
4.1.13.1.17	DS_REPL_ATTR_META_DATA_2	303
4.1.13.1.18	DS_REPL_KCC_DSA_FAILURESW	303
4.1.13.1.19	DS_REPL_KCC_DSA_FAILUREW	304
4.1.13.1.20	DS_REPL_PENDING_OPSW	304
4.1.13.1.21	DS_REPL_OPW	304
4.1.13.1.22	DS_REPL_ATTR_VALUE_META_DATA	305
4.1.13.1.23	DS_REPL_VALUE_META_DATA	305
4.1.13.1.24	DS_REPL_ATTR_VALUE_META_DATA_2	306
4.1.13.1.25	DS_REPL_VALUE_META_DATA_2	307
4.1.13.1.26	DS_REPL_CLIENT_CONTEXTS	308
4.1.13.1.27	DS_REPL_CLIENT_CONTEXT	308
4.1.13.1.28	DS_REPL_SERVER_OUTGOING_CALLS	309
4.1.13.1.29	DS_REPL_SERVER_OUTGOING_CALL	309
4.1.13.2	Method-Specific Abstract Types and Procedures	310
4.1.13.2.1	GetDNFromInvocationID	310
4.1.13.2.2	GetDNFromObjectGuid	310
4.1.13.2.3	GetNCs	311
4.1.13.2.4	GetUpToDatenessVector	311
4.1.13.3	Server Behavior of the IDL_DRSGetReplInfo Method	311
4.1.13.4	Examples of the IDL_DRSGetReplInfo Method	323
4.1.13.4.1	Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_NEIGHBORS to find replication neighbors for a specified NC	323
4.1.13.4.1.1	Initial State	323
4.1.13.4.1.2	Client Request	324

4.1.13.4.1.3	Server Response.....	324
4.1.13.4.1.4	Final State	325
4.1.13.4.2	Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_NEIGHBORS to find which naming contexts a DC receives updates for from a replication neighbor.....	325
4.1.13.4.2.1	Initial State.....	325
4.1.13.4.2.2	Client Request.....	327
4.1.13.4.2.3	Server Response.....	327
4.1.13.4.2.4	Final State	329
4.1.13.4.3	Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_REPSTO to find replication neighbors for a specified NC.....	329
4.1.13.4.3.1	Initial State.....	329
4.1.13.4.3.2	Client Request.....	330
4.1.13.4.3.3	Server Response.....	330
4.1.13.4.3.4	Final State	331
4.1.13.4.4	Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_CURSORS_3_FOR_NC.....	331
4.1.13.4.4.1	Initial State.....	331
4.1.13.4.4.2	Client Request.....	332
4.1.13.4.4.3	Server Response.....	332
4.1.13.4.4.4	Final State	333
4.1.13.4.5	Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_METADATA_2_FOR_OBJ.....	333
4.1.13.4.5.1	Initial State.....	333
4.1.13.4.5.2	Client Request.....	334
4.1.13.4.5.3	Server Response.....	334
4.1.13.4.5.4	Final State	337
4.1.13.4.6	Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE to view the replication metadata for all values of a link value attribute.....	337
4.1.13.4.6.1	Initial State.....	337
4.1.13.4.6.2	Client Request.....	337
4.1.13.4.6.3	Server Response.....	337
4.1.13.4.6.4	Final State	339
4.1.13.4.7	Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE to view the replication metadata for a specific value of a link value attribute.....	339
4.1.13.4.7.1	Initial State.....	340
4.1.13.4.7.2	Client Request.....	340
4.1.13.4.7.3	Server Response.....	340
4.1.13.4.7.4	Final State	341
4.1.13.4.8	Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES.....	341
4.1.13.4.8.1	Initial State.....	341
4.1.13.4.8.2	Client Request.....	341
4.1.13.4.8.3	Server Response.....	342
4.1.13.4.8.4	Final State	342
4.1.13.4.9	Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_PENDING_OPS.....	342
4.1.13.4.9.1	Initial State.....	342
4.1.13.4.9.2	Client Request.....	343
4.1.13.4.9.3	Server Response.....	343
4.1.13.4.9.4	Final State	344

4.1.14	IDL_DRSInitDemotion (Opnum 25)	344
4.1.14.1	Method-Specific Concrete Types	344
4.1.14.1.1	DRS_MSG_INIT_DEMOTIONREQ	344
4.1.14.1.2	DRS_MSG_INIT_DEMOTIONREQ_V1	345
4.1.14.1.3	DRS_MSG_INIT_DEMOTIONREPLY	345
4.1.14.1.4	DRS_MSG_INIT_DEMOTIONREPLY_V1	345
4.1.14.2	Server Behavior of the IDL_DRSInitDemotion Method	345
4.1.15	IDL_DRSInterDomainMove (Opnum 10)	346
4.1.15.1	Method-Specific Concrete Types	347
4.1.15.1.1	DRS_MSG_MOVEREQ	347
4.1.15.1.2	DRS_MSG_MOVEREQ_V1	347
4.1.15.1.3	DRS_MSG_MOVEREQ_V2	348
4.1.15.1.4	DRS_MSG_MOVEREPLY	348
4.1.15.1.5	DRS_MSG_MOVEREPLY_V1	348
4.1.15.1.6	DRS_MSG_MOVEREPLY_V2	349
4.1.15.2	Method-Specific Abstract Types and Procedures	349
4.1.15.2.1	AttrIsBacklink	349
4.1.15.2.2	AttrIsConstructed	349
4.1.15.2.3	AttrIsNonReplicated	350
4.1.15.2.4	AuthorizationInfoFromClientCredentials	350
4.1.15.2.5	ImpersonateAuthorizationInfo	350
4.1.15.2.6	IsApplicationNC	350
4.1.15.2.7	RevertToSelf	350
4.1.15.3	Server Behavior of the IDL_DRSInterDomainMove Method	350
4.1.15.4	Examples of the IDL_DRSInterDomainMove Method	354
4.1.15.4.1	Initial State	354
4.1.15.4.2	Client Request	355
4.1.15.4.3	Server Response	357
4.1.15.4.4	Final State	357
4.1.16	IDL_DRSQuerySitesByCost (Opnum 24)	359
4.1.16.1	Method-Specific Concrete Types	359
4.1.16.1.1	DRS_MSG_QUERYSITESREQ	359
4.1.16.1.2	DRS_MSG_QUERYSITESREQ_V1	359
4.1.16.1.3	DRS_MSG_QUERYSITESREPLY	360
4.1.16.1.4	DRS_MSG_QUERYSITESREPLY_V1	360
4.1.16.1.5	DRS_MSG_QUERYSITESREPLYELEMENT_V1	360
4.1.16.2	Method-Specific Abstract Types and Procedures	361
4.1.16.2.1	ValidateSiteRDN	361
4.1.16.2.2	WeightedArc and WeightedArcSet	361
4.1.16.2.3	MinWeightPath	361
4.1.16.3	Server Behavior of the IDL_DRSQuerySitesByCost Method	362
4.1.16.4	Examples of IDL_DRSQuerySitesByCost Method	365
4.1.16.4.1	Nontransitive Communication Using siteLinkBridge	365
4.1.16.4.1.1	Initial State	366
4.1.16.4.1.2	Client Request	371
4.1.16.4.1.3	Server Response	372
4.1.16.4.1.4	Final State	372
4.1.16.4.2	Transitive Communication	372
4.1.16.4.2.1	Initial State	373
4.1.16.4.2.2	Client Request	378
4.1.16.4.2.3	Server Response	379
4.1.16.4.2.4	Final State	379
4.1.17	IDL_DRSRemoveDsDomain (Opnum 15)	379

4.1.17.1	Method-Specific Concrete Types	380
4.1.17.1.1	DRS_MSG_RMDMNREQ	380
4.1.17.1.2	DRS_MSG_RMDMNREQ_V1	380
4.1.17.1.3	DRS_MSG_RMDMNREPLY	380
4.1.17.1.4	DRS_MSG_RMDMNREPLY_V1	381
4.1.17.2	Method-Specific Abstract Types and Procedures	381
4.1.17.2.1	HasNCReplicated	381
4.1.17.3	Server Behavior of the IDL_DRSRemoveDsDomain Method	381
4.1.18	IDL_DRSRemoveDsServer (Opnum 14)	383
4.1.18.1	Method-Specific Concrete Types	383
4.1.18.1.1	DRS_MSG_RMSVRREQ	383
4.1.18.1.2	DRS_MSG_RMSVRREQ_V1	384
4.1.18.1.3	DRS_MSG_RMSVRREPLY	384
4.1.18.1.4	DRS_MSG_RMSVRREPLY_V1	384
4.1.18.2	Server Behavior of the IDL_DRSRemoveDsServer Method	385
4.1.19	IDL_DRSReplicaAdd (Opnum 5)	387
4.1.19.1	Method-Specific Concrete Types	388
4.1.19.1.1	DRS_MSG_REPADD	388
4.1.19.1.2	DRS_MSG_REPADD_V1	388
4.1.19.1.3	DRS_MSG_REPADD_V2	388
4.1.19.2	Server Behavior of the IDL_DRSReplicaAdd Method	389
4.1.20	IDL_DRSReplicaDel (Opnum 6)	392
4.1.20.1	Method-Specific Concrete Types	392
4.1.20.1.1	DRS_MSG_REPDEL	392
4.1.20.1.2	DRS_MSG_REPDEL_V1	393
4.1.20.2	Server Behavior of the IDL_DRSReplicaDel Method	393
4.1.21	IDL_DRSReplicaDemotion (Opnum 26)	396
4.1.21.1	Method-Specific Concrete Types	397
4.1.21.1.1	DRS_MSG_REPLICA_DEMOTIONREQ	397
4.1.21.1.2	DRS_MSG_REPLICA_DEMOTIONREQ_V1	397
4.1.21.1.3	DRS_MSG_REPLICA_DEMOTIONREPLY	397
4.1.21.1.4	DRS_MSG_REPLICA_DEMOTIONREPLY_V1	398
4.1.21.2	Method-Specific Abstract Types and Procedures	398
4.1.21.2.1	ReplicationPartners()	398
4.1.21.2.2	AbandonAllFSMORoles()	398
4.1.21.2.3	ReplicateOffChanges()	399
4.1.21.3	Server Behavior of the IDL_DRSReplicaDemotion Method	400
4.1.22	IDL_DRSReplicaModify (Opnum 7)	401
4.1.22.1	Method-Specific Concrete Types	402
4.1.22.1.1	DRS_MSG_REPMOD	402
4.1.22.1.2	DRS_MSG_REPMOD_V1	402
4.1.22.2	Server Behavior of the IDL_DRSReplicaModify Method	403
4.1.23	IDL_DRSReplicaSync (Opnum 2)	404
4.1.23.1	Method-Specific Concrete Types	405
4.1.23.1.1	DRS_MSG_REPSYNC	405
4.1.23.1.2	DRS_MSG_REPSYNC_V1	405
4.1.23.2	Server Behavior of the IDL_DRSReplicaSync Method	406
4.1.24	IDL_DRSReplicaVerifyObjects (Opnum 22)	407
4.1.24.1	Method-Specific Concrete Types	407
4.1.24.1.1	DRS_MSG_REPVERIFYOBJ	407
4.1.24.1.2	DRS_MSG_REPVERIFYOBJ_V1	408
4.1.24.2	Method-Specific Abstract Types and Procedures	408
4.1.24.2.1	GetRemoteUTD	408

4.1.24.2.2	ObjectExistsAtDC	408
4.1.24.3	Server Behavior of the IDL_DRSReplicaVerifyObjects Method	409
4.1.24.4	Examples of the IDL_DRSReplicaVerifyObjects Method	410
4.1.24.4.1	Initial State	410
4.1.24.4.2	Client Request.....	414
4.1.24.4.3	Server Response	414
4.1.24.4.4	Final State.....	414
4.1.25	IDL_DRSUnbind (Opnum 1)	415
4.1.25.1	Server Behavior of the IDL_DRSUnbind Method	415
4.1.26	IDL_DRSUpdateRefs (Opnum 4)	415
4.1.26.1	Method-Specific Concrete Types	416
4.1.26.1.1	DRS_MSG_UPDREFS	416
4.1.26.1.2	DRS_MSG_UPDREFS_V1	416
4.1.26.2	Server Behavior of the IDL_DRSUpdateRefs Method	417
4.1.26.3	Examples of the IDL_DRSUpdateRefs Method	418
4.1.26.3.1	Adding a repsTo Entry	418
4.1.26.3.1.1	Initial State.....	418
4.1.26.3.1.2	Client Request.....	418
4.1.26.3.1.3	Server Response.....	418
4.1.26.3.1.4	Final State	419
4.1.26.3.2	Replacing a repsTo Entry	419
4.1.26.3.2.1	Initial State.....	419
4.1.26.3.2.2	Client Request.....	420
4.1.26.3.2.3	Server Response.....	420
4.1.26.3.2.4	Final State	420
4.1.27	IDL_DRSVerifyNames (Opnum 8)	421
4.1.27.1	Method-Specific Concrete Types	422
4.1.27.1.1	DRS_MSG_VERIFYREQ	422
4.1.27.1.2	DRS_MSG_VERIFYREQ_V1	422
4.1.27.1.3	DRS_MSG_VERIFYREPLY	423
4.1.27.1.4	DRS_MSG_VERIFYREPLY_V1	423
4.1.27.2	Server Behavior of the IDL_DRSVerifyNames Method	424
4.1.27.3	Examples of the IDL_DRSVerifyNames Method	427
4.1.27.3.1	Initial State	427
4.1.27.3.2	Client Request.....	427
4.1.27.3.3	Server Response	428
4.1.27.3.4	Final State.....	428
4.1.28	IDL_DRSWriteSPN (Opnum 13)	428
4.1.28.1	Method-Specific Concrete Types	429
4.1.28.1.1	DRS_MSG_SPNREQ	429
4.1.28.1.2	DRS_MSG_SPNREQ_V1	429
4.1.28.1.3	DRS_MSG_SPNREPLY.....	430
4.1.28.1.4	DRS_MSG_SPNREPLY_V1	430
4.1.28.1.5	DS_SPN_OPERATION	430
4.1.28.2	Method-Specific Abstract Types and Procedures.....	431
4.1.28.2.1	ExecuteWriteSPNRemotely	431
4.1.28.3	Server Behavior of the IDL_DRSWriteSPN Method.....	431
4.2	dsaop RPC Interface.....	433
4.2.1	IDL_DSAPrepareScript (Opnum 0)	434
4.2.1.1	Method-Specific Concrete Types	434
4.2.1.1.1	DSA_MSG_PREPARE_SCRIPT_REQ	434
4.2.1.1.2	DSA_MSG_PREPARE_SCRIPT_REQ_V1	434
4.2.1.1.3	DSA_MSG_PREPARE_SCRIPT_REPLY.....	435

4.2.1.1.4	DSA_MSG_PREPARE_SCRIPT_REPLY_V1	435
4.2.1.2	Method-Specific Abstract Types and Procedures	435
4.2.1.2.1	GetKeyLengthHandleT	435
4.2.1.2.2	PrepareScriptInProgress	436
4.2.1.2.3	PrepareScriptVerifyScript	436
4.2.1.2.4	PrepareScriptHashBody	436
4.2.1.2.5	PrepareScriptHashSignature	436
4.2.1.2.6	PrepareScriptGeneratePassword	436
4.2.1.3	Server Behavior of the IDL_DSAPrepareScript Method	436
4.2.2	IDL_DSASExecuteScript (Opnum 1)	438
4.2.2.1	Method-Specific Concrete Types	439
4.2.2.1.1	DSA_MSG_EXECUTE_SCRIPT_REQ	439
4.2.2.1.2	DSA_MSG_EXECUTE_SCRIPT_REQ_V1	439
4.2.2.1.3	DSA_MSG_EXECUTE_SCRIPT_REPLY	439
4.2.2.1.4	DSA_MSG_EXECUTE_SCRIPT_REPLY_V1	440
4.2.2.2	Method-Specific Abstract Types and Procedures	440
4.2.2.2.1	ExecuteScriptInProgress	440
4.2.2.2.2	ExecuteScript	440
4.2.2.3	Server Behavior of the IDL_DSASExecuteScript Method	440
5	Common Data Types, Variables, and Procedures	442
5.1	AbstractPTFromConcretePT	442
5.2	AccessCheckAttr	442
5.3	AccessCheckCAR	443
5.4	AccessCheckObject	443
5.5	AccessCheckWriteToSpnAttribute	443
5.6	AddSubRef	444
5.7	AmIRODC	445
5.8	AmILHServer	445
5.9	ATTR	446
5.10	ATTRBLOCK	446
5.11	AttributeStamp	446
5.12	AttributeSyntax	448
5.13	AttrStamp	448
5.14	ATTRTYP	448
5.15	AttrtypFromSchemaObj	448
5.16	ATTRVAL	449
5.16.1	Concrete Value Representations	449
5.16.1.1	INT32	449
5.16.1.2	INT64	449
5.16.1.3	OctetString	450
5.16.1.4	String8	450
5.16.1.5	String16	450
5.16.1.6	SECURITY_DESCRIPTOR	450
5.16.1.7	SID	450
5.16.2	Abstract Value Representations	450
5.16.2.1	Object(DS-DN)	451
5.16.2.2	Object(DN-String)	452
5.16.2.3	Object(DN-Binary)	452
5.16.2.4	Object(Access-Point)	452
5.16.2.5	Object(OR-Name)	453
5.16.2.6	String(NT-Sec-Desc)	453
5.16.2.7	String(Sid)	453

5.16.2.8	String(Teletex)	453
5.16.3	Converting Between Abstract and Concrete Value Representations	453
5.16.3.1	Boolean	455
5.16.3.2	Enumeration and Integer	455
5.16.3.3	LargeInteger	456
5.16.3.4	Object(Presentation-Address).....	456
5.16.3.5	Object(Replica-Link) String (Octet)	456
5.16.3.6	String(IA5) String(Printable) String(Numeric) String(Teletex).....	456
5.16.3.7	String(Unicode)	457
5.16.3.8	String(Object-Identifier)	457
5.16.3.9	String(UTC-Time) and String(Generalized-Time).....	457
5.16.3.10	Object(DS-DN)	458
5.16.3.11	Object(DN-Binary)	459
5.16.3.12	Object(DN-String)	460
5.16.3.13	Object(OR-Name)	461
5.16.3.14	Object(Access-Point)	461
5.16.3.15	String(Sid)	461
5.16.3.16	String(NT-Sec-Desc)	462
5.16.4	ATTRTYP-to-OID Conversion	463
5.17	ATTRVALBLOCK	469
5.18	ATTRVALFromValue	469
5.19	BindToDSA()	469
5.20	BOOL	469
5.21	BYTE	470
5.22	CHANGE_LOG_ENTRIES.....	470
5.23	CHANGELOG_ENTRY.....	470
5.24	CheckGroupMembership	470
5.25	ClientAuthorizationInfo	470
5.26	ClientExtensions	470
5.27	ClientUUID	471
5.28	ConcretePTFromAbstractPT	471
5.29	ConfigNC	471
5.30	dc, DC	471
5.31	DefaultNC	472
5.32	DelSubRef.....	472
5.33	DescendantObject.....	473
5.34	DN	473
5.35	DNBinary	473
5.36	DomainNameFromNT4AccountName.....	473
5.37	DRS_EXTENSIONS	474
5.38	DRS_EXTENSIONS_INT	474
5.39	DRS_HANDLE	478
5.40	DRS_OPTIONS.....	478
5.41	DRS_MORE_GETCHGREQ_OPTIONS	480
5.42	DRS_SecBuffer	481
5.43	DRS_SecBufferDesc	482
5.44	DRS_SPN_CLASS	483
5.45	DS_REPL_OP_TYPE	483
5.46	DSABObj	483
5.47	DSA_RPC_INST	483
5.48	DSName	484
5.49	DSNAME	484
5.49.1	DSNAME Equality.....	486

5.50	DSTIME	486
5.51	DWORD	487
5.52	ENTINF	487
5.53	ENTINF_GetValue	488
5.54	ENTINF_SetValue.....	488
5.55	ENTINF_EnumerateAttributes	488
5.56	ENTINFLIST.....	488
5.57	Expunge	489
5.58	FILETIME	489
5.59	FilteredGCPAS	489
5.60	FilteredPAS	490
5.61	FindChar	490
5.62	FindCharRev.....	491
5.63	FOREST_TRUST_INFORMATION	491
5.63.1	Record	492
5.63.2	Determining If a Name Is In a Trusted Forest	495
5.64	FOREST_TRUST_RECORD_TYPE	502
5.65	ForestRootDomainNC	502
5.66	FullReplicaExists	502
5.67	GCPAS.....	503
5.68	GetFilteredAttributeSet	503
5.69	GetNCType.....	504
5.70	GetAttrVals	505
5.71	GetCallerAuthorizationInfo	505
5.72	GetDefaultObjectCategory	505
5.73	GetDSNameFromAttrVal	505
5.74	GetDSNameFromDN	506
5.75	GetDSNameFromNetworkAddress	506
5.76	GetForestFunctionalLevel	506
5.77	GetFSMORoleOwner	506
5.78	GetInstanceNameFromSPN	507
5.79	GetObjectNC	507
5.80	GetProxyEpoch	507
5.81	GetProxyType.....	507
5.82	GetServiceClassFromSPN	507
5.83	GetServiceNameFromSPN	507
5.84	groupType Bit Flags.....	508
5.85	GUID	508
5.86	GuidFromString	508
5.87	GuidToString	509
5.88	handle_t	509
5.89	instanceType Bit Flags	509
5.90	Is2PartSPN.....	509
5.91	Is3PartSPN.....	510
5.92	IsAdlds	510
5.93	IsBuiltinPrincipal	510
5.94	IsDomainNameInTrustedForest.....	510
5.95	IsDomainSidInTrustedForest	510
5.96	IsDCAccount	511
5.97	IsForwardLinkAttribute	511
5.98	IsGC.....	511
5.99	IsGetNCChangesPermissionGranted	511
5.100	IsGUIDBasedDNSName	512

5.101	IsMemberOfBuiltinAdminGroup	512
5.102	IsRecycleBinEnabled	512
5.103	IsRevealFilteredAttribute	513
5.104	IsRevealSecretRequest	513
5.105	IsServerExtensionsChanged	514
5.106	IsUPNInTrustedForest	514
5.107	IsValidServiceName	515
5.108	KCCFailedConnections	515
5.109	KCCFailedLinks	515
5.110	LARGE_INTEGER	515
5.111	LDAP_CONN_PROPERTIES	515
5.112	LDAP_SERVER_DIRSYNC_OID LDAP Search Control	516
5.112.1	Abstract Types	516
5.112.1.1	AttributeList	516
5.112.1.2	AttributeListElement	517
5.112.1.3	AttributeVals	517
5.112.1.4	Control	517
5.112.1.5	DirSyncControlValue	517
5.112.1.6	DirSyncSearchArg	518
5.112.1.7	LDAPString	518
5.112.1.8	SearchResultEntry	518
5.112.1.9	SearchResultEntryList	518
5.112.2	Concrete Types	519
5.112.2.1	Cookie	519
5.112.3	ProcessDirSyncSearchRequest	519
5.112.4	DirSyncReqToGetChgReq	520
5.112.5	GetChgReplyToSearchResult	521
5.112.6	TransformEntinfToSearchEntry	523
5.112.7	TransformRepIValInfListToSearchEntry	524
5.112.8	TransformDSNameToLdapDN	525
5.112.9	LDAPDisplayNameFromAttrTyp	525
5.112.10	GetResponseDirSyncControlValue	525
5.112.11	GetUsnUtdVectorFromCookie	527
5.112.12	SecurityCheckForChanges	528
5.112.13	IsFilteredAttributePresent	528
5.113	LDAPConnections	529
5.114	LinkStamp	529
5.115	LinkValueStamp	530
5.116	LocalAttidFromRemoteAttid	530
5.117	LONG	531
5.118	LONGLONG	531
5.119	LPWSTR	531
5.120	MakeAttid	531
5.121	MakeProxyValue	531
5.122	MasterReplicaExists	531
5.123	MD5_CTX	531
5.124	MD5Final	532
5.125	MD5Init	532
5.126	MD5Update	532
5.127	MergeUTD	532
5.128	MTX_ADDR	532
5.129	NCType Bits	533
5.130	NetworkAddress	533

5.131	NewPrefixTable	534
5.132	Nt4ReplicationState	534
5.133	NT4SID	534
5.134	NTSAPI_CLIENT_GUID	534
5.135	NTDSTRANSPORT_OPT Values	535
5.136	NULLGUID	535
5.137	ObjExists	535
5.138	OID	535
5.139	OID_t	535
5.140	OidFromAttid	535
5.141	parent	536
5.142	PARTIAL_ATTR_VECTOR_V1_EXT	536
5.143	partialAttributeSet	536
5.144	PartialGCReplicaExists	536
5.145	PAS_DATA	536
5.146	PdcChangeLog	537
5.147	PerformAddOperation	537
5.148	PerformAddOperationAsSystem	538
5.149	PrefixTable	538
5.150	PrefixTableEntry	538
5.151	PROPERTY_META_DATA_EXT	538
5.152	PROPERTY_META_DATA_EXT_VECTOR	539
5.153	proxiedObjectName Value Format	539
5.154	RDN	539
5.155	rdnType	539
5.156	RecycleObj	540
5.157	RemoveObj	540
5.158	REPLENTINFLIST	540
5.159	ReplicatedAttributes	541
5.160	ReplicationQueue	541
5.161	REPLTIMES	541
5.162	replUpToDateVector, ReplUpToDateVector	542
5.163	REPLVALINF	542
5.164	REPS_FROM	543
5.165	REPS_TO	546
5.166	repsFrom, RepsFrom	549
5.167	repsTo, RepsTo	550
5.168	Rid	551
5.169	Right	551
5.170	RIGHT Values	551
5.171	RPCClientContexts	551
5.172	RPCOutgoingContexts	551
5.173	sAMAccountType Values	552
5.174	SCHEMA_PREFIX_TABLE	552
5.175	SchemaNC	553
5.176	SchemaObj	553
5.177	ServerExtensions	553
5.178	SID	553
5.179	SidFromStringSid	553
5.180	StampLessThanOrEqualUTD	553
5.181	StartsWith	554
5.182	StringSidFromSid	554
5.183	SubString	554

5.184	Syntax.....	554
5.185	SYNTAX_ADDRESS.....	554
5.186	SYNTAX_DISTNAME_BINARY.....	555
5.187	systemFlags Values.....	557
5.188	UCHAR.....	557
5.189	ULARGE_INTEGER.....	557
5.190	ULONG.....	557
5.191	ULONGLONG.....	557
5.192	UndeleteObject.....	557
5.193	UnbindFromDSA().....	557
5.194	UpdateRefs.....	558
5.195	UPTODATE_CURSOR_V1.....	559
5.196	UPTODATE_CURSOR_V2.....	559
5.197	UPTODATE_VECTOR_V1_EXT.....	559
5.198	UPTODATE_VECTOR_V2_EXT.....	560
5.199	userAccountControl Bits.....	560
5.200	UserNameFromNT4AccountName.....	561
5.201	USHORT.....	561
5.202	USN.....	561
5.203	USN_VECTOR.....	561
5.204	UUID.....	562
5.205	ValidateDRSDemotionInput.....	562
5.206	ValidateDRSInput.....	562
5.207	Value.....	563
5.208	VALUE_META_DATA_EXT_V1.....	563
5.209	ValueFromATTRVAL.....	564
5.210	WCHAR.....	564
6	Security.....	565
6.1	Security Considerations for Implementers.....	565
6.2	Index of Security Parameters.....	565
7	Appendix A: Full IDL.....	566
8	Appendix B: Product Behavior.....	594
9	Change Tracking.....	603
10	Index.....	605

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**.

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 Microsoft Windows® 2000 Server operating system and is available in all subsequent server releases. It is not available in Microsoft Windows NT® 3.1 operating system, Microsoft Windows NT® 3.51 operating system, or Microsoft Windows NT® 4.0 operating system.

The Microsoft Windows® client operating systems (Microsoft Windows® 2000 operating system, Windows® XP operating system, Windows Vista® operating system, and Windows® 7 operating system) can implement a client role for some methods, but cannot implement a server role for any methods. The Windows server operating systems (Windows 2000 Server, Windows Server® 2003 operating system, Windows Server® 2003 R2 operating system, Windows Server® 2008 operating system, and Windows Server® 2008 R2 operating system) can implement both client and server roles for all methods. Section [4.1](#) gives details about the methods for which the Windows client operating systems can implement a client role.

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.

Pervasive Concepts

The following concepts are pervasive throughout this specification.

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 \(section 1.1\)](#).

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 more information, see [\[MS-ADTS\]](#) section 3.1.1.1.3). 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", above.

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 Glossary

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

ancestor object
AttributeStamp
authentication
authentication level
back link value
binary large object (BLOB)
canonical name
constructed attribute
container
control access right (CAR)
digest
directory
discretionary access control list (DACL)
domain controller (DC)
domain name (3)
dynamic endpoint
expunge
forest

forward link attribute
 forward link value
 FSMO role owner
 full NC replica
 fully qualified domain name (FQDN) (1) (2)
 garbage collection
 global catalog (GC)
 global catalog server (GC server)
 Interface Definition Language (IDL)
 Internet host name
 LDAP connection
 Lightweight Directory Access Protocol (LDAP)
 link attribute
 link value
 LinkValueStamp
 local domain controller (DC)
 Microsoft Interface Definition Language (MIDL)
 Network Data Representation (NDR)
 non-replicated attribute
 NULL GUID
 opnum
 originating update
 partial attribute set (PAS)
 primary domain controller (PDC) role owner
 prefix table
 remote procedure call (RPC)
 replication latency
 RPC transport
 schema
 security principal
 security provider

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

NetBIOS domain name

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: 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, Windows Server 2003,

and Windows Server 2003 R2 products; in these products it is called "Active Directory". It is also available as part of Windows Server 2008 and Windows Server 2008 R2. AD DS is not present in Windows NT 3.1, Windows NT 3.51, Windows NT 4.0, or Windows XP. For more information, see [MS-WSO](#) section 3.1.2.1.5.2 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" concept that is described in the [Introduction \(section 1\)](#).) 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: The [attributeID](#) **attribute**. 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, and 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 **object identifier (OID)** in a Basic Encoding Rules (BER)-encoded binary format, as specified in [\[ITUX690\]](#) 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 principal: A **security principal** within the **built-in domain** whose **SID** is identical in every **domain**.

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.

child object, children: See the [Introduction \(section 1\)](#).

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: 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** that contains 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 **fully qualified domain name (FQDN)**, 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**.

deleted-object: An object that has been deleted, but that remains in storage until a configured amount of time (the **deleted-object lifetime**) has passed, after which the object is transformed into a **recycled-object**. Unlike a **recycled-object** or a **tombstone**, a **deleted-object** maintains virtually all the state of the object before deletion and may be undeleted without loss of information. **Deleted-objects** exist only when the **Recycle Bin optional feature** is enabled.

deleted-object lifetime: The time period that a **deleted-object** is kept in storage before it is transformed into a **recycled-object**.

directory object (or object): An **Active Directory object**, which is a specialization of the "object" concept that is described in the [Introduction \(section 1\)](#). 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-WSO](#) section 3.1.2.1.2, 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 **relative identifier (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 **fully qualified domain name (FQDN)** component. For more information, see [RFC1034](#). Such a **DN** corresponds to the **FQDN**

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

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

directory service agent (DSA): A term from the X.500 directory specification [\[X501\]](#) that represents a component that maintains and communicates directory information.

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 [5.48](#).

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 **fully qualified domain name (FQDN)** 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 can be 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: The [governsID](#) attribute. An **OID**-valued identifying **attribute** of each [classSchema object](#) in the **schema NC**.

group: See **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 interdomain group membership. See **native mode**.

MSZIP compression algorithm: A compression algorithm specified in [\[RFC1951\]](#) 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 interdomain group membership. See **mixed mode**.

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

nTDSDSA object: An **object** of class [nTDSDSA](#). See **DSA object**.

object: See the [Introduction \(section 1\)](#).

object class: See the [Introduction \(section 1\)](#).

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

objectGUID: The [objectGUID attribute](#). The identifying **attribute** on an **object**, in the sense of the "object" concept that is described in the [Introduction \(section 1\)](#). 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 [objectSid](#) attribute. 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.

optional feature: A non-default behavior that modifies the **Active Directory** state model. For more information, refer to [\[MS-ADTS\]](#) section 3.1.1.9.

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.

parent object: See the [Introduction \(section 1\)](#).

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

Recycle Bin: An **optional feature** that modifies the state model of object deletions and undeletions, making undeletion of **deleted-objects** possible without loss of the object's attribute values. For more information, refer to [\[MS-ADTS\]](#) section 3.1.1.9.1.

recycled-object: An object that has been deleted, but that remains in storage until a configured amount of time (the **tombstone lifetime**) has passed, after which the object is permanently removed from storage. Unlike a **deleted-object**, most of the state of the object has been removed, and the object may no longer be undeleted without loss of information. By keeping the **recycled-object** in existence for the **tombstone lifetime**, the deleted state of the object is able to replicate. **Recycled-objects** exist only when the **Recycle Bin optional feature** is enabled.

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 subauthority 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**.

replica: See the [Introduction \(section 1\)](#).

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 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-WSO](#) section 3.1.2.1.3. 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 class: The first part of a **service principal name**. See [\[MS-KILE\]](#) section 3.1.5.12.

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**, 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**.

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 **Windows error codes**.

target object: An object referenced by a **forward link value**.

subordinate reference object (sub-ref object): The **NC** root of a **parent NC** has a list of all the **NC** roots of its **child NCs** in the [subRefs](#) attribute. Each entry in this list is a subordinate reference and the object named by the entry is denominated a **subordinate reference object**. An object is a **subordinate reference object** if and only if it is in such a list. If a server has **replicas** of both an **NC** and its **child NC**, then the **child NC** root is the **subordinate reference object**, in the context of the **parent NC**. If the server does not have a **replica** of the **child NC**, then another object, with [distinguishedName](#) and [objectGUID](#) attributes equal to the **child NC** root, is present in the server and is the **subordinate reference object**.

tombstone: An object that has been deleted, but remains in storage until a configured amount of time (the **tombstone lifetime**) has passed, after which the object is permanently removed from storage. By keeping the **tombstone** in existence for the **tombstone lifetime**, the deleted state of the object is able to replicate. **Tombstones** exist only when the **Recycle Bin optional feature** is not enabled.

tombstone lifetime: The amount of time that a **tombstone** or **recycled-object** is kept in storage before it is 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** and **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: 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

References to Microsoft Open Specification documents do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

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](#)".

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

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

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

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

[MS-ADTS] Microsoft Corporation, "[Active Directory Technical Specification](#)".

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

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

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

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

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

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

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

[MS-WSO] Microsoft Corporation, "[Windows System Overview](#)".

[RC4] RSA Security, "The RC4 Encryption Algorithm", <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", STD 13, 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>

[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.rfc-editor.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>

[RFC2254] Howes, T., "The String Representation of LDAP Search Filters", RFC 2254, December 1997, <http://www.ietf.org/rfc/rfc2254.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", STD 10, 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>

[X501] ITU-T, "Information Technology - Open Systems Interconnection - The Directory: The Models", Recommendation X.501, August 2005, <http://www.itu.int/rec/T-REC-X.501-200508-I/en>

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.

[MS-ADSO] Microsoft Corporation, "[Active Directory System Overview](#)".

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

[MS-LSAT] Microsoft Corporation, "[Local Security Authority \(Translation Methods\) Remote Protocol Specification](#)".

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

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

1.3 Overview

This document specifies the Directory Replication Service (DRS) 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 dsap. The name of each drsuapi method begins with "IDL_DRS", while the name of each dsap method begins with "IDL_DSA".

1.3.1 Methods Categorized by Function

The DRS Remote Protocol contains 30 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 dsap 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**.
- Microsoft Windows NT® 4.0 operating system 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 (KCC)** support methods: IDL_DRSUpdateRefs, IDL_DRSReplicaAdd, IDL_DRSReplicaDel, IDL_DRSReplicaModify, IDL_DRSExecuteKCC. These methods are used by the KCC ([\[MS-ADTS\]](#) 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_DRSWriteSPN, IDL_DRSInitDemotion, IDL_DRSFinishDemotion, IDL_DRSReplicaDemotion. These methods are used by administrator tools to perform various specialized functions.

The specification of each method [RPC Methods and Their Behavior \(section 4\)](#), 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_DRSBind](#), 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_DRSUnbind](#), 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_DRSBind. 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_DRSBind, 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_DRSGetNCChanges](#): 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_DRSGetNT4ChangeLog](#): 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 Microsoft Windows NT® 4.0 operating system 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**.
 - Successfully processing an **IDL_DRSRemoveDsDomain** request first requires the removal of the metadata for all DCs hosting the domain NC for the domain that is to be removed. This precondition is achieved by the client calling **IDL_DRSRemoveDsServer** for each such DC.

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.

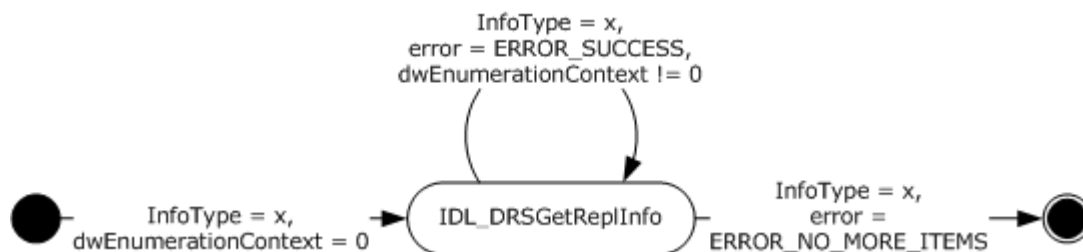


Figure 1: Using dwEnumerationContext

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 Remote 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 also supports replication based on the SMTP protocol; SMTP-based replication is specified in [\[MS-SRPL\]](#).

Some of the Active Directory state exposed by this protocol is also exposed by the Active Directory 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-RepAllInboundNeighbors](#), [msDS-RepConnectionFailures](#), [msDS-RepLinkFailures](#), [msDS-RepPendingOps](#), [msDS-RepAllOutboundNeighbors](#), [msDS-RepQueueStatistics](#) (these expose some functionality of IDL_DRSGetRepInfo), [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-NCRepInboundNeighbors](#), [msDS-NCRepCursors](#), [msDS-RepAttributeMetaData](#), [msDS-RepValueMetaData](#), [msDS-NCRepOutboundNeighbors](#) (these expose some functionality of IDL_DRSGetRepInfo), [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](#).

The Active Directory service must be fully initialized as described in [\[MS_ADSO\]](#) section 6.6.

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 uses Win32 error codes as defined in [\[MS-ERREF\]](#) section 2.2. Vendors SHOULD reuse those values with their indicated meaning. Choosing any other value risks a collision in the future.

This protocol uses [NTSTATUS](#) values as defined in [\[MS-ERREF\]](#) section 2.3. Vendors can choose their own values for this field, as long as the C bit (0x20000000) is set, indicating that it is a customer code.

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 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 the Local System account 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 Remote 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** (as specified in [\[MS-RPCE\]](#) section 2.2.1.1.8) of "packet privacy" (RPC_C_AUTHN_LEVEL_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** (see [\[MS-ADTS\]](#) section 3.1.1.1.10) 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 **FQDN** of the domain of the target DC.
- <DNS hostname> is the DNS host name of the target DC.

- <DNS forest name> is the FQDN 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/" as values of the [servicePrincipalName](#) attribute of the DC's **computer object**, but not the other form of SPN because that form of SPN is used for outbound replication. 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.
- <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 object (in the external AD DS domain) that corresponds to the **security principal** that the AD LDS service is running as. 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 Remote 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). Regardless of whether mutual authentication is required, a client MUST request integrity and encryption of the RPC messages by specifying an authentication level of "packet privacy" (RPC_C_AUTHN_LEVEL_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 Remote 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 (see [\[MS-ADTS\]](#) section 3.1.1.1.10) 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 an **Internet host name** for a DC.

- A client wants to connect to a DC in a particular domain. The client constructs any of the following three SPNs:
 - "ldap/<DNS hostname>/<NetBIOS domain name>"
 - "ldap/<DNS hostname>/<DNS domain name>"
 - "ldap/<NetBIOS hostname>/<NetBIOS domain name>" [<3>](#)

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 fully qualified domain name (FQDN) 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 domain name** of the target DC.
- <DNS domain name> is the FQDN 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>".
- <DNS forest name> is the FQDN of the forest of the target DC or the target GC server.

As an 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"
- "ldap/DC1/CONTOSO"

To allow mutual authentication to occur in client-to-DC protocol operations, an AD DS DC MUST store these seven 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.

- <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), then the AD LDS DC MUST store these two forms of SPN as values of the [servicePrincipalName](#) attribute of the object (in the external AD DS domain) that corresponds to the security principal that the AD LDS service is running as. 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.

2.3 Directory Service Schema Elements

This protocol is part of the Active Directory core family of protocols. To be fully compliant with Active Directory, an implementation of this protocol must be used in conjunction with the full Active Directory schema, containing all the schema attributes and classes specified in [\[MS-ADA1\]](#), [\[MS-ADA2\]](#), [\[MS-ADA3\]](#), [\[MS-ADLS\]](#), and [\[MS-ADSC\]](#).

3 Background to Behavior Specifications

3.1 Document Organization

In this document, information that is relevant to each particular RPC method is grouped with the specification of the behavior for that method. Information that is relevant to a particular RPC method includes: the **IDL** definition of that method, definitions for all types used exclusively by that method, and examples specific to that 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 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 Active Directory and by clients of the Active Directory 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 Active Directory 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. These identifiers, used in this document to improve the readability of the examples, are equivalent to the [ATTRTYP \(section 5.14\)](#) that actually identifies object classes and attributes. The mapping between **ATTRTYP** and the schema object representing a class or attribute is specified in [\[MS-ADTS\]](#) section 3.1.1.2.6.

- 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:

- $s := "\backslash x55\backslash x06\backslash x02"$

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:

- $d: \text{[DN](#)}; g: \text{[GUID](#)}; s: \text{[SID](#)}$
- $d := dsName.dn$
- $g := dsName.guid$
- $s := dsName.sid$

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:

- $\{one\ element, another\ element, \dots, yet\ another\ element\}$

Therefore:

- $S := \{\}$

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

If S is a set, the predicate $x \text{ in } S$ is true if x is a member of S . Therefore, the value of the expression:

- $13 \text{ in } \{1, 2, 3, 5, 7, 11\}$

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:

- $\text{stringSet} := \text{stringArrayPtr}^\wedge$

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
```

```
raise expr
```

The keyword **raise** is used to raise an RPC exception. The operand of the **raise** expression specifies the RPC exception to be raised. Details of how an RPC exception is raised are specified in [\[C706\]](#) section 12.6.4.7.

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 multivalued, 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*, noted earlier) 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 operating system 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), whose NetBIOS name is CONTOSO.
- ASIA.CONTOSO.COM (Domain NC), whose NetBIOS name is ASIA.

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(ld, "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](#).

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. [<4>](#)

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
<u>IDL_DRS CrackNames</u>	Looks up each of a set of objects in the directory and returns it to the caller in the requested format. Opnum: 12
<u>IDL_DRS WriteSPN</u>	Updates the set of service principal names (SPNs) on an object. Opnum: 13
<u>IDL_DRS RemoveDsServer</u>	Removes the representation of a DC from the directory. Opnum: 14
<u>IDL_DRS RemoveDsDomain</u>	Removes the representation of a domain from the directory. Opnum: 15
<u>IDL_DRS DomainControllerInfo</u>	Retrieves information about DCs in a given domain. Opnum: 16
<u>IDL_DRS AddEntry</u>	Adds one or more objects. Opnum: 17
<u>IDL_DRS ExecuteKCC</u>	Validates the replication interconnections of DCs and updates them if necessary. Opnum: 18
<u>IDL_DRS GetReplInfo</u>	Retrieves the replication state of the server. Opnum: 19
<u>IDL_DRS AddSidHistory</u>	Adds one or more SIDs to the <u>sIDHistory</u> attribute of a given object. Opnum: 20
<u>IDL_DRS GetMemberships2</u>	Retrieves group memberships for a sequence of objects. Opnum: 21
<u>IDL_DRS ReplicaVerifyObjects</u>	Verifies the existence of objects in an NC replica. Opnum: 22
<u>IDL_DRS GetObjectExistence</u>	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
<u>IDL_DRS QuerySitesByCost</u>	Determines the communication cost from a "from" site to one or more "to" sites. Opnum: 24
<u>IDL_DRS InitDemotion</u>	Performs the first phase of the removal of a DC from an AD LDS forest. Opnum: 25
<u>IDL_DRS ReplicaDemotion</u>	Replicates off all changes to the specified NC and moves any FSMOs held to another server. Opnum: 26
<u>IDL_DRS FinishDemotion</u>	Finishes or cancels the removal of a DC from an AD LDS forest.

Method	Description
	Opnum: 27

For information about the methods for which the Windows client operating systems can implement a client role, see the following behavior note. [<5>](#)

The methods will affect only the directory instance that is bound to the current context. If a server has several directory instances installed, the other instances will remain unchanged.

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.

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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE, ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.

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.

V3: Version 3 request.

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. [<6>](#)

```
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.

```
typedef struct {
    ENTINFLIST EntInfList;
} DRS_MSG_ADDENTRYREQ_V2;
```

EntInfList: The objects to be added, as specified in section [5.56](#).

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.

```
typedef struct {
    ENTINFLIST EntInflList;
    DRS_SecBufferDesc* pClientCreds;
} DRS_MSG_ADDENTRYREQ_V3;
```

EntInflList: 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.

V3: Version 3 response.

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. [<7>](#)

```
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 (section [4.1.1.1.26](#)).

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 (section [4.1.1.1.25](#)).

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 (section [4.1.1.1.26](#)).

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;
    ATRTYP 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 (section [4.1.1.1.26](#)).

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 (section [4.1.1.1.26](#)).

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: Unused. MUST be 0 and ignored.

nextRDN: Unused. MUST be 0 and ignored.

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;
    RPC_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: Unused. MUST be 0 and ignored.

RDNsInternal: Unused. MUST be 0 and ignored.

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 (section [4.1.1.1.26](#)).

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 (section [4.1.1.1.26](#)).

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 (section [4.1.1.1.26](#)).

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 (section [4.1.1.1.26](#)).

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.

Value and symbolic name	Description
0x00000001 attributeError	Attribute errors.
0x00000002 nameError	Name resolution error.
0x00000003 referralError	Referral.
0x00000004 securityError	Security error.
0x00000005 serviceError	Service error.
0x00000006 updError	Update error.
0x00000007 systemError	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.

Value and symbolic name	Description
0x0000000C ENOMEM	Out of memory.
0x00000010 EBUSY	Too busy to proceed.
0x00000016 EINVAL	Invalid parameter.
0x0000001C ENOSPC	Out of storage.
0x000007D1 NA_PROBLEM_NO_OBJECT	Parent object not found in the NC replica.
0x000003E9 PR_PROBLEM_NO_ATTRIBUTE_OR_VAL	Attribute or value not found.
0x000003EA PR_PROBLEM_INVALID_ATT_SYNTAX	Invalid attribute syntax.
0x000003EB PR_PROBLEM_UNDEFINED_ATT_TYPE	Unknown attribute type.
0x000003EC PR_PROBLEM_WRONG_MATCH_OPER	Incorrect matching operation (only applies to match operators in LDAP filters).
0x000003ED	Attribute value violated a schema constraint.

Value and symbolic name	Description
PR_PROBLEM_CONSTRAINT_ATT_TYPE	
0x000003EE PR_PROBLEM_ATT_OR_VALUE_EXISTS	Attribute or value already exists (multiple values specified for a single-valued attribute OR duplicate value specified for a multi-valued attribute).
0x00000FA1 SE_PROBLEM_INAPPROPRIATE_AUTH	Inappropriate authentication method.
0x00000FA2 SE_PROBLEM_INVALID_CREDENTIALS	Invalid user name or password.
0x00000FA3 SE_PROBLEM_INSUFF_ACCESS_RIGHTS	Access denied.
0x00000FA4 SE_PROBLEM_INVALID_SIGNATURE	Invalid signature.
0x00000FA5 SE_PROBLEM_PROTECTION_REQUIRED	Encrypted connection required.
0x00000FA6 SE_PROBLEM_NO_INFORMATION	Insufficient permissions to generate a referral.
0x00001389 SV_PROBLEM_BUSY	Directory service is busy.
0x0000138A SV_PROBLEM_UNAVAILABLE	Directory service is unavailable.
0x0000138B SV_PROBLEM_WILL_NOT_PERFORM	The requested operation will not be performed.
0x0000138C SV_PROBLEM_CHAINING_REQUIRED	Chaining is required to perform the operation.
0x0000138D SV_PROBLEM_UNABLE_TO_PROCEED	Directory service is unable to proceed with the requested operation.
0x0000138E SV_PROBLEM_INVALID_REFERENCE	Invalid crossRef object.
0x0000138F SV_PROBLEM_TIME_EXCEEDED	Time limit exceeded while processing the operation.
0x00001390 SV_PROBLEM_ADMIN_LIMIT_EXCEEDED	Administrative limit exceeded while processing the operation.
0x00001391 SV_PROBLEM_LOOP_DETECTED	Not in use.
0x00001392 SV_PROBLEM_UNAVAIL_EXTENSION	The requested extended operation is not available.
0x00001393 SV_PROBLEM_OUT_OF_SCOPE	Not in use.

Value and symbolic name	Description
0x00001394 SV_PROBLEM_DIR_ERROR	Generic directory service error.
0x00001771 UP_PROBLEM_NAME_VIOLATION	Naming violation.
0x00001772 UP_PROBLEM_OBJ_CLASS_VIOLATION	Object class violation.
0x00001773 UP_PROBLEM_CANT_ON_NON_LEAF	The operation cannot be performed on an object with child objects.
0x00001774 UP_PROBLEM_CANT_ON_RDN	The operation cannot be performed on an RDN attribute.
0x00001775 UP_PROBLEM_ENTRY_EXISTS	Object already exists.
0x00001776 UP_PROBLEM_AFFECTS_MULT_DSAS	The operation affects multiple DCs.
0x00001777 UP_PROBLEM_CANT_MOD_OBJ_CLASS	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: unicodestring,
    guid: GUID): unicodestring
```

This procedure returns a replication SPN with the given DNS host name (in `dnsHostName`) and GUID (in `guid`). 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, TRUE)

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
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

/*The cross ref was created/enabled. Ensure that the respective
sub-ref object is created */
AddSubRef(cr!ncName)

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
    /* Creating the domain crossRef in the same call is
     * allowed. 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

```

Informative summary of behavior: This procedure examines the values for the [hasMasterNCs](#) attribute found in *e* and returns the domain NC. The [hasMasterNCs](#) attribute always contains the dsnames of the **schema NC**, the config NC, and the default domain NC of the DC represented by *e*. The domain NC is identified by a process of elimination.

```

prefixTable: PrefixTable
attr:ATTR
j:DWORD

prefixTable := NewPrefixTable()

/* Scan the ENTINF e to get the attribute for which ATTRTYP is
 * hasMasterNCs.*/
attr := ENTINF_GetAttribute(e, hasMasterNCs, prefixTable)
for j=0 to (attr.AttrVal.valCount-1)
    if (attr.AttrVal.pAVal[j].pVal ≠ SchemaNC()) and
        (attr.AttrVal.pAVal[j].pVal ≠ ConfigNC()) and
            (attr.AttrVal.pAVal[j].valLen > 0) then
                return attr.AttrVal.pAVal[j].pVal^
    end if
endfor

return null

```

4.1.1.2.7 ENTINF_GetAttribute

```

procedure ENTINF_GetAttribute (
    entInf: ENTINF,
    attribute: ATTRTYP,
    prefixTable: PrefixTable
): ATTR

```

Informative summary of behavior: The ENTINF_GetAttribute procedure scans an [ENTINF](#) structure and returns the first [ATTR](#) 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) then
        return entInf.AttrBlock.pAttr[i]
    endif
endfor
return null

```

4.1.1.2.8 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](#).

4.1.1.2.9 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.

4.1.1.2.10 PerformModifyEntInf

```

procedure PerformModifyEntInf(
  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, PerformModifyEntInf returns true. If the operation fails for some reason, PerformModifyEntInf 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 operating system and Windows Server® 2008 R2 operating system.

```
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

ValidateDRSInput(hDrs, 17)

/* Only attributes and classes in the base schema may be specified.*/
```



```

prefixTable := NewPrefixTable()

/* Set the default response version */
pdwOutVersion := 2

if dwInVersion = 1 then /* obsolete */
    pmsgOut^.V1.Guid := 0
    pmsgOut^.V1.Sid := 0
    pmsgOut^.V1.errCode := 0
    pmsgOut^.V1.dsid := 0
    pmsgOut^.V1.extendedErr := 0
    pmsgOut^.V1.extendedData := 0
    pmsgOut^.V1.problem := 0
else if dwInVersion = 2 then
    pmsgOut^.V2.pErrorObject := null
    pmsgOut^.V2.errCode := 0
    pmsgOut^.V2.dsid := 0
    pmsgOut^.V2.extendedEr := 0
    pmsgOut^.V2.extendedData := 0
    pmsgOut^.V2.problem := 0
    pmsgOut^.V2.cObjectsAdded := 0
    pmsgOut^.V2.infoList := null
else if dwInVersion = 3 then
    pmsgOut^.V3.pdsErrObject := null
    pmsgOut^.V3.dwErrVer := 0
    pmsgOut^.V3.pErrData := null
    pmsgOut^.V3.ULONG cObjectsAdded := 0
    pmsgOut^.V3.infoList := null
endif

/* Validate parameters. */
if not (dwInVersion in {2,3}) then
    SetErrorData(SV_PROBLEM_UNAVAILABLE, 0, ERROR_DS_UNAVAILABLE,
        pmsgOut, 2)
    return 0
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_WIN2008 then
            res := PerformModifyEntInf(
                hDrs, e^.Entinf, ADR(infoList[cObjects]))
            if not res then
                return 0
            endif
        else
            /* Not supported (Win2k3 or older DC). */
            SetErrorData(SV_PROBLEM_UNAVAILABLE,
                0,
                ERROR_DS_UNAVAILABLE,
                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
        else
            /* Not supported. */
            SetErrorData(SV_PROBLEM_BUSY, 0, ERROR_DS_DRA_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 must be 1, because no other version is supported.

pmsgOut: Pointer to the response message.

Return Values: 0 or one of the following Windows error codes:
 ERROR_DS_MUST_RUN_ON_DST_DC or ERROR_INVALID_PARAMETER.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE, ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.

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 an FQDN or a NetBIOS name.

SrcPrincipal: Name of a security principal (user, computer, or group) in the source domain. This is the source security principal, whose SIDs will be added to the destination security principal. If **Flags** contains DS_ADDSID_FLAG_PRIVATE_CHK_SECURE, this parameter is not used and is not validated. Otherwise, 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 an Internet host name or a NetBIOS name. This parameter is only used when **Flags** contains neither DS_ADDSID_FLAG_PRIVATE_CHK_SECURE nor DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ. If **Flags** contains DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ, this parameter is not used, but it is validated.

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. The domain name can be an FQDN or a NetBIOS domain name.

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 an FQDN 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. Otherwise, if **Flags** contains DS_ADDSID_FLAG_PRIVATE_CHK_SECURE, this parameter is not used and is not validated. 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: Zero if successful, otherwise a 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. The flags are represented in little-endian byte order.

										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	
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	C	X	X	X	X	X	X	
																							E	S								
																							L									

X: Unused. MUST be zero and ignored.

CS (DS_ADDSID_FLAG_PRIVATE_CHK_SECURE, 0x40000000): If set, the server verifies whether the channel is secure and returns the result of the verification in the response.

DEL (DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ, 0x80000000): If set, the server appends the [objectSid](#) and [sIDHistory](#) attributes of SrcPrincipal to the [sIDHistory](#) attribute of DstPrincipal, and deletes 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: unicaststring): ConnectionCtx
```

Creates a [ConnectionCtx](#) for the DC named by dcname, associating the credentials of the client's security context, which MUST be retrieved using the method described in [\[MS-RPCE\]](#) section 3.3.3.4.3, with the ConnectionCtx. dcname may be the Internet host 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: unicaststring,  
    username: unicaststring,  
    pwd: unicaststring,  
    domain: unicaststring): 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 Internet host 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. [<8>](#)

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 Internet host name of the DC that holds the PDC FSMO role for the domain whose name is domainName (see [\[MS-ADTS\]](#) section 7.1.5.4), or null if such a DC cannot be found. domainName can be either the FQDN 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 IsAuditingEnabled

```
procedure IsAuditingEnabled(): boolean
```

Returns true if auditing on the server on which it is called is enabled, 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 Microsoft Windows NT® 4.0 operating system and is not running at least Microsoft Windows NT® 4.0 operating system Service Pack 4 (SP4), this procedure returns false. Otherwise, it returns true. [<9>](#)

4.1.2.2.13 IsAuditingGroupPresent

```
procedure IsAuditingGroupPresent(dcname: unicodestring, nETBIOSName: unicodestring): DWORD
```

Returns `ERROR_NO_SUCH_ALIAS` if the DC represented by the dcname does not have a domain local group whose sAMAccountName is the value of the *nETBIOSName* parameter appended with three dollar signs \$\$\$\$. Otherwise, it returns `ERROR_SUCCESS`. This group is not present by default and must be created by the administrator of the directory service.

4.1.2.2.14 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** (as specified in [\[MS-WSO\]](#) section 3.1.2.1.5) whose value is less than 1000, and returns false otherwise.

4.1.2.2.15 LastRID

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

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

4.1.2.2.16 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. See [\[MS-ADTS\]](#) section 3.1.1.5.5 for more information about the delete operation. 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.

The preceding are the only variants supported by the **IDL_DRSAddSidHistory** method. In particular, the case of source and destination principals in different domains within the same forest is not supported.

```
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
```

```

rt: ULONG

ValidateDRSInput(hDrs, 20)

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

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

/* Currently, only version 1 is supported. The RPC IDL definitions
* for the interface do not allow passing in a version other than 1. */
if dwInVersion # 1 then
    return ERROR_INVALID_PARAMETER
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_DS_INTERNAL_FAILURE
        return ERROR_INVALID_PARAMETER
    endif

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

    /* Source and destination principals must be in same domain. */
    if srcNc = null or dstNc = null or srcNc # dstNc then
        pmsgOut^.V1.dwWin32Error := ERROR_INVALID_PARAMETER
        return 0
    endif
endif

```

```

/* Destination NC must be this server's default domain NC. */
if dstNc ≠ DefaultNC() then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_MASTERDSA_REQUIRED
    return 0
endif

/* Verify that this server has auditing enabled */
if not IsAuditingEnabled() then
    pmsgOut^.V1.dwWin32Error :=
        ERROR_DS_DESTINATION_AUDITING_NOT_ENABLED
    return 0
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 0
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 0
else
    if crDst!nTMixedDomain = 1 then
        pmsgOut^.V1.dwWin32Error := ERROR_DS_DST_DOMAIN_NOT_NATIVE
        return 0
    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 0
endif

/* Check that this machine has rights to delete the source principal. */
if (not AccessCheckObject(srcPrinc, RIGHT_DELETE)) and
    (not AccessCheckObject(srcPrinc.parent, RIGHT_DS_DELETE_CHILD))
    then
        pmsgOut^.V1.dwWin32Error := ERROR_ACCESS_DENIED
        return 0
    endif
endif

```

```

/* Save the source principal's SID and SID history and then delete the principal */
srcPrincSid := srcPrinc!objectSid
srcPrincSidHistory := srcPrinc!sidHistory
rt = RemoveObj(srcPrinc,false)
if(rt ≠ 0) then
    pmsgOut^.V1.dwWin32Error := rt
    return 0
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
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_DS_INTERNAL_FAILURE
    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!dnsRoot = pmsgIn^.V1.DstDomain or
            dd!nETBIOSName = pmsgIn^.V1.DstDomain))
if crDst = null then
    pmsgOut^.V1.dwWin32Error :=
        ERROR_DS_DESTINATION_DOMAIN_NOT_IN_FOREST
    return 0
endif

/* Confirm source domain is not in forest of server. */
crSrc := select one ss from subtree ConfigNC() where
    (crossRef in ss!objectClass and
        (ss!dnsRoot = pmsgIn^.V1.SrcDomain or
            ss!nETBIOSName = pmsgIn^.V1.SrcDomain))

```

```

        and FLAG_CR_NTDS_NC in ss!systemFlags
        and FLAG_CR_NTDS_DOMAIN in ss!systemFlags)
if crSrc ≠ null then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_SOURCE_DOMAIN_IN_FOREST
    return 0
endif

/* Destination NC must be this server's default domain NC. */
if crDst!nCName ≠ DefaultNC() then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_MASTERDSA_REQUIRED
    return 0
endif

/* Destination domain must be in native mode. */
if crDst!nTMixedDomain = 1 then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_DST_DOMAIN_NOT_NATIVE
    return 0
endif

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

/* Verify this server has auditing enabled for destination domain. */
if not IsAuditingEnabled() then
    pmsgOut^.V1.dwWin32Error :=
        ERROR_DS_DESTINATION_AUDITING_NOT_ENABLED
    return 0
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 0
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 0
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 0
    endif
endif
if srcDomainController = null then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_CANT_FIND_DC_FOR_SRC_DOMAIN
    return 0
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 0
endif

/* Confirm client has administrative rights on source DC. */
if not HasAdminRights(srcCtx) then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_INSUFF_ACCESS_RIGHTS
    return 0
endif

/* Retrieve source principal from source DC using the remote connection.
 * In this case, pmsgIn^.V1.SrcPrincipal is a SAM name.
 * Example: If pmsgIn^.V1.SrcPrincipal value is username1 then
 * following query is executed in the source DC:
 *   select one o from subtree dc.defaultNC where (o!sAMAccountName = "username1")
 */
srcPrinc := RemoteQuery(srcCtx, "select one o
                                "select one o from subtree dc.defaultNC where (o!sAMAccountName = "
                                + '""' + pmsgIn^.V1.SrcPrincipal + '""' + ")"
                                )
if srcPrinc = null then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_OBJ_NOT_FOUND
    return 0
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 0
endif

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

/* Verify that no principal other than the destination
 * principal exists in the destination domain that contains
 * a SID that matches the source principal. */

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_EXISTS_IN_FOREST
    return 0
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 0
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 0
endif

/* Verify that if source domain has a domain local group srcDomainNetBIOSName$$$
 */
if IsAuditingGroupPresent(srcDomainController, pmsgIn^.V1.SrcDomain) = ERROR_NO_SUCH_ALIAS
    pmsgOut^.V1.dwWin32Error := ERROR_NO_SUCH_ALIAS
    return 0
endif

/* 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, the case is disallowed 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 0
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 0
endif

if group in srcPrinc!objectClass and
    srcPrinc!groupType # dstPrinc!groupType then
    pmsgOut^.V1.dwWin32Error :=
        ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
    return 0
endif

/* Check if source principal is built-in principal. */
if IsBuiltinPrincipal(srcPrinc!objectSid) then
    pmsgOut^.V1.dwWin32Error := ERROR_DS_UNWILLING_TO_PERFORM
    return 0
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 0
    endif
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

```

4.1.2.4 Examples of the IDL_DRSAddSidHistory Method

4.1.2.4.1 Calling IDL_DRSAddSidHistory with DS_ADDSID_FLAG_PRIVATE_CHK_SECURE Flags

This flag is used when the caller wants to check whether an RPC call to DC1 is secure.

4.1.2.4.1.1 Client Request

A client invokes the IDL_DRSAddSidHistory method against DC1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 1
- pmsgIn = DRS_MSG_ADDSIDREQ_V1
 - Flags = 0x40000000

4.1.2.4.1.2 Server Response

The server returns a code of 0 and the following values:

- pdwOutVersion = 1
- pmsgOut = DRS_MSG_ADDSIDREPLY_V1
 - dwWin32Error: 0

4.1.2.4.1.3 Final State

There are no changes in state.

4.1.2.4.2 Calling IDL_DRSAddSidHistory with DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ Flags

In this example, the user "Kim Akers" has an account in domain DC=contoso, DC=com with a Microsoft Windows NT® 4.0 operating system account name "CONTOSO\kimakers". There is another account in the same domain for the user "Kim Akers" with the Windows NT 4.0 account name "CONTOSO\kimakers1". The domain administrator wants to add a SID of "CONTOSO\Kimakers1" account to the SIDHistory of "CONTOSO\kimakers" and delete "CONTOSO\Kimakers1".

4.1.2.4.2.1 Initial State

Querying the **user object** whose sAMAccountName is kimakers in the domain NC DC=CONTOSO, DC=COM on DC1:

- ldap_search_s("DC=contoso,DC=com", wholeSubtree, "(sAMAccountName=kimakers)", [objectClass, distinguishedName, sAMAccountName, objectSid, sIDHistory])
- Result <0>: (null)
- Matched DNs:
- Getting 1 entry:
- >> 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;
 - 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1144

Querying the user object whose sAMAccountName is kimakers1 in the domain NC DC=CONTOSO, DC=COM on DC1:

- ldap_search_s("DC=contoso,DC=com", wholeSubtree, "(sAMAccountName=kimakers1)", [objectClass, distinguishedName, sAMAccountName, objectSid, sIDHistory])
- Result <0>: (null)
- Matched DNs:

- Getting 1 entries:
- >> Dn: CN=Kim Akers1,CN=Users,DC=contoso,DC=com
 - 4> objectClass: top; person; organizationalPerson; user;
 - 1> distinguishedName: CN=Kim Akers1,CN=Users,DC=contoso,DC=com;
 - 1> sAMAccountName: KimAkers1;
 - 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1129;

4.1.2.4.2.2 Client Request

A client invokes the **IDL_DRSAddSidHistory** method against DC1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 1
- pmsgIn = DRS_MSG_ADDSIDREQ_V1
 - Flags = 0x80000000
 - SrcPrincipal = "CN=Kim Akers1,CN=Users,DC=contoso,DC=com"
 - DstPrincipal = "CN=Kim Akers,CN=Users,DC=contoso,DC=com"

4.1.2.4.2.3 Server Response

The server returns a code of 0 and the following values:

- pdwOutVersion = 1
- pmsgOut = DRS_MSG_ADDSIDREPLY_V1
 - dwWin32Error: 0

4.1.2.4.2.4 Final State

The **sIDHistory** attribute on the user object whose DN is "Kim Akers,CN=Users,DC=contoso,DC=com" contains one value:

- ldap_search_s("DC=contoso,DC=com", *wholeSubtree*, "(sAMAccountName=kimakers)", [*objectClass, distinguishedName, sAMAccountName, objectSid, sIDHistory*])
- Result <0>: (null)
- Matched DNs:
- Getting 1 entry:
- >> 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;

- 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1144;
- 1> sIDHistory: S-1-5-21-254470460-2440132622-709970653-1129;

The user object whose DN is "Kim Akers1,CN=Users,DC=contoso,DC=com" is deleted:

```
ldap_search_s("DC=contoso,DC=com", wholeSubtree, "(sAMAccountName=kimakers1)",
[objectClass, distinguishedName, sAMAccountName, objectSid, sIDHistory])
```

- Result <0>: (null)
- Matched DNs:
- Getting 0 entries:

4.1.2.4.3 Calling IDL_DRSAddSidHistory with 0 in Flags

The user "Kim Akers" has an account in domain DC=contoso, DC=com with a Microsoft Windows NT® 4.0 operating system account name "CONTOSO\kimakers". The user has another account in a separate forest in domain DC=legacycontoso,DC=com with a Windows NT 4.0 account name "LEGACYCONTOSO\kimakers1". The domain administrator wants to add the SID of "LEGACYCONTOSO\kimakers1" account to the sIDHistory of "CONTOSO\kimakers". The administrator's account name in the LEGACYCONTOSO domain is LegacyContosoAdmin with password Passw0rd123. LEGACYCONTOSO is the NetBIOS name for the LEGACYCONTOSO.com domain.

4.1.2.4.3.1 Initial State

Querying the user object whose sAMAccountName is kimakers in the domain NC DC=CONTOSO, DC=COM on DC1:

- ldap_search_s("DC=contoso,DC=com", wholeSubtree, "(sAMAccountName=kimakers)", [objectClass, distinguishedName, sAMAccountName, objectSid, sIDHistory])
- Result <0>: (null)
- Matched DNs:
- Getting 1 entry:
- 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
 - 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1144

Querying the user object whose sAMAccountName is kimakers1 in the domain NC DC=LEGACYCONTOSO, DC=COM on DC9:

- ldap_search_s("DC=legacycontoso,DC=com", wholeSubtree, "(sAMAccountName=kimakers1)", [objectClass, distinguishedName, sAMAccountName, objectSid, sIDHistory])
- Result <0>: (null)

- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=Kim Akers1,CN=Users,DC=legacycontoso,DC=com
 - 4> objectClass: top; person; organizationalPerson; user;
 - 1> distinguishedName: CN=Kim Akers1,CN=Users,DC=legacycontoso,DC=com;
 - 1> sAMAccountName: KimAkers1;
 - 1> objectSid: S-1-5-21-1137440724-3092688314-3181763971-1153;

4.1.2.4.3.2 Client Request

A client invokes the IDL_DRSAddSidHistory method against DC1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 1
- pmsgIn = DRS_MSG_ADDSIDREQ_V1
 - Flags = 0
 - SrcDomain = "legacycontoso.com"
 - SrcPrincipal = "KimAkers1 "
 - SrcCredsDomain = "legacycontoso "
 - SrcCredsDomainLength = 13
 - SrcCredsUser=" LegacyContosoAdmin"
 - SrcCredsUserLength = 18
 - SrcCredsPassword= "Passw0rd123"
 - SrcCredsPasswordLength =11
 - DstDomain = "contoso"
 - DstPrincipal = "KimAkers"

4.1.2.4.3.3 Server Response

The server returns a code of 0 and the following values:

- pdwOutVersion = 1
- pmsgOut = DRS_MSG_ADDSIDREPLY_V1
 - dwWin32Error: 0

4.1.2.4.3.4 Final State

The `sIDHistory` attribute on the user object whose DN is "Kim Akers,CN=Users,DC=contoso,DC=com" contains one value:

- `ldap_search_s("DC=contoso,DC=com", wholeSubtree, "(sAMAccountName=kimakers)", [objectClass, distinguishedName, sAMAccountName, objectSid, sIDHistory])`
- Result <0>: (null)
- Matched DNs:
- Getting 1 entry:
- >> 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;
 - 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1144;
 - 1> sIDHistory: S-1-5-21-1137440724-3092688314-3181763971-1153;

In the domain NC DC=LEGACYCONTOSO, DC=COM, the user object whose `sAMAccountName` is `kimakers1` is unchanged:

- `ldap_search_s("DC=legacycontoso,DC=com", wholeSubtree, "(sAMAccountName=kimakers1)", [objectClass, distinguishedName, sAMAccountName, objectSid, sIDHistory])`
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=Kim Akers1,CN=Users,DC=legacycontoso,DC=com
 - 4> objectClass: top; person; organizationalPerson; user;
 - 1> distinguishedName: CN=Kim Akers1,CN=Users,DC=legacycontoso,DC=com;
 - 1> sAMAccountName: KimAkers1;
 - 1> objectSid: S-1-5-21-1137440724-3092688314-3181763971-1153;

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** pextServer,
```

```
[out, ref] DRS_HANDLE* phDrs
);
```

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

puuidClientDsa: A pointer to a GUID that identifies 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.

Exceptions Thrown: This method does not throw exceptions beyond those thrown by the underlying RPC protocol.

4.1.3.1 Client Behavior When Sending the IDL_DRSBind Request

The client uses `puuidClientDsa` to pass an identifier. If the client uses the returned `DRS_HANDLE` for subsequent calls to the [IDL_DRSWriteSPN](#) method, then the client MUST pass `NTDSAPI_CLIENT_GUID` in `puuidClientDsa`. For any other uses, the server places no constraints on the value of `puuidClientDsa`.[<10>](#)

The client uses `pextClient` to pass a properly initialized [DRS_EXTENSIONS_INT](#) structure to the server. If the client is a DC, it reads the value of [msDS-ReplicationEpoch](#) from its [nTDSDSA](#) object and assigns this value to the `dwReplEpoch` field of the [DRS_EXTENSIONS_INT](#) structure; otherwise, it sets the `dwReplEpoch` field of the [DRS_EXTENSIONS_INT](#) structure to zero. If the client is a DC, it reads the value of `objectGUID` from the Config NC object and assigns this value to the `ConfigObjGUID` field of the [DRS_EXTENSIONS_INT](#) structure; otherwise, it sets the `ConfigObjGUID` field of the [DRS_EXTENSIONS_INT](#) structure to the `NULL GUID` value.

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.[<11>](#)

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 [5.40](#).
- **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.

[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
8	<code>DRS_EXT_GETCHGREPLY_V6</code>	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_DRSSetSPN

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

[IDL_DRSRemoveDsServer](#)

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

[IDL_DRSRemoveDsDomain](#)

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

IDL_DRSDomainControllerInfo

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

IDL_DRSSAddEntry

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_DRSExecuteKCC](#)

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

IDL_DRSGetReplInfo

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 UUID passed as `puuidClientDsa^` and the [DRS_EXTENSIONS_INT](#) structure passed as `pextClient^`.

The server sets `ppextServer` to a [DRS_EXTENSIONS_INT](#) structure whose `dwReplEpoch` and `ConfigObjGUID` fields are 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. [<12>](#) The server then returns a [DRS_HANDLE](#) as the referent of `phDrs`.

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
10	DRS_EXT_GETCHGREQ_V10 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_DRSTransferDomainMove](#)

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

Request version	DRS_EXTENSIONS_INT bit(s)
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
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_QUERY_SITES_BY_COST_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. In the case when the client is an AD LDS server, the client additionally verifies that the value of the **ConfigObjGUID** field in the [DRS_EXTENSIONS_INT](#) structure returned by the server matches the GUID of its own config NC object. If the values do not match, the AD LDS client considers the handle to be invalid and does not use the handle in method calls on the `drsuapi` interface. Once a valid handle has been acquired by the client, 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 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;
- 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_CRYPTOBIND
 - 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
- DRS_EXT_GETCHGREQ_V10
- 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
- DRS_EXT_GETCHGREQ_V10
- 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 Windows error code.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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. This field SHOULD be ignored by the server.

LocaleId: The locale used by the client. This field SHOULD be ignored by the server.

dwFlags: Zero or more of the following bit flags, which are presented in little-endian byte order.

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

X: Unused. MUST be zero and ignored.

GC (DS_NAME_FLAG_GCVERIFY, 0x00000004): If set, the call fails if the server is not a GC server.

TR (DS_NAME_FLAG_TRUST_REFERRAL, 0x00000008): 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, 0x80000000): 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 0xFFFFFFFF9	Get value of sAMAccountName attribute.
DS_MAP_SCHEMA_GUID	Get LDAP display name from schema GUID. The given schema GUID should be in the curly braced

Value	Meaning
0xFFFFFFFF8	GUID string format as specified in [MS-DTYP] section 2.3.2.3.
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	Compares input names against the values of the altSecurityIdentities attribute.
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 in the forest.
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 0xFFFFFEEF	Compares input names against the user principal name and the values of the altSecurityIdentities attribute.

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: Curly braced string representation of an [objectGUID](#). The format of the string representation is specified in [\[MS-DTYP\]](#) section 2.3.2.3.

DS_CANONICAL_NAME: A **canonical name**.

DS_USER_PRINCIPAL_NAME: User principal name.

DS_CANONICAL_NAME_EX: Same as DS_CANONICAL_NAME except that the 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	Description
0 DS_NAME_NO_ERROR	No error occurred during the name translation.
1	Generic processing error during the name

Symbolic name	Description
DS_NAME_ERROR_RESOLVING	translation.
2 DS_NAME_ERROR_NOT_FOUND	The object with the specified name cannot be found.
3 DS_NAME_ERROR_NOT_UNIQUE	More than one object is located with the specified name.
4 DS_NAME_ERROR_NO_MAPPING	The desired output format cannot be applied to the object with the specified name.
5 DS_NAME_ERROR_DOMAIN_ONLY	Only the domain part of the name was translated.
7 DS_NAME_ERROR_TRUST_REFERRAL	The specified name belongs to a trusted forest, a referral is returned.
0xFFFFFFFF2 DS_NAME_ERROR_IS_SID_HISTORY_UNKNOWN	The specified name matches a value in the sidHistory attribute of an object, but the type of the object is unknown.
0xFFFFFFFF3 DS_NAME_ERROR_IS_SID_HISTORY_ALIAS	Translation was successful. The specified name matches a value in the sidHistory attribute of an object. The object's sAMAccountType attribute value is either SAM_NON_SECURITY_ALIAS_OBJECT or SAM_ALIAS_OBJECT as defined in [MS-SAMR] section 2.2.1.9, ACCOUNT_TYPE Values.
0xFFFFFFFF4 DS_NAME_ERROR_IS_SID_HISTORY_GROUP	Translation was successful. The specified name matches a value in the sidHistory attribute of an object. The object's sAMAccountType attribute value is either SAM_GROUP_OBJECT or SAM_NON_SECURITY_GROUP_OBJECT as defined in [MS-SAMR] section 2.2.1.9, ACCOUNT_TYPE Values.
0xFFFFFFFF5 DS_NAME_ERROR_IS_SID_HISTORY_USER	Translation was successful. The specified name matches a value in the sidHistory attribute of an object. The object's sAMAccountType attribute value is SAM_USER_OBJECT or SAM_MACHINE_ACCOUNT or SAM_TRUST_ACCOUNT as defined in [MS-SAMR] section 2.2.1.9, ACCOUNT_TYPE Values.
0xFFFFFFFF6 DS_NAME_ERROR_IS_SID_UNKNOWN	The specified name matches the objectSid attribute of an object, but the type of the object is unknown.
0xFFFFFFFF7 DS_NAME_ERROR_IS_SID_ALIAS	Translation was successful. The specified name matches the objectSid attribute of an object. The object's sAMAccountType attribute value is either SAM_NON_SECURITY_ALIAS_OBJECT or SAM_ALIAS_OBJECT as defined in [MS-SAMR] section 2.2.1.9, ACCOUNT_TYPE Values.
0xFFFFFFFF8	Translation was successful. The specified name

Symbolic name	Description
DS_NAME_ERROR_IS_SID_GROUP	matches the objectSid attribute of an object. The object's sAMAccountType attribute value is either SAM_GROUP_OBJECT or SAM_NON_SECURITY_GROUP_OBJECT as defined in [MS-SAMR] section 2.2.1.9, ACCOUNT_TYPE Values.
0xFFFFFFFF9 DS_NAME_ERROR_IS_SID_USER	Translation was successful. The specified name matches the objectSid attribute of an object. The object's sAMAccountType attribute value is SAM_USER_OBJECT or SAM_MACHINE_ACCOUNT or SAM_TRUST_ACCOUNT as defined in [MS-SAMR] section 2.2.1.9, ACCOUNT_TYPE Values.
0xFFFFFFFFFA DS_NAME_ERROR_SCHEMA_GUID_CONTROL_RIGHT	Translation was successful. The GUID identifies a control access right.
0xFFFFFFFFFB DS_NAME_ERROR_SCHEMA_GUID_CLASS	Translation was successful. The GUID identifies a classSchema object.
0xFFFFFFFFFC DS_NAME_ERROR_SCHEMA_GUID_ATTR_SET	Translation was successful. The GUID identifies a property set.
0xFFFFFFFFFD DS_NAME_ERROR_SCHEMA_GUID_ATTR	Translation was successful. The GUID identifies an attributeSchema object.
0xFFFFFFFFFE DS_NAME_ERROR_SCHEMA_GUID_NOT_FOUND	The GUID cannot be resolved.
0xFFFFFFFFFF DS_NAME_ERROR_IS_FPO	The object with the specified name is a Foreign Principal Object.

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 FQDN. 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 FQDN 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 subauthority 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 SIDs.

4.1.4.2.9 CrackNames

```
procedure CrackNames(DRS_MSG_CRACKREQ_V1 msgIn, DS_NAME_RESULTW *pmsgOut): ULONG
```

The CrackNames method implements the core functionality of [IDL_DRSCrackNames](#), that is, looking up directory object names that are provided in one format (for example, SPNs) and returning them in a different format (for example, SPN).

```
i: DWORD
rt: set of DSName
serverObj, siteObj, attr, class, er: DSName
guid: GUID

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.formatDesired,
            msgIn.rpNames[i])
    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.rpNames[0]. */
    rt := select all o from subtree msgIn.rpNames[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.rpNames[0]. */
    /* First find all DCs in a site named msgIn.rpNames[0]. */
    rt := select all o from subtree msgIn.rpNames[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.rpNames[0] in a site named
     * msgIn.rpNames[1]. */
    rt := select all o from subtree msgIn.rpNames[1]
        where o!objectCategory = GetDefaultObjectCategory(nTDSDSA)
        and msgIn.rpNames[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.rpNames[0]. */
    rt := select all o from subtree msgIn.rpNames[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.rpNames[0]. */
    serverObj := GetDSNameFromDN(msgIn.rpNames[0])
    dsaObj := select one o from subtree msgIn.rpNames[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!dnsHostName
        msgOut^.rItems[1].status := DS_NAME_NO_ERROR
        msgOut^.rItems[2].pName := serverObj!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.rpNames[i] to attribute or class
     * or propertySet. */
    /* Assume no match by default. */
    msgOut^.rItems[i].status := DS_NAME_ERROR_SCHEMA_GUID_NOT_FOUND

    /* Validate the string guid contained in msgIn.rpNames[i] */
    guid := GuidFromString(true, msgIn.rpNames[i])
    if guid ≠ null then

      /* First, try to find a matching attribute. */
      attr := select one o from subtree SchemaNC()
        where attributeSchema in o!objectClass and
          o!schemaIdGuid = msgIn.rpNames[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.rpNames[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.rpNames[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
  endfor
  msgOut^.cItems := msgIn.cNames
endif

return ERROR_SUCCESS

```

4.1.4.2.10 LookupName

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

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 given output format.

```
    rt: sequence of DSName  
    obj: DSName  
    fSidHistory: boolean  
    result: DS_NAME_RESULT_ITEMW  
    names: sequence of unicodestring  
    domainName: unicodestring  
    fCanonicalEx: boolean  
    referredDomain: unicodestring  
  
    if formatOffered = DS_UNKNOWN_NAME then  
        return LookupUnknownName(flags, name, formatDesired)  
    endif  
  
    domainName := null  
  
    if formatOffered = DS_FQDN_1779_NAME then  
        rt := LookupAttr(flags, distinguishedName, name)  
        domainName := DomainDNSNameFromDomain(RetrieveDCSuffixFromDn(name))  
    else if formatOffered = DS_NT4_ACCOUNT_NAME then  
        rt := LookupAttr(flags, sAMAccountName,  
            UserNameFromNT4AccountName(name))  
        domainName := DomainNameFromNT4AccountName(name)  
    else if formatOffered = DS_USER_PRINCIPAL_NAME then  
        rt := LookupUPNAndAltSecID(flags, false, name)  
        domainName := DomainNameFromUPN(name)  
    else if formatOffered = DS_CANONICAL_NAME then  
        rt := LookupCanonicalName(name)  
        domainName := DomainNameFromCanonicalName(name)  
    else if formatOffered = DS_UNIQUE_ID_NAME then  
        rt := select all o from all where o!objectGuid = GuidFromString(true, 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)  
        domainName := GetServiceNameFromSPN(name)  
    else if formatOffered in {DS_SID_OR_SID_HISTORY_NAME,  
        DS_STRING_SID_NAME} then  
        rt := LookupSID(flags, SidFromStringSid(name))  
        domainName := DomainNameFromSid(DomainSidFromSid(SidFromStringSid(name)))  
    else if formatOffered = DS_CANONICAL_NAME_EX then  
        rt := LookupCanonicalName(CanonicalNameFromCanonicalNameEx(name))  
        domainName := DomainNameFromCanonicalName(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, true, name)
        domainName := DomainNameFromUPN(name)
    else
        rt := null
    endif

    result.pName^ := null
    result.pDomain^ := null
    result.status := DS_NAME_NO_ERROR

    if rt = null and domainName ≠ null then
        result.status := DS_NAME_ERROR_DOMAIN_ONLY
        if formatOffered in {DS_NT4_ACCOUNT_NAME, DS_USER_PRINCIPAL_NAME,
            DS_SERVICE_PRINCIPAL_NAME, DS_SID_OR_SID_HISTORY_NAME,
            DS_STRING_SID_NAME, DS_USER_PRINCIPAL_NAME_AND_ALTSECID} then
            if IsDomainNameInTrustedForest(domainName, referredDomain) then
                result.pDomain^ := referredDomain
                if DS_NAME_FLAG_TRUST_REFERRAL in flags then
                    result.status := DS_NAME_ERROR_TRUST_REFERRAL
                else
                    result.status := DS_NAME_ERROR_DOMAIN_ONLY
                endif
            endif
        endif
        return result
    endif

    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, formatDesired)

```



```

if names = null and
    foreignSecurityPrincipal in obj!objectClass and
    obj!SID ≠ null and
    DS_NAME_FLAG_PRIVATE_RESOLVE_FPOS in flags and
    formatDesired in { DS_NT4_ACCOUNT_NAME, DS_DISPLAY_NAME,
        DS_CANONICAL_NAME, DS_CANONICAL_NAME_EX, DS_USER_PRINCIPAL_NAME,
        DS_USER_PRINCIPAL_NAME_FOR_LOGON, DS_SERVICE_PRINCIPAL_NAME} then
/* Found a foreign security principal for which the desired name is not
 * included. Use the LSAT protocol to lookup the name. Note: For any
 * desired format, it can only return either DS_CANONICAL_NAME or
 * DS_CANONICAL_NAME_EX. */
if (formatDesired=DS_CANONICAL_NAME_EX) then
    fCanonicalEx := true
else
    fCanonicalEx := false
endif
result := LookupFPO(fCanonicalEx, obj, result)
return result
endif

if names = null then
/* Could not construct the required name format. */
result.status := DS_NAME_ERROR_NO_MAPPING
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
/* The type of the object needs to be specified 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
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

return result

```

4.1.4.2.11 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_GCVERIFY 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

```

4.1.4.2.12 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

```

4.1.4.2.13 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

```

4.1.4.2.14 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 NT,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(false, 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

```

4.1.4.2.15 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
```

4.1.4.2.16 LookupUnknownName

```
procedure LookupUnknownName(
    flags: DWORD,
    name: unicodestring,
    formatDesired: DWORD): DS_NAME_RESULT_ITEMW
```

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 and produces the output name in the given output format.

```
result: DS_NAME_RESULT_ITEMW
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, formatDesired, name)
    if result.status # DS_NAME_ERROR_NOT_FOUND then
        return result
    endif
endfor
return result
```

4.1.4.2.17 LookupUPNAndAltSecID

```
procedure LookupUPNAndAltSecID(
    flags: DWORD,
    IncludingAltSecID: boolean,
    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, rt1, rt2: set of DSName

/* Try lookup by userPrincipalName and altSecurityIdentities
 * or by only userPrincipalName depending on what is
 * requested */
if IncludingAltSecID then
    rt1 := LookupAttr(flags, userPrincipalName, name)
    rt2 := LookupAttr(flags, altSecurityIdentities, name)
```

```

    rt := rt1 + rt2
else
    rt := LookupAttr(flags, userPrincipalName, name)
endif

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

```

4.1.4.2.18 LookupFPO

```

procedure LookupFPO(
    fCanonicalEx: boolean,
    obj: DSName,
    result: DS_NAME_RESULT_ITEMW
): DS_NAME_RESULT_ITEMW

```

Informative summary of behavior: LookupFPO is a helper function that attempts to resolve the domain and account name of an object, with an appropriate status value.

```

pReferencedDomain: PLSAPR_REFERENCED_DOMAIN_LIST
TranslatedName: LSAPR_TRANSLATED_NAMES_EX
NtStatus: NTSTATUS

pReferencedDomain := null
TranslatedName := null

NtStatus := TranslateFPOToName(obj, ADR(pReferencedDomain),
                                ADR(TranslatedName))
if (NtStatus = 0x0 and pReferencedDomain ≠ null and TranslatedName ≠ null)
then
    result.pDomain^ :=
        pReferencedDomain^.Domains[TranslatedName.DomainIndex].Name
    if fCanonicalNameEx then
        result.pName^ := result.pDomain^ + {"\n"} + TranslatedName.Name
    else
        result.pName^ := result.pDomain^ + {"\""} + TranslatedName.Name
    endif
    result.status := DS_NAME_ERROR_IS_FPO
    return result
else
    result.status := DS_NAME_ERROR_RESOLVING
endif

/* leave result as-is. */

```

```
return result
```

4.1.4.2.19 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=alserter,apppgmt,cisvc"}
```

If spn is "alserter/fabrikam.com", then the procedure returns "host/fabrikam.com".

4.1.4.2.20 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.

4.1.4.2.21 RetrieveDCSuffixFromDn

```
procedure RetrieveDCSuffixFromDn(dn: unicodestring): unicodestring
```

The RetrieveDCSuffixFromDn procedure parses the distinguished name (DN) syntactically and returns the suffix that consists entirely of the DN components whose attribute type is "DC". For example, given "CN=Administrator,CN=Users,DC=fabrikam,DC=com", this procedure would return "DC=fabrikam,DC=com".

4.1.4.2.22 UserNameFromUPN

```
procedure UserNameFromUPN(upn: uncodestring): uncodestring
```

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.

4.1.4.2.23 TranslateFPOToName

```
procedure TranslateFPOToName(  
  obj: DSName,  
  ppReferencedDomains: PLSAPR_REFERENCED_DOMAIN_LIST*,  
  pTranslatedNames: PLSAPR_TRANSLATED_NAMES_EX  
): NTSTATUS
```

Informative summary of behavior: The TranslateFPOToName procedure performs an LsarLookupSids2 call ([\[MS-LSAT\]](#) section 3.1.4.10) to translate *obj* to its Microsoft Windows NT® 4.0 operating system account name and domain.

```
hlsaPolicy: LSAPR_HANDLE  
mappedCount: unsigned long  
systemName: unicodeString  
objectAttributes: LSAPR_OBJECT_ATTRIBUTES  
desiredAccess: DWORD  
sidEnumBuffer: LSAPR_SID_ENUM_BUFFER  
sidInfo: LSAPR_SID_INFORMATION  
NtStatus: NTSTATUS  
  
sidEnumBuffer.Entries := 1  
sidInfo.Sid := obj!Sid  
sidEnumBuffer.SidInfo := ADR(sidInfo)  
  
systemName := ""  
objectAttributes.Length := 0  
objectAttributes.RootDirectory := null  
objectAttributes.ObjectName := null  
objectAttributes.attributes := 0  
objectAttributes.SecurityDescriptor := null  
objectAttributes.SecurityQualityOfService := null  
desiredAccess := 0x00000800  
  
NtStatus := LsarOpenPolicy2(systemName, ADR(objectAttributes),  
                           desiredAccess, ADR(hlsaPolicy))  
if 0x0 = NtStatus then  
  NtStatus := LsarLookupSids2(hlsaPolicy, ADR(sidEnumBuffer),  
                             ppReferencedDomains, pTranslatedNames,  
                             0x1, ADR(mappedCount), 0x0, 0x2)  
endif  
  
If hlsaPolicy ≠ null  
  LsarClose(ADR(hlsaPolicy))  
  
return NtStatus
```


4.1.4.2.24 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 \(section 4.1.4.2.10\)](#).

```
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
ULONG result

ValidatedDRSInput(hDrs, 12)

pdwOutVersion^ := 1
pmsgOut^.V1.pResult^.cItems := 0
pmsgOut^.V1.pResult^.rItems := null
if dwInVersion # 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
if DS_NAME_FLAG_GCVERIFY in msgIn.dwFlags and
    not IsGC() then
    return ERROR_DS_GCVERIFY_ERROR
endif

/* Enable FPO resolution for non-DC callers. */
if ClientUUID(hDrs) = NTDSAPI_CLIENT_GUID then
    msgIn.dwFlags := msgIn.dwFlags + {DS_NAME_FLAG_PRIVATE_RESOLVE_FPOS}
endif

result = CrackNames(pmsgIn^.V1, ADR(msgOut))
if(result = ERROR_SUCCESS) then
    pmsgOut^.V1.pResult := ADR(msgOut)
endif
return result

```

4.1.4.4 Examples of the IDL_DRSCrackNames Method

When user "Kim Akers" logs on to the computer MS1.Contoso.com using her Microsoft Windows NT® 4.0 operating system account name "CONTOSO\kimakers", the domain controller needs to obtain a fully qualified domain name (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 for the domain NC CONTOSO.COM on DC1 by performing the following LDAP search: Querying the [crossRef](#) object on the NC root object for domain NC CONTOSO.COM on DC1:

- ldap_search_s("CN=CONTOSO,CN=Partitions,CN=Configuration,DC=contoso,DC=com", baseObject, "(objectClass=crossRef)", [objectClass, nCName, dnsRoot, nETBIOSName])
- 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 = 0
 - 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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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: The domain for which the client requests information. The domain can be an FQDN or a NetBIOS domain name. 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_VFFFFFFFFF

The **DRS_MSG_DCINFOREPLY_VFFFFFFFFF** 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_VFFFFFFFFF;
```

cItems: The count of items in the **rItems** array.

rItems: Information about the active LDAP connections.

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: A Boolean value that indicates whether or not the machine is a domain controller. This value 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: A Boolean value that indicates whether or not the machine is a domain controller.
This value 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: A Boolean value that indicates whether or not the machine is a domain controller.
This value 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_FFFFFFFF;

```

IPAddress: The IPv4 address of the client that established the LDAP connection to the server. If the client is connected with IPv6, this field MUST be 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: Unused. MUST be 0 and ignored.

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 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.

For information about the Microsoft Windows® versions in which information levels were introduced and supported, see the following behavior note. [<13>](#)

Note The server behavior of the **IDL_DRSDomainControllerInfo** method uses the [CrackNames](#) procedure defined in section [4.1.4.2.9](#).

```

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
serversContainer, crObj, dcObj, dsaObj, svrObj, siteObj, obj, v: DSName
lc: DS_DOMAIN_CONTROLLER_INFO_FFFFFFFF

```

```

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_V1
crackOut: DS_NAME_RESULTW
outV: DWORD
userAccountControl: set of integer

ValidateDRSInput(hDrs, 16)

msgIn := pmsgIn^.V1
infoLevel := msgIn.InfoLevel
domainName := msgIn.Domain

pdwOutVersion^ := infoLevel

if infoLevel = 1 then
    pmsgOut^.V1.cItems := 0
    pmsgOut^.V1.rItems := null
else if infoLevel = 2 then
    pmsgOut^.V2.cItems := 0
    pmsgOut^.V2.rItems := null
else if infoLevel = 3 then
    pmsgOut^.V3.cItems := 0
    pmsgOut^.V3.rItems := null
else if infoLevel = 0xFFFFFFFF then
    pmsgOut^.VFFFFFFFF.cItems := 0
    pmsgOut^.VFFFFFFFF.rItems := null
endif

if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif

if not (infoLevel in {1,2,3,0xFFFFFFFF}) then
    return ERROR_INVALID_PARAMETER
endif

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
    endforeach
endif

```

```

        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. If this doesn't
 * find a match, 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!dnsRoot = 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.dwFlags := 0
    crackMsgIn.formatOffered := DS_UNKNOWN_NAME
    crackMsgIn.formatDesired := DS_FQDN_1779_NAME
    crackMsgIn.cNames := 3
    crackMsgIn.rpNames[0] := domainName
    crackMsgIn.rpNames[1] := domainName + "\"
    crackMsgIn.rpNames[2] := domainName + "/"

    /* Call IDL_DRSCrackNames as a local procedure. */
    CrackNames(crackMsgIn, ADR(crackOut))

    i := 0
    while i < 3 and not found
        if crackOut.rItems[i].status = DS_NAME_NO_ERROR
            then
                if crackOut.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 dcObj in dcSet
        rI1 := ADR(pmsgOut^.V1.rItems[i])

        rI1^.DnsHostName := dcObj!dNSHostName
        rI1^.ComputerObjectName := dcObj.dn
        /* sAMAccountName excluding the "$" at the end. */
        rI1^.NetbiosName := SubString(dcObj!sAMAccountName, 0,
            dcObj!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" */
        svrObj :=
            select one v from all where v.dn in dcObj!serverReferenceBL
            and DescendantObject(v, "CN=NTDS Settings") ≠ null
        if svrObj = null then
            svrObj :=
                select one v from all where v.dn in dcObj!serverReferenceBL
        endif
        if svrObj ≠ null then
            rI1^.ServerObjectName := svrObj.dn
            serversContainer :=
                select one o from all where o!objectGUID = svrObj!parent
            siteObj := serversContainer!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 dcObj in dcSet
            rI2 := ADR(pmsgOut^.V2.rItems[i])

            rI2^.DnsHostName := dcObj!dNSHostName
            rI2^.ComputerObjectName := dcObj.dn
            /* sAMAccountName excluding the "$" at the end. */
            rI2^.NetbiosName := SubString(dcObj!sAMAccountName, 0,
                dcObj!sAMAccountName.length-1)
            rI2^.ComputerObjectGUID := dcObj.guid
            rI2^.fDsEnabled := true
        endfor
    endif
endif

```

```

/* select a server object from the serverReferenceBL, it is
   preferred that the server object has a child object with
   CN "NTDS Settings" */
svrObj :=
    select one v from all where v.dn in dcObj!serverReferenceBL
        and DescendantObject(v, "CN=NTDS Settings") ≠ null
if svrObj = null then
    svrObj :=
        select one v from all where v.dn in dcObj!serverReferenceBL
endif
if svrObj ≠ null then
    rI2^.ServerObjectName := svrObj.dn
    rI2^.ServerObjectGuid := svrObj.guid

serversContainer :=
    select one o from all where o!objectGUID = svrObj!parent
siteObj := serversContainer!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^.V3.cItems := number(dcSet)

i := 0
foreach dcObj in dcSet
    rI3 := ADR(pmsgOut^.V3.rItems[i])

    rI3^.DnsHostName := dcObj!dNSHostName
    rI3^.ComputerObjectName := dcObj.dn
    /* sAMAccountName excluding the "$" at the end. */
    rI3^.NetbiosName := SubString(dcObj!samAccountName, 0,
        dcObj!samAccountName.length-1)
    rI3^.ComputerObjectGUID := dcObj.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 Settings" */
svrObj :=
    select one v from all where v.dn in dcObj!serverReferenceBL
        and DescendantObject(v, "CN=NTDS Settings") ≠ null
if svrObj = null then
    svrObj :=
        select one v from all where v.dn in dcObj!serverReferenceBL
endif
if svrObj ≠ null then
    rI3^.ServerObjectName := svrObj.dn
    rI3^.ServerObjectGuid := svrObj.guid

serversContainer :=

```

```

        select one o from all where o!objectGUID = svrObj!parent
        siteObj := serversContainer!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 ∩
            dcObj!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 for the domain NC CONTOSO.COM on DC2 by performing an LDAP search with base scope on the DN

'CN=CONTOSO,CN=Partitions,CN=Configuration,DC=contoso,DC=com':

- 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 by performing an LDAP search with base scope on the DN 'CN=DC1,OU=Domain Controllers,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 operating system;
 - 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 by performing an LDAP search with base scope on the DN 'CN=DC2,OU=Domain Controllers,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
 - fDsEnabled: 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
- fDsEnabled: 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: A pointer to the request message.

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

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE, ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.

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, which are presented in little-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
X	X	X	X	X	X	D P	A 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: Unused. MUST be zero and ignored.

AS (DS_KCC_FLAG_ASYNC_OP, 0x00000001): Request the KCC to run, then return immediately.

DP (DS_KCC_FLAG_DAMPED, 0x00000002): Request the KCC to run unless there is already such a request pending according to implementation-defined rules. Implementations MAY choose to ignore this flag and always request the KCC to run.

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. <14> See [\[MS-ADTS\]](#) section 7.2.2 for more information related to the tasks performed by the KCC upon receipt of an **IDL_DRSExecuteKCC** request.

```
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

ValidatedDRSInput(hDrs, 18)

/* Validate the request version */
if dwInVersion ≠ 1 then
    return ERROR_DS_DRA_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

if msgIn.dwFlags = DS_KCC_FLAG_ASYNC_OP then
    Asynchronous Processing: Initiate a logical thread of control
    to process the remainder of this request asynchronously
    return 0
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: 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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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;
    [string] LPWSTR szScriptBase;
} DRS_MSG_FINISH_DEMOTIONREQ_V1;
```

dwOperations: Zero or more of the following bit flags, which are presented in little-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	U 2	U 1	D	C	R	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	F	X	X	X	X	X	X	X	X

X: Unused. MUST be zero and ignored.

R (DS_DEMOTE_ROLLBACK_DEMOTE, 0x00000001): Undo the effects of [IDL DRSInitDemotion](#). If present, any other flags present (except for `DS_DEMOTE_OPT_FAIL_ON_UNKNOWN`) are ignored.

C (DS_DEMOTE_COMMIT_DEMOTE, 0x00000002): Discontinue being a DC for the current DC instance by stopping all AD LDS protocols.

D (DS_DEMOTE_DELETE_CSMETA, 0x00000004): Delete the [nTDSDSA](#) object for this DC; see [RemoveADLDSServer \(section 4.1.7.2.1\)](#).

U1 (DS_DEMOTE_UNREGISTER_SCPS, 0x00000008): Delete any [serviceConnectionPoint](#) objects for this DC from AD DS; see [RemoveADLDSSCP \(section 4.1.7.2.2\)](#).

U2 (DS_DEMOTE_UNREGISTER_SPNS, 0x00000010): Delete any AD LDS SPNs from the object (in the external AD DS domain) that corresponds to the security principal that the AD LDS service is running as; see [RemoveADLDSPNs \(section 4.1.7.2.3\)](#).

F (DS_DEMOTE_OPT_FAIL_ON_UNKNOWN_OP, 0x80000000): If this flag is present, then the request fails.

uuidHelperDest: Unused. Must be NULL GUID and ignored.

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. This MUST include any value from the input element **DRS_MSG_FINISH_DEMOTIONREQ_V1.dwOperations** whose corresponding operations see pseudocode in section [4.1.7.3](#) succeeded.

dwOpFailed: The set of operations that failed during demotion. This may include the same values as the **dwOperationsDone** field. This MUST include any value from the input element

DRS_MSG_FINISH_DEMOTIONREQ_V1.dwOperations whose corresponding operations (see pseudocode in section [4.1.7.3](#)) failed.

dwOpError: The Win32 error code (as specified in [\[MS-ERREF\]](#) section 2.2) of the first failed operation (if any), from the following operations: DS_DEMOTE_ROLLBACK_DEMOTE, DS_DEMOTE_COMMIT_DEMOTE, DS_DEMOTE_DELETE_CSMETA, or DS_DEMOTE_UNREGISTER_SCPS.

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): boolean
```

The RemoveADLDSSPNs procedure connects to an AD DS DC and attempts to delete any SPN values registered for the AD LDS DC on the object (in the external AD DS domain) that corresponds to the security principal that the AD LDS service is running as. 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 TRUE; otherwise, it returns FALSE, indicating that a batch file was created in the folder specified by the *szScriptBase* parameter. This batch file contains commands that an administrator can run to clean up the SPNs.

Note When the procedure fails to create a batch file for any reason, RemoveADLDSSPNs returns TRUE.

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 it undoes the effects of the first phase of removal (performed by [IDL_DRSInitDemotion](#)).<15>

```
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
res: boolean

ValidatedDRSInput(hDrs, 27)

if dwInVersion # 1 then
    return ERROR_INVALID_PARAMETER
endif

if pmsgIn = null then
    return ERROR_INVALID_PARAMETER
endif

msgIn := pmsgIn^.V1
if DS_DEMOTE_OPT_FAIL_ON_UNKNOWN_OP in msgIn.dwOperations 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_DS_DRA_ACCESS_DENIED
endif

pdwOutVersion^ := 1
msgOut.dwOperationDone := 0
msgOut.dwOpFailed := 0
msgOut.dwOpError := ERROR_SUCCESS

if DS_DEMOTE_ROLLBACK_DEMOTE in msgIn.dwOperations then
    /* Begin operations corresponding to dwOperations value of DS_DEMOTE_ROLLBACK_DEMOTE */

    /* undo the effects of IDL_DRSInitDemotion */

    dc.fEnableUpdates := TRUE

    msgOut.dwOperationDone :=
        msgOut.dwOperationDone + {DS_DEMOTE_ROLLBACK_DEMOTE}
    msgOut.dwOpError := ERROR_SUCCESS

```

```

/* no other operations are allowed on rollback */
/* End operations corresponding to dwOperations value of DS_DEMOTE_ROLLBACK_DEMOTE */

else
  if DS_DEMOTE_COMMIT_DEMOTE in msgIn.dwOperations then
    /* Begin operations corresponding to dwOperations value of DS_DEMOTE_COMMIT_DEMOTE */

    After this call to IDL_DRSFinishDemotion completes, the server should discontinue being a
    DC, which for AD LDS means stopping the MS-DRSR protocol, the MS-DSSP protocol, the LDAP
    protocol, and if they are already enabled also the MS-ADCAP protocol, the WS-Enumeration
    protocol, the WS-Transfer protocol, the MS-WSTIM protocol, the MS-WSDS protocol, and the MS-
    WSPELD protocol. In addition, the state model, constraints and processing rules, and so on,
    in MS-ADTS should also be stopped.

    msgOut.dwOperationDone :=
      msgOut.dwOperationDone + {DS_DEMOTE_COMMIT_DEMOTE}

    msgOut.dwOpError := ERROR_SUCCESS
    /* End operations corresponding to dwOperations value of DS_DEMOTE_COMMIT_DEMOTE */

  endif
  if DS_DEMOTE_DELETE_CSMETA in msgIn.dwOperations then
    /* Begin operations corresponding to dwOperations value of DS_DEMOTE_DELETE_CSMETA */

    ret := RemoveADLDSSTServer()
    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
  /* End operations corresponding to dwOperations value of DS_DEMOTE_DELETE_CSMETA */
endif
  if DS_DEMOTE_UNREGISTER_SCPS in msgIn.dwOperations then
    /* Begin operations corresponding to dwOperations value of DS_DEMOTE_UNREGISTER_SCPS */

    ret := RemoveADLDSSTSCP()
    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
  /* End operations corresponding to dwOperations value of DS_DEMOTE_UNREGISTER_SCPS */
endif
  if DS_DEMOTE_UNREGISTER_SPNS in msgIn.dwOperations then
    /* Begin operations corresponding to dwOperations value of DS_DEMOTE_UNREGISTER_SPNS */

    res := RemoveADLDSSTSPNs(msgIn.szScriptBase)
    if res = TRUE then
      msgOut.dwOperationDone :=

```

```

        msgOut.dwOperationDone + {DS_DEMOTE_UNREGISTER_SPNS}
    else
        msgOut.dwOpFailed =
            msgOut.dwOpFailed + {DS_DEMOTE_UNREGISTER_SPNS}
    endif
    /* End operations corresponding to dwOperations value of DS_DEMOTE_UNREGISTER_SPNS */
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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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](#) of the object whose reverse membership is being requested, plus the [DSNames](#) 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, which are presented in little-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	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: Unused. MUST be zero and ignored.

A (DRS_REVMEMB_FLAG_GET_ATTRIBUTES, 0x00000001): 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 groups that are confined to a given domain, excluding built-in groups and domain-local groups. See [\[MS-ADSC\]](#) section 2.14.

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: Non-transitive 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 errorCode;
    [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 $v[0], v[1], \dots, v[k]$, 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 $i=1,2,\dots,k$.

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 GetDSNameOfEnterpriseRODCsGroup

```
procedure GetDSNameOfEnterpriseReadonlyDomainControllerGroup(): DSName
```

This procedure constructs a SID s consisting of the domain SID of the root domain and the relative identifier (RID) of the Enterprise Read-only Domain Controllers Group (as defined in [\[MS-ADTS\]](#) section 7.1.1.6.14), and returns the **DSName** of the object o for which $o!objectSid = s$. If no such object o exists, this procedure returns null.

4.1.8.2.5 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.

4.1.8.2.6 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_TYPE_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

```

4.1.8.2.7 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 the 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

ValidatedDRSInput(hDrs, 9)

pdwOutVersion^ := 1
pmsgOut^.V1.errCode := 0
pmsgOut^.V1.cDsNames := 0
pmsgOut^.V1.cSidHistory := 0
pmsgOut^.V1.ppDsNames := null
pmsgOut^.V1.pAttributes := null
pmsgOut^.V1.ppSidHistory := null

msgIn := pmsgIn^.V1

if dwInVersion # 1 then
    return ERROR_DS_DRA_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

/* 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
        if ((u!userAccountControl & ADS_UF_WORKSTATION_TRUST_ACCOUNT =
            ADS_UF_WORKSTATION_TRUST_ACCOUNT) or
            (u!userAccountControl & ADS_UF_PARTIAL_SECRETS_ACCOUNT =
            ADS_UF_PARTIAL_SECRETS_ACCOUNT))
            wSet := wSet + GetDSNameOfEnterpriseRODCsGroup()
        endif
    endif
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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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

ValidatedDRSInput(hDrs, 21)

pdwOutVersion^ := 1
pMsgOut^.V1.Count := 0
pMsgOut^.V1.Replies := null

if dwInVersion # 1 then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif

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: 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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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` \neq `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 [<16>](#) 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 [<17>](#) 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 [<18>](#) 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 [<19>](#) 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. [<20>](#)
- 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. [<21>](#)
- Allocate RIDs (`EXOP_FSMO_REQ_RID_ALLOC`): Allocates a new block of RIDs to the client DC. [<22>](#)
- Replicate Single Object (`EXOP_REPL_OBJ`): Adds any changes to the specified object to the response. [<23>](#)
- Replicate Single Object including Secret Data (`EXOP_REPL_SECRETS`): Adds any changes to the specified object to the response. In addition, it also adds the secret attribute values of the specified object to the response, regardless of whether they have recent changes. See the `IsSecretAttribute` procedure in section [4.1.10.3.12](#) for a list of these attributes. [<24>](#)

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, V6, or V9 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;
    [case(10)]
        DRS_MSG_GETCHGREQ_V10 V10;
} 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).

V10: Version 10 request (Windows Server 2008 R2 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](#).<25>

```
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 (section [4.1.10.2.18](#)).

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 a superset of [DRS_MSG_GETCHGREQ_V3](#).<26>

```
typedef struct {
    UUID uuidTransportObj;
    [ref] MTX_ADDR* pmtxReturnAddress;
    DRS_MSG_GETCHGREQ_V3 V3;
} DRS_MSG_GETCHGREQ_V4;
```

uuidTransportObj: The [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.

```
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 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 (section [4.1.10.2.18](#)).

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 a superset of [IDL_DRS_MSG_GETCHGREQ_V4](#).<27>

```
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: The [objectGUID](#) of the [interSiteTransport](#) object that 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 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 (section [4.1.10.2.18](#)).

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_GETCHGREQ_V10

The **DRS_MSG_GETCHGREQ_V10** structure defines the request message sent to the [IDL_DRSGetNCChanges](#) method. This message version is a superset of [DRS_MSG_GETCHGREQ_V8](#).

```
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;
    ULONG ulMoreFlags;
} DRS_MSG_GETCHGREQ_V10;
```

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 (section [4.1.10.2.18](#)).

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.

ulMoreFlags: A [DRS_MORE_GETCHGREQ_OPTIONS](#) bit field.

4.1.10.2.8 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).

4.1.10.2.9 DRS_MSG_GETCHGREPLY_V1

The **DRS_MSG_GETCHGREPLY_V1** structure defines the response message received from the [IDL DRSGetNCChanges](#) method.

```
typedef struct {
    UUID uuidDsaObjSrc;
    UUID uuidInvocIdSrc;
    [unique] 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: The 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**.

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 (section [4.1.10.2.17](#)).

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 applies to its NC replica.

fMoreData: False if and only if this is the last response in a replication cycle.

4.1.10.2.10 DRS_MSG_GETCHGREPLY_V2

The **DRS_MSG_GETCHGREPLY_V2** structure defines the compressed [DRS MSG GETCHGREPLY V1](#) message received from the [IDL_DRSGetNCChanges](#) method.

```

typedef struct {
    DRS_COMPRESSED_BLOB CompressedV1;
} DRS_MSG_GETCHGREPLY_V2;

```

CompressedV1: Compressed **DRS_MSG_GETCHGREPLY_V1** response.

4.1.10.2.11 DRS_MSG_GETCHGREPLY_V6

The **DRS_MSG_GETCHGREPLY_V6** structure defines the response message received from the [IDL_DRSGetNCChanges](#) method. This message version is a superset of [DRS MSG GETCHGREPLY V1](#).

```

typedef struct {
    UUID uuidDsaObjSrc;
    UUID uuidInvocIdSrc;
    [unique] 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: The 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 (section [4.1.10.2.17](#)).

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 applies 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.

4.1.10.2.12 DRS_MSG_GETCHGREPLY_V7

The **DRS_MSG_GETCHGREPLY_V7** structure defines a compressed [DRS_MSG_GETCHGREPLY_V6](#) message received from the [IDL_DRSGetNCChanges](#) method.

```
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.

4.1.10.2.13 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.

4.1.10.2.14 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_UNUSED = 1,
    DRS_COMP_ALG_MSZIP = 2,
    DRS_COMP_ALG_WIN2K3 = 3
} DRS_COMP_ALG_TYPE;

```

DRS_COMP_ALG_NONE: No compression.

DRS_COMP_ALG_UNUSED: Unused. MUST not be used.

DRS_COMP_ALG_MSZIP: **MSZIP** algorithm.

DRS_COMP_ALG_WIN2K3: Windows Server 2003 compression.

4.1.10.2.15 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.

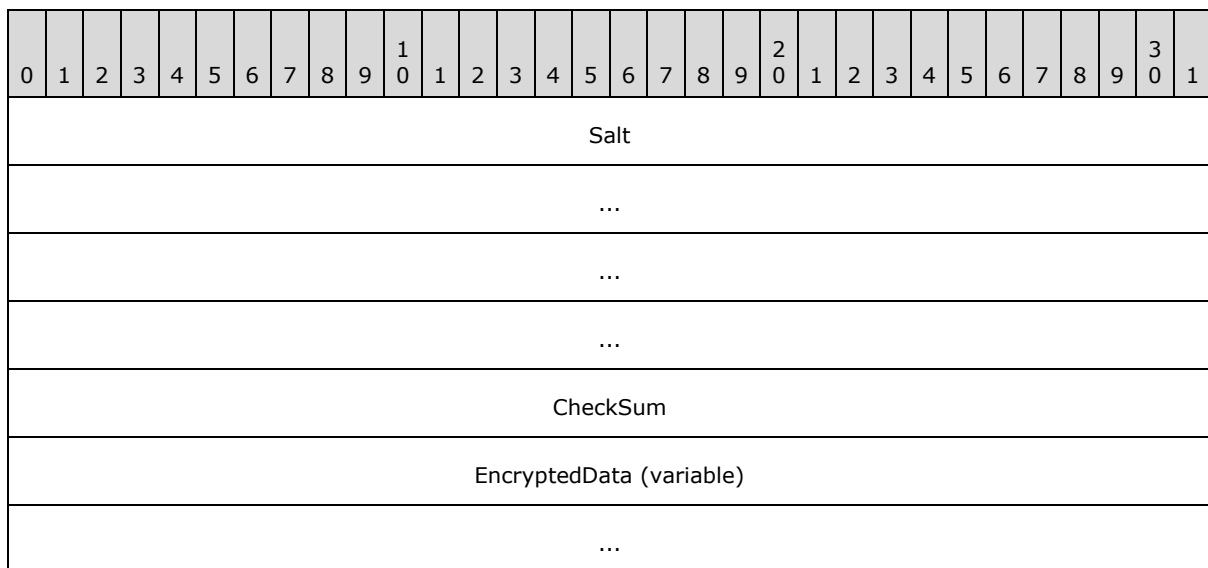
pbCompressedData: Compressed byte stream.

Padding: Data is padded with zeros, if necessary, so that the block ends on an alignment boundary of [LONG](#).

4.1.10.2.16 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;
```



Salt (16 bytes): A 128-bit randomly generated value.

Checksum (4 bytes): A 32-bit **CRC32 checksum** of the data that is encrypted along with the data.

EncryptedData (variable): A variable-length byte array that represents the encrypted value.

4.1.10.2.17 EXOP_ERR Codes

The following values are error codes for an extended operation request to the [IDL DRSGetNCChanges](#) method.

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)
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)

4.1.10.2.18 EXOP_REQ Codes

The following values are request codes for extended operation.

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)

4.1.10.2.19 PROPERTY_META_DATA

The **PROPERTY_META_DATA** structure contains attribute and stamp information. For more information, see section [4.1.10.5.9](#).

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 is greater than stamp2, the procedure returns an integer with a value greater than 0. If stamp1 is equal to stamp2, the procedure returns 0. If stamp1 is less than stamp2, then the procedure returns an integer value less than 0. Refer to section [5.11](#) for details on the comparison of [AttributeStamps](#).

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](#) 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](#) 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 := AbstractAttrStampFromConcereteAttrStamp(  
        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 [LinkValueStamps](#), 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

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

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

```

4.1.10.3.15 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 its new 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)
```

The RemoveAttrVal procedure removes the value attributeValue from the attribute attr on the object with [DSName](#) obj.

4.1.10.3.19 SetAttrStamp

```
procedure SetAttrStamp(  
  obj: DSName,  
  attr: ATTRTYP,  
  stamp: AttributeStamp)
```

The SetAttrStamp procedure sets the [AttributeStamp](#) for the attribute attr on the object obj to stamp.

4.1.10.3.20 SetAttrVal

```
procedure SetAttrVal(  
  obj: DSName,  
  attr: ATTRTYP,  
  attributeValue: attribute value)
```

The SetAttrVal procedure sets the value attributeValue for the attribute attr on the object obj.

4.1.10.3.21 SetLinkStamp

```
procedure SetLinkStamp(  
  obj: DSName,  
  attr: ATTRTYP,  
  val: attribute value,  
  stamp: AttributeStamp)
```

The SetLinkStamp procedure sets the [LinkValueStamp](#) for the attribute value val on the attribute attr on the object obj to *stamp*.

4.1.10.4 Client Behavior When Sending the IDL_DRSGetNCChanges 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 ≠ 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,
  ulMoreFlags: 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](#) 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 operating system ignore this flag.
- **DRS_GET_ALL_GROUP_MEMBERSHIP**: Requests that the server ship all group membership. If this flag is not specified, the server ships only universal group membership. Servers prior to Windows Server 2008 ignore this flag.
- **DRS_REF_GCSPN**: Requests that the server add 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 notifying client DC contacts the server DC using the service principal name that begins with "GC" (section [2.2.3.2](#)).
- `ulMoreFlags`: 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_GET_TGT`.
 - **DRS_GET_TGT**: Signals the server to send all updates for the **target object** of a link value update before sending the link value update.
- `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 *version* parameter.

```

msgIn: DRS_MSG_GETCHGREQ_V10
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, "nTDSDSA Object"*/
if NTDSDSA_OPT_DISABLE_INBOUND_REPL in DSAObj()!options and
    not DRS_SYNC_FORCED in ulFlags then
    return ERROR_DS_DRA_SINK_DISABLED
endif

if IsAdlds() and ServerExtensions(hDrs).ConfigObjGUID != NULLGUID and
ServerExtensions(hDrs).ConfigObjGUID != ConfigNC().GUID then
    return ERROR_DS_DIFFERENT_REPL_EPOCHS;

```

```

endif

msgIn.ulMoreFlags := ulMoreFlags
msgIn.cMaxObjects := cMaxObjects
msgIn.cMaxBytes := cMaxBytes
msgIn.ulExtendedOp := 0
msgIn.uuidDsaObjDest := dc.serverGuid
msgIn.pNC := ADR(nc)
msgIn.liFsmoInfo := 0

if (ObjExists(nc)) then
    msgIn.pUpToDateVecDest :=
        ConcreteUTDFromAbstractUTD(nc!replUpToDateVector)
else
    msgIn.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
    msgIn.usnvecFrom := 0
    msgIn.uuidInvocIdSrc := 0
else
    msgIn.usnvecFrom := rf.usnVec
    msgIn.uuidInvocIdSrc := rf.uuidInvocId
endif

if AmIRODC() then
    if DRS_WRIT_REP in ulFlags then
        return ERROR_DS_DRA_INVALID_PARAMETER
    endif
    ext := ServerExtensions(hDrs)
    if not DRS_EXT_LH_BETA2 in ext.dwFlags and
        msgIn.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
msgIn.ulFlags := ulFlags

if (DRS_WRIT_REP in ulFlags) or
    (not DRS_SYNC_PAS in ulFlags) then
    msgIn.pPartialAttrSetEx := null
else
    msgIn.pPartialAttrSetEx := ConcretePASFromAbstractPAS(rf.pasData)
endif

/* set msgIn.pPartialAttrSet field */
if ObjExists(nc) and nc!partialAttributeSet ≠ null then

```

```

        msgIn.pPartialAttrSet := ConcretePASFromAbstractPAS(
                                nc!partialAttributeSet)
    else
        if (NCT_GC_PARTIAL in ncType and
            NCT_FILTERED_ATTRIBUTE_SET in ncType) then
            msgIn.pPartialAttrSet := FilteredGCPAS()
        else if NCT_FILTERED_ATTRIBUTE_SET in ncType then
            msgIn.pPartialAttrSet := FilteredPAS()
        else if NCT_GC_PARTIAL in ncType then
            msgIn.pPartialAttrSet := GCPAS()
        else
            msgIn.pPartialAttrSet := null
        endif
    endif
endif

msgIn.PrefixTableDest = ConcretePTFromAbstractPT(dc.prefixTable)

/* Add schema signature to msgIn.PrefixTableDest */
schemaSignature := SchemaNC()!schemaInfo
prefixEntry.ndx := 0
prefixEntry.prefix.length := schemaSignature.length
prefixEntry.prefix.element := elements of schemaSignature
Append prefixEntry to msgIn.PrefixTableDest.pPrefixEntry
msgIn.PrefixTableDest.PrefixCount :=
    msgIn.PrefixTableDest.PrefixCount + 1

if version = 5 then
    msgRequest.V5 := msgIn
    msgRequest.V5.pUpToDateVecDestV1 := msgIn.pUpToDateVecDest
else if version = 8 then
    msgRequest.V8 := msgIn
else
    msgRequest.V10 := msgIn
endif
endif

```

4.1.10.4.2 ReplSingleObjRequestMsg

```

procedure ReplSingleObjRequestMsg(
    hDrs: DRS_HANDLE,
    version: DWORD,
    nc: DSName,
    object: DSName,
    rf: RepsFrom,
    ulFlags: ULONG,
    ulMoreFlags: 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](#) 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
msgIn: DRS_MSG_GETCHGREQ_V10
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
    msgIn.ulExtendedOp := EXOP_REPL_SECRETS
else
    msgIn.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
        msgIn.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

msgIn.ulFlags := ulFlags
msgIn.ulMoreFlags := ulMoreFlags
msgIn.cMaxObjects := cMaxObjects
msgIn.cMaxBytes := cMaxBytes
msgIn.uuidDsaObjDest := dc.serverGuid
msgIn.pNC := ADR(object)
msgIn.liFsmoInfo := 0
msgIn.pUpToDateVecDest :=
    ConcreteUTDFromAbstractUTD(nc!replUpToDateVector)
msgIn.pPartialAttrSetEx := null

/* set msgIn.pPartialAttrSet field */
if ObjExists(nc) and nc!partialAttributeSet ≠ null then
    msgIn.pPartialAttrSet := ConcretePASFromAbstractPAS(
        nc!partialAttributeSet)
else

```

```

    if (NCT_GC_PARTIAL in ncType and
        NCT_FILTERED_ATTRIBUTE_SET in ncType) then
        msgIn.pPartialAttrSet := FilteredGCPAS()
    else if NCT_FILTERED_ATTRIBUTE_SET in ncType then
        msgIn.pPartialAttrSet := FilteredPAS()
    else if NCT_GC_PARTIAL in ncType then
        msgIn.pPartialAttrSet := GCPAS()
    else
        msgIn.pPartialAttrSet := null
    endif
endif

msgIn.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
    msgIn.usnvecFrom := 0
    msgIn.uuidInvocIdSrc := 0
else
    msgIn.usnvecFrom := rf.usnVec
    msgIn.uuidInvocIdSrc := rf.uuidInvocId
endif

if version = 5 then
    msgRequest.V5 := msgIn
    msgRequest.V5.pUpToDateVecDestV1 := msgIn.pUpToDateVecDest
else if version = 8 then
    msgRequest.V8 := msgIn
else
    msgRequest.V10 := msgIn
endif

return 0

```

4.1.10.4.3 PerformExtendedOpRequestMsg

```

procedure PerformExtendedOpRequestMsg (
    hDrs: DRS_HANDLE,
    version: DWORD,
    nc: DSName,
    roleOwnerObject: DSName,
    rf: RepsFrom,
    ulFlags: ULONG,
    ulMoreFlags: 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:

<code>ulExtendedOp</code>	<code>roleOwnerObject</code>
<code>EXOP_FSMO_REQ_ROLE</code>	The DSName of the FSMO role object.
<code>EXOP_FSMO_REQ_RID_ALLOC</code>	The value of the rIDManagerReference attribute of DefaultNC() .
<code>EXOP_FSMO_RID_REQ_ROLE</code>	The value of the rIDManagerReference attribute of DefaultNC() .
<code>EXOP_FSMO_REQ_PDC</code>	DefaultNC() .
<code>EXOP_FSMO_ABANDON_ROLE</code>	The DSName of the FSMO role object.

The procedure returns a Windows error code if it not able to construct `msgRequest`.

```

msgIn: DRS_MSG_GETCHGREQ_V10
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

msgIn.ulFlags := ulFlags
msgIn.ulMoreFlags := ulMoreFlags
msgIn.cMaxObjects := cMaxObjects
msgIn.cMaxBytes := cMaxBytes
msgIn.ulExtendedOp := ulExtendedOp
msgIn.uuidDsaObjDest := dc.serverGuid
msgIn.pNC := ADR(roleOwnerObject)
msgIn.pUpToDateVecDest :=
    ConcreteFromAbstractUTD(nc!replUpToDateVector)
msgIn.pPartialAttrSetEx := null
msgIn.pPartialAttrSet := null
msgIn.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
        msgIn.liFsmoInfo := ridSetReferences!rIDAllocationPool
    else
        msgIn.liFsmoInfo := 0
    endif
else
    msgIn.liFsmoInfo := 0
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
    msgIn.usnvecFrom := 0
    msgIn.uuidInvocIdSrc := 0
else
    msgIn.usnvecFrom := rf.usnVec
    msgIn.uuidInvocIdSrc := rf.uuidInvocId
endif

if version = 5 then
    msgRequest.V5 := msgIn
    msgRequest.V5.pUpToDateVecDestV1 := msgIn.pUpToDateVecDest
else if version = 8 then
    msgRequest.V8 := msgIn
else
    msgRequest.V10 := msgIn
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_V10
ncRoot: DSName
obj: DSName
msgOut: DRS_MSG_GETCHGREPLY_V6
schemaSignature: sequence of BYTE
prefixEntry: PrefixTableEntry
responseSmtAddress: unicodestring
fullReplicaFlags: set of integer
fullReplicaRequest : boolean

ValidatedDRSInput(hDrs, 3)

pdwOutVersion^ := 1
pmsgOut^ := 0

err := TransformInput(ext, dwInVersion, pmsgIn^, msgIn,
    pdwOutVersion, responseSmtAddress)
if err # 0 then
    return err
endif

/* Perform access checks. */
if msgIn.pNC = null then
    return ERROR_DS_DRA_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
endif

```

```

else
    /* Validate extended operation request. */
    if not ObjExists(obj) then
        return ERROR_DS_CANT_FIND_EXPECTED_NC
    endif
endif

if AmILHServer() then
    if (msgIn.pPartialAttrSet = null and
        msgIn.pPartialAttrSetEx = null) then
        fullReplicaRequest := true
    else
        fullReplicaRequest := false
    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, null, null, 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 responseSmtAddress ≠ null then
    Send the response using using the SMTP transport to
        responseSmtAddress
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_V10,
    pdwOutVersion: ADDRESS OF DWORD,
    var responseSmtAddress: unicodestring): ULONG

```

Informative summary of behavior: The TransformInput procedure transforms the received request message into a V10 request, which is a superset of the supported request messages.

```

partialAttrObjs: sequence of DSName
i: integer

responseSmtAddress := null

if requestVersion = 10

```

```

/* Windows Server 7 RPC request. */
nativeRequest := requestUnion.V10
if not DRS_EXT_GETCHGREPLY_V6 in ext.dwFlags then
    return ERROR_REVISION_MISMATCH
else
    pdwOutVersion^ := 6
endif

else if requestVersion = 8
/* Windows Server 2003 RPC request. */
nativeRequest := requestUnion.V8
nativeRequest.ulMoreFlags := 0
if not DRS_EXT_GETCHGREPLY_V6 in ext.dwFlags then
    return ERROR_REVISION_MISMATCH
else
    pdwOutVersion^ := 6
endif

else if requestVersion = 7 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
nativeRequest.ulMoreFlags := 0
if not DRS_EXT_GETCHGREPLY_V6 in ext.dwFlags then
    return ERROR_REVISION_MISMATCH
else
    pdwOutVersion^ := 6
endif

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)
nativeRequest.ulMoreFlags := 0
if ({DRS_WRIT_REP} ∩ requestUnion.V5.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
if ({DRS_WRIT_REP} ∩ requestUnion.V4.V3.ulFlags) = null then
    nativeRequest.pPartialAttrSet := GCPAS()
endif
nativeRequest.pPartialAttrSetEx := null

```

```

    nativeRequest.ulMoreFlags := 0
    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,
    searchFilter: LDAPString,
    dirSyncFlags: ULONG,
    msgIn: DRS_MSG_GETCHGREQ_V10,
    var msgOut: DRS_MSG_GETCHGREPLY_V6): ULONG

```

Informative summary of behavior: The GetReplChanges procedure processes an LDAP Search request with LDAP_SERVER_DIRSYNC_OID control or 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, searchFilter, dirSyncFlags), 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.

```

err: ULONG
ncRoot: DSName
pUtd: ADDRESS OF UPTODATE_VECTOR_V1_EXT
scope: set of DSName
attribute: ATTRTYP
partialAttrs: set of ATTRTYP
partialAttrsEx: 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_V1 /* See IDL_DRSUpdateRefs structures. */
tgt: DSName

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, "nTDSDSA Object" */

if NTDSDSA_OPT_DISABLE_OUTBOUND_REPL in DSAObj()!options and
    not DRS_SYNC_FORCED in msgIn.ulFlags and
    not dirSyncFlags 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, searchFilter)

/* Get object and link value changes in scope. */
GetChangesInScope(scope, pUtd, msgIn.ulExtendedOp, partialAttrs,
    partialAttrsEx, dirSyncFlags, 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
                err := AddObjToResponse(
                    hDrs, anc, ncRoot, msgIn.ulFlags, 0, clientDSA, msgOut)
                if err ≠ 0 then
                    return err
                endif
            endif
        endfor
    endif
    err := AddObjToResponse(
        hDrs, o, ncRoot, msgIn.ulFlags, 0, clientDSA, msgOut)
    if err ≠ 0 then
        return err
    endif
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
            err := AddObjToResponse(hDrs, anc, ncRoot,
                msgIn.ulFlags, 0, clientDSA, msgOut)
            if err ≠ 0 then
                return err
            endif
        endif
    endif
endif

if DRS_GET_TGT in msgIn.ulMoreFlags then
    /* Target predicate: insert any changes to the target object
     * before any changes to the link value. */
    tgt := GetDSNameFromAttrVal(v.att, v.val)
    if DRS_GET_ANC in msgIn.ulFlags then
        /* Ancestors predicate: insert any changes to the ancestors of
         * the target before any changes to the target. */
        foreach n in Ancestors of tgt, most distant ancestor first
            anc := select one a from changedObjs where a.obj = n
            if anc ≠ null then
                err := AddObjToResponse(hDrs, anc, ncRoot,
                    msgIn.ulFlags, 0, clientDSA, msgOut)
                if err ≠ 0 then
                    return err
                endif
            endif
        endfor
    endif
    err := AddObjToResponse(hDrs, tgt, ncRoot,

```

```

        msgIn.ulFlags, 0, clientDSA, msgOut)
    if err ≠ 0 then
        return err
    endif
    AddLinkToResponse(v, msgIn, msgOut)
endfor

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 and msgIn.uuidDsaObjDest ≠ NULLGUID) then
    /* Client has requested the server to add a repsTo entry. */
    updRefs.uuidDsaDes := msgIn.uuidDsaObjDest
    updRefs.pNC := msgIn.pNC^
    updRefs.pszDsaDest := NetworkAddress of DC corresponding to
        msgIn.uuidDsaObjDest
    updRefs.ulOptions := {DRS_ADD_REF, DRS_ASYNC_OP,
        DRS_GETCHG_CHECK} +
        { msgIn.ulFlags ∩ {DRS_WRIT_REP, DRS_REF_GCSPN}}

    /* Using updRefs, perform repsTo add to the specified NC replica,
     * the result value is a Windows error code or 0.
    err := UpdateRefs(updRefs^.V1)
    if(err ≠ 0) then
        return err
    endif
endif

return 0

```

4.1.10.5.3 GetReplScope

```

procedure GetReplScope(
    msgIn: DRS_MSG_GETCHGREQ_V10,
    searchFilter: LDAPString): set of DSName

```

Informative summary of behavior: The GetReplScope procedure returns the set of objects considered for normal replication or for an LDAP Search request with LDAP_SERVER_DIRSYNC_OID control: the objects in the requested NC replica (msgIn.pNC^) or a subset thereof, as indicated by the request flags (msgIn.ulFlags) and the search filter (searchFilter). 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
    if (ObjectMatchesSearchFilter(ncRoot, searchFilter) = true) then
        scope := {ncRoot}
    endif
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
                if (ObjectMatchesSearchFilter(anc, searchFilter) = true) then
                    scope := scope + {anc}
                endif
            endif
        endfor
    endif
else
    scope := select all o from subtree-ts-included ncRoot where true
    foreach o in ncRoot!subRefs
        if (ObjectMatchesSearchFilter(o, searchFilter) = true) then
            scope := scope + {o}
        endif
    endfor
endif

return scope

```

4.1.10.5.4 ObjectMatchesSearchFilter

```

procedure ObjectMatchesSearchFilter (
    o: DSNAME,
    searchFilter: LDAPString) : boolean

```

This procedure returns true if the search filter (*searchFilter*) is null. If the search filter is not null, it returns true if the object whose DSNAME is "o" matches the search filter; otherwise it returns false. See [\[MS-ADTS\]](#) for search filter processing, specifically section [3.1.1.3.1.3.1](#), Search Filters.

4.1.10.5.5 GetChangesInScope

```

procedure GetChangesInScope(
    scope: set of DSName,
    pUtd: ADDRESS OF UPTODATE_VECTOR_V1_EXT,
    ulExtendedOp: DWORD,
    partialAttrs: set of ATTRTYP,
    partialAttrsEx: set of ATTRTYP,
    dirSyncFlags: ULONG,
    var changedObjs: set of ObjAtts,
    var changedLinks: set of ObjAttVal)

```

Informative summary of behavior: The GetChangesInScope procedure 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 or as a result of processing LDAP Search request with LDAP_SERVER_DIRSYNC_OID control, as determined by the up-to-date vector (pUtd), the extended operation (ulExtendedOp), flags (dirSyncFlags) associated with the LDAP_SERVER_DIRSYNC_OID control, and the partial replica attribute filters (partialAttrs and partialAttrsEx).

```

o: DSName
a: ATTRTYP
attrsFound: set of ATTRTYP
attrsReq: set of ATTRTYP
stamp: AttributeStamp
cursor: UPTODATE_CURSOR_V2

/* 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
  attrsFound := {}
  attrsReq := {}
  foreach a of o's object class
    stamp := AttrStamp(o, a)
    if stamp ≠ null and
      ((ulExtendedOp = EXOP_REPL_SECRETS and IsSecretAttribute(a))
      or not FilterAttribute(o, a, stamp, pUtd,

          partialAttrs, partialAttrsEx, dirSyncFlags)) then
      attrsFound := attrsFound + {a}
    endif
    if(a = instanceType or a = proxiedObjectName) then
      attrsReq := attrsReq + {a}
    endif
  endfor
  if attrsFound ≠ {} then
    changedObjs := changedObjs + [obj: o, atts: attrsFound + attrsReq]
  else if (IT_NC_HEAD in o!instanceType and pUtd ≠ null)
    stamp := AttrStamp(o, usNChanged)
    cursor := select one c from pUtd^.rgCursors where c.uuidDsa =
      stamp.uuidOriginating
    if cursor = null or cursor.usnHighPropUpdate < stamp.usnOriginating
    then
      changedObjs := changedObjs + [obj: o, atts: attrsFound + attrsReq]
    endif
  endif
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 or dc.fLinkValueStampEnabled = true) 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, dirSyncFlags) then
            changedLinks := changedLinks + [obj: o, att: a, val: v]
        endif
    endfor
endfor
endif
endif
endif

```

4.1.10.5.6 FilterAttribute

```

procedure FilterAttribute(
    o: DSName,
    attribute: ATTRTYP,
    s: AttributeStamp,
    pUtd: ADDRESS OF UPTODATE_VECTOR_V1_EXT,
    partialAttrs: set of ATTRTYP,
    partialAttrsEx: set of ATTRTYP,
    dirSyncFlags: ULONG): boolean

```

Informative summary of behavior: The FilterAttribute procedure 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 stamps, 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 (that is, [cn](#) for **objects of class** [container](#), as shown below) for the object class of o, the update is filtered out.
- If LDAP_DIRSYNC_OBJECT_SECURITY is in dirSyncFlags, and the client does not have access rights to read the object, all the updates are filtered out except updates to the [isDeleted](#) and [isRecycled](#) attributes.

```

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
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

if not filtered then
    /* Filter non replicated attributes of o. */
    if AttrIsNonReplicated(attribute) then
        filtered := true
    endif
endif

if not filtered then
    /* If LDAP_DIRSYNC_OBJECT_SECURITY in dirSyncFlags, and the client does
    not have access rights to read the object, all the updates are filtered
    out except updates to isDeleted and isRecycled attributes. */

    if LDAP_DIRSYNC_OBJECT_SECURITY in dirSyncFlags and
        (AccessCheckObject(o, RIGHT_DS_LIST_OBJECT) = false or
        AccessCheckObject(o.parent, RIGHT_DS_LIST_CONTENTS) = false) and
        attribute # isDeleted and
        attribute # isRecycled then
        filtered := true
    endif
endif

return filtered

```

4.1.10.5.7 GetResponseSubset

```

procedure GetResponseSubset(
    msgIn: DRS_MSG_GETCHGREQ_V10,

```

```

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 procedure selects subsets of the changed objects and link values to include in this response. It 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 procedure then sets msgOut.usnvecTo to a new cookie that the client presents in its next request as msgIn.usnvecFrom.

The server SHOULD [<28>](#) 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.

4.1.10.5.8 AddObjToResponse

```

procedure AddObjToResponse(
  hDrs: DRS_HANDLE,
  o: ObjAtts,
  ncRoot: DSName,
  ulFlags: set of integer,
  ulExtendedOp: DWORD,
  clientDSA: DSName,
  var msgOut: DRS_MSG_GETCHGREPLY_V6) : ULONG

```

Informative summary of behavior: The AddObjToResponse procedure constructs a [REPLENTINFLIST](#) structure for a changed object and appends it to the response.

```

err: ULONG
re: REPLENTINFLIST
pAttr: ADDRESS OF ATTR
attribute: ATTRTYP
attrObj: DSName
attrVals: sequence of attribute values
i: DWORD
j: DWORD

err := 0

```

```

/* Construct a REPLENTINFLIST to represent the changes. */
re := all zeros
re.fIsNcPrefix := (o.obj = ncRoot)
if name in o.attrs and not re.fIsNcPrefix then
    re.pParentGuid := ADR(o.obj!parent)
endif
re.EntInf.pName := ADR(o.obj)
re.EntInf.AttrBlock.pAttrs := array of ATTR of size o.attrs.length
re.EntInf.AttrBlock.attrCount := o.attrs.length
re.pMetaDataExt := PROPERTY_META_DATA_EXT_VECTOR
    with rgMetaData of size o.attrs.length
re.pMetaDataExt^.cNumProps := o.attrs.length
for i := 0 to o.attrs.length - 1
    attribute := o.attrs[i]
    attrObj := SchemaObj(attribute)
    re.pMetaDataExt^.rgMetaData[i] = AttrStamp(o.obj, attribute)
    pAttr := ADR(re.EntInf.AttrBlock.pAttrs[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 if not AmILHServer() and /* W2K3 or lower */
        not DRS_WRIT_REP in ulFlags and /* partial replication */
        IsSecretAttribute(attribute) then
        /* secret attribute in W2K3 or lower servers, send a null value */
        pAttr^.AttrVal.pAVal = null
        re.pMetaDataExt^.rgMetaData[i].timeChanged = 0
    else
        /* not special processing */
        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,
             * "msDS-Behavior-Version: Forest Functional Level".
             */
            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
endfor
err := EncryptValuesIfNecessary(hDrs, pAttr^)

```



```

    if err ≠ 0 then
        return err
    endif
    /* if secrets are being sent 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

return err

```

4.1.10.5.9 UpdateRevealedList

```

procedure UpdateRevealedList(
    rodcdsa: DSName,
    revealedObject: DSName,
    attribute: ATTRTYP)

```

Informative summary of behavior: The UpdateRevealedList procedure adds or updates an entry for the attribute *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 the attribute value contains a [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 superseded 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

```

4.1.10.5.10 AddLinkToResponse

```

procedure AddLinkToResponse(
  v: ObjAttVal,
  msgIn: DRS_MSG_GETCHGREQ_V10,
  var msgOut: DRS_MSG_GETCHGREPLY_V6)

```

Informative summary of behavior: The AddLinkToResponse procedure constructs a [REPLVALINF](#) 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

```

4.1.10.5.11 EncryptValuesIfNecessary

```

procedure EncryptValuesIfNecessary(
    hDrs: DRS_HANDLE,
    var attr: ATTR) : ULONG

```

Informative summary of behavior: The EncryptValuesIfNecessary procedure encrypts the values of attributes that contain secret data. It 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 0
endif

if not DRS_EXT_STRONG_ENCRYPTION in ClientExtensions(hDrs).dwFlags then
    return SEC_E_ALGORITHM_MISMATCH
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, "Building and Using a

```

```

Security Context", and [MS-KILE] section 3.1.1.2, "Cryptographic
Material"

/* 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

return 0

```

4.1.10.5.12 ProcessFsmoRoleRequest

```

procedure ProcessFsmoRoleRequest(
    hDrs: DRS_HANDLE,
    msgIn: DRS_MSG_GETCHGREQ_V10,
    var msgOut: DRS_MSG_GETCHGREPLY_V6)

```

Informative summary of behavior: The ProcessFsmoRoleRequest procedure performs the requested FSMO role operation indicated by msgIn.ulExtendedOp.

```

fsmoObj: DSName
clientDsaObj: DSName
serverObj: DSName: DSName

```

```

rodcObj: 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

/* Specific error check when at DC functional level Win2K3 */
if (DSAObj()!msDS-Behavior-Version = DS_BEHAVIOR_WIN2003) and
    (not DRS_WRIT_REP in msgIn.ulFlags) then
    msgOut.ulExtendedRet := EXOP_ERR_PARAM_ERR
    return
endif

fsmoObj := msgIn.pNC^
if not ObjExists(fsmoObj) then
    msgOut.ulExtendedRet := EXOP_ERR_UPDATE_ERR
    return
endif
if msgIn.uuidDsaObjDest = null 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_FSMO_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. The server will refuse to take the FSMO role if
    it is not a full replica and cannot own FSMO. */
    if AmIRODC() then
        msgOut.ulExtendedRet := EXOP_ERR_FSMO_REFUSING_ROLES
    endif

    if fsmoObj!FSMORoleOwner # DSAObj() then
        ownerDsaObj := fsmoObj!FSMORoleOwner
        if not ObjExists(ownerDsaObj) then

```

```

        msgOut.ulExtendedRet := EXOP_ERR_UNKNOWN_CALLER
        return
    else if ownerDsaObj!isDeleted = true
        msgOut.ulExtendedRet := EXOP_ERR_OWNER_DELETED
        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
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_FSMO_NOT_OWNER
        return
    endif

    /* Locate or create the RID Set object for the client DC. */
    serverObj := clientDsaObj!parent
    clientComputerObj := serverObj!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" (RID 498)
 *    [MS-ADTS] section 7.1.1.6.14.
 * 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
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 */
pmsgOut.dwDRSError ERROR_DS_DRA_SOURCE_DISABLED
return
endif

/* Operation is invalid if destination is full replica but this server
 * is not, or if both are partial replicas but this server does not have
 * all the attributes needed by the destination in its PAS. */
if(not FullReplicaExists(GetObjectNC(pMsgIn.NC)) and
not pMsgIn.pPartialAttrSet = null)
msgOut.ulExtendedRet := EXOP_ERR_PARAM_ERR
return
else if not GetFilteredAttributeSet()∩ pMsgIn.pPartialAttrSet = {} then

```

```

        msgOut.ulExtendedRet := EXOP_ERR_PARAM_ERR
        return
    endif

    scope := {fsmObj}
else
    /* Unrecognized request. */
    msgOut.ulExtendedRet := EXOP_ERR_UNKNOWN_OP
    return
endif
endif
if scope ≠ {} then
    /* Add updates in scope to the response. */
    GetChangesInScope(scope, msgIn.pUpToDateVecDest, msgIn.ulExtendedOp,
        msgIn.pPartialAttrSet, msgIn.pPartialAttrSet,
        0, 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

```

4.1.10.5.13 RevealSecretsPolicy

```

typedef enum
{
    RevealSecretsDeny = 0,
    RevealSecretsAllow = 1,
    RevealSecretsNoPolicy = 2
} RevealSecretsPolicy;

```

4.1.10.5.14 GetRevealSecretsPolicyForUser

```

procedure GetRevealSecretsPolicyForUser(
    rodObj: DSName, userObj: DSName): RevealSecretsPolicy

```

Informative summary of behavior: The GetRevealSecretsPolicyForUser procedure 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
    krbtgt = 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-RevealOnDemand1Group
    */
    foreach revealObj in rodObj!msDS-RevealOnDemand1Group
        if IsUserIncluded(
            userObj!objectSid, revealObj!objectSid) then
            return RevealSecretsAllow
        endif
    endfor
    return RevealSecretsNoPolicy

```

4.1.10.5.15 RevealSecretsForUserAllowed

```

procedure RevealSecretsForUserAllowed(
    rodObj: DSName, userObj: DSName): boolean

```

Informative summary of behavior: The RevealSecretsForUserAllowed procedure 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

```

4.1.10.5.16 GetRoleScope

```

procedure GetRoleScope(fsObj: DSName): set of DSName

```

Informative summary of behavior: The GetRoleScope procedure returns the set of objects in the FSMO role identified by the FSMO role object fsObj.

```

scope: set of DSName
partitionsFsObj: DSName
schemaFsObj: DSName
ridFsObj: DSName
pdcFsObj: DSName
c: DSName
r: set of DSName

partitionsFsObj := select one o from children ConfigNC()
    where o!name = "Partitions"
schemaFsObj := SchemaNC()
infrastructureFsObj := select one o from children DefaultNC()
    where o!name = "Infrastructure"
ridFsObj := DefaultNC()!rIDManagerReference

/* Scope always includes fsObj. For the PDC Emulation Role, scope
 * includes only fsObj. */
scope := {fsObj}

if fsObj = partitionsFsObj then
    /* Partition Naming Master Role: Add to scope the children of the
     * Partitions container. */
    r := select all o from children partitionsFsObj where true
    scope := scope + r
else if fsObj = schemaFsObj then
    /* Schema Master Role: Set scope to all objects in the Schema
     * NC. */
    scope := select all o from subtree SchemaNC() where true
else if fsObj = infrastructureFsObj 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

```

4.1.10.5.17 SortResponseLinks

```

procedure SortResponseLinks(var msgOut: DRS_MSG_GETCHGREPLY_V6)

```

The SortResponseLinks procedure 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
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

```

4.1.10.5.18 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 procedure transforms the V6 reply (a superset of all supported reply messages) into the reply version supported by the client, optionally compressing it. The compression algorithms used for the DRS_COMP_ALG_WIN2K3 algorithm type are specified in section [4.1.10.5.19](#). The compression algorithm used for the DRS_COMP_ALG_MSZIP algorithm type is specified in [\[RFC1951\]](#).

```

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, "Type Serialization

```

```

        Version 1" and "Type Serialization Version 2"

/* 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. */
if (compressionAlg = DRS_COMP_ALG_MSZIP) then
    compressed :=
        Compress pickled in accordance
        with [RFC1951].
else
    CompressOrDecompressWin2k3(pickled, compressed, pickled.length, TRUE)
endif
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, "Type Serialization
            Version 1" and "Type Serialization Version 2"

        /* 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. */
        if (compressionAlg = DRS_COMP_ALG_MSZIP) then
            compressed :=
                Compress pickled in accordance

```

```

        with [RFC1951].
    else
        CompressOrDecompressWin2k3(pickled, compressed, pickled.length, TRUE)
    endif
    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

```

4.1.10.5.19 CompressOrDecompressWin2k3

```

procedure CompressOrDecompressWin2k3(
    inputBuffer: sequence of BYTE,
    inputSize: DWORD
    ref outputBuffer: sequence of BYTE,
    comp: BOOLEAN)

```

Informative summary of behavior: The CompressOrDecompressWin2k3 procedure compresses or decompresses data using the compression algorithm LZ77 (section [4.1.10.5.19.1](#)) and the basic encoding algorithm DIRECT2 (section [4.1.10.5.19.2](#)). The procedure has the following parameters:

- **inputBuffer:** A sequence of [BYTE](#) containing data to compress or decompress.
- **inputSize:** The [DWORD](#) value that indicates the size of inputBuffer in bytes.
- **outputBuffer:** A sequence of BYTE that is an empty buffer. Compressed or decompressed data is filled into this buffer.
- **comp:** A Boolean indicating whether to compress (comp=TRUE) or decompress (comp=FALSE) the inputBuffer.

4.1.10.5.19.1 LZ77 Compression Algorithm

The LZ77 compression algorithm is used to analyze input data and determine how to reduce the size of that input data by replacing redundant information with metadata. Sections of the data that are identical to sections of the data that have been encoded are replaced by a small amount of metadata that indicates how to expand those sections again. The encoding algorithm is used to take that combination of data and metadata and serialize it into a stream of bytes that can later be decoded and decompressed.

Compression Algorithm Terminology

The following terms are associated with the compression algorithm. Some of the terms also apply to the DIRECT2 encoding algorithm defined in the next section.

input stream: The sequence of bytes to be compressed.

byte: The basic data element in the input stream.

coding position: The position of the byte in the input stream that is currently being coded (the beginning of the lookahead buffer).

lookahead buffer: The byte sequence from the coding position to the end of the input stream.

window: A buffer that indicates the number of bytes from the coding position backward. A window of size W contains the last W processed bytes.

pointer: Information about the beginning of the match in the window (referred to as "B" in the example later in this section) and its length (referred to as "L" in the example later in this section).

match: The string that is used to find a match of the byte sequence between the lookahead buffer and the window.

Using the Compression Algorithm

To use the LZ77 compression algorithm:

1. Set the coding position to the beginning of the input stream.
2. Find the longest match in the window for the lookahead buffer.
3. Output the P,C pair, where P is the pointer to the match in the window, and C is the first byte in the lookahead buffer that does not match.
4. If the lookahead buffer is not empty, move the coding position (and the window) L+1 bytes forward.
5. Return to step 2.

Compression Process

The compression algorithm searches the window for the longest match with the beginning of the lookahead buffer and then outputs a pointer to that match. Because even a 1-byte match might not be found, the output cannot contain only pointers. The compression algorithm solves this problem by outputting after the pointer the first byte in the lookahead buffer after the match. If no match is found, the algorithm outputs a null-pointer and the byte at the coding position.

Compression Process Example

The following table shows the input stream that is used for this compression example. The bytes in the input, "AABCBBABC", occupy the first nine positions of the stream.

Input stream

Position	1	2	3	4	5	6	7	8	9
Byte	A	A	B	C	B	B	A	B	C

The following table shows the output from the compression process. The table includes the following columns:

Step: Indicates the number of the encoding step. A step in the table finishes every time that the encoding algorithm makes an output. With the compression algorithm, this process happens in each pass through step 3.

Position: Indicates the coding position. The first byte in the input stream has the coding position 1.

Match: Shows the longest match found in the window.

Byte: Shows the first byte in the lookahead buffer after the match.

Output: Presents the output in the format (B,L)C, where (B,L) is the pointer (P) to the match. This gives the following instructions to the decoder: Go back B bytes in the window and copy L bytes to the output. C is the explicit byte.

Note One or more pointers might be included before the explicit byte that is shown in the Byte column. That is, a metadata pointer does not always need to be followed by an explicit byte. An input stream of "ABCABCABC", for example, can be represented as "(0,0)A(0,0)B(0,0)C(3,3)(6,3)" using the (B,L)C notation, with the last two elements being pointers without explicit bytes. The compressed output can be any combination of pointers and explicit bytes.

Compression process output

Step	Position	Match	Byte	Output
1.	1	--	A	(0,0)A
2.	2	A	B	(1,1)B
3.	4	--	C	(0,0)C
4.	5	B	B	(2,1)B
5.	7	A B	C	(5,2)C

The result of compression, conceptually, is the output column—that is, a series of bytes and optional metadata that indicates whether that byte is preceded by some sequence of bytes that is already in the output.

Because representing the metadata itself requires bytes in the output stream, it is inefficient to represent a single byte that has previously been encoded by two bytes of metadata (offset and length). The overhead of the metadata bytes equals or exceeds the cost of outputting the bytes directly. Therefore, the protocol considers sequences of bytes to be a match only if the sequences have three or more bytes in common.

4.1.10.5.19.2 DIRECT2 Encoding Algorithm

The basic notion of the DIRECT2 encoding algorithm is that data appears unchanged in the compressed representation, and metadata is encoded in the same output stream, and in line with, the data.

The key to decoding the compressed data is recognizing what bytes are metadata and what bytes are data. The decoder **MUST** be able to identify the presence of metadata in the compressed and encoded data stream. Bitmasks are inserted periodically in the byte stream to provide this information to the decoder.

This section describes the bitmasks that enable the decoder to distinguish data from metadata. It also describes the process of encoding the metadata.

Bitmask

To distinguish data from metadata in the compressed byte stream, the data stream begins with a 4-byte bitmask that indicates to the decoder whether the next byte to be processed is data (a "0" value in the bit), or if the next byte (or series of bytes) is metadata (a "1" value in the bit). If a "0" bit is encountered, the next byte in the input stream is the next byte in the output stream. If a "1" bit is encountered, the next byte or series of bytes is metadata that **MUST** be interpreted further.

For example, a bitmask of 0x01000000 indicates that the first seven bytes are actual data, followed by encoded metadata that starts at the eighth byte. The metadata is followed by 24 additional bytes of data. A bitmask of 0x112000000 indicates that there will be metadata in the 4th, 8th, and 11th elements (note that the actual byte positions in the compressed data may be different because metadata elements will range from 2 to 6 bytes in length), with the remaining elements being data bytes.

When the bitmask has been consumed, the next four bytes in the input stream are another bitmask.

The bitmask must also contain a "1" in the bit following the last encoded element, to indicate the end of the compressed data. For example, given a hypothetical 8-bit bitmask, the string "ABCABCDEF" should be compressed as (0,0)A(0,0)B(0,0)C(3,3)D(0,0)E(0,0)F. Its bitmask would be b'00010001' (0x11). This would indicate three bytes of data, followed by metadata, followed by an additional 3 bytes, finally terminated with a "1" to indicate the end of the stream.

The final end bit is always necessary, even if an additional bitmask has to be allocated. If the string in the above example was "ABCABCDEF", for example, it would require an additional bitmask. It would begin with the bitmask b'00010000', followed by the compressed data, and followed by another bitmask with a "1" as the next bit to indicate the end of the stream.

Encoding Metadata

In the output stream, actual data bytes are stored unchanged. Bitmasks are stored periodically to indicate whether the next byte or bytes are data or metadata. If the next bit in the bitmask is a "1", the next set of bytes in the input data stream is metadata (unless the last element of data was read, in which case the "1" bit would indicate the end of the stream as noted above). This metadata contains an offset back to the start of the data to be copied to the output stream, and the length of the data to be copied.

To represent the metadata as efficiently as possible, the encoding of that metadata is not fixed in length. The encoding algorithm supports the largest possible floating compression window to increase the probability of finding a large match; the larger the window, the greater the number of bytes that are needed for the offset. The encoding algorithm also supports the longest possible match; the longer the match length, the greater the number of bytes that are needed to encode the length.

Metadata Offset

The protocol assumes the metadata is two bytes in length. The three low-order bits are used to encode the length. The high-order 13 bits are a first complement of the offset, which is represented as a negative signed value in 2's complement. The offset is only encoded with those 13 bits. This value cannot be extended and defines the maximum size of the compression floating window. For example, the metadata 0x0018 is converted into the offset b'000000000011', and the length b'000'. The offset is '-4', computed by inverting the offset bits, treating the result as a 2's complement, and converting it to an integer.

Match Length

Unlike the metadata offset, the match length is extensible. If the length is less than 10 bytes, it is encoded in the three low-order bits of the 2-byte metadata. Although three bits seems to allow for a maximum length of six (the value b'111' is reserved), because the minimum match is three bytes, these three bits actually allow for the expression of lengths from three to nine. The match length goes from $L = b'000' + 3$ bytes, to $L = b'110' + 3$ bytes. Because smaller lengths are much more common than the larger lengths, the algorithm tries to optimize for smaller lengths. To encode a length between three and nine, we use the three bits that are "in-line" in the 2-byte metadata.

If the length of the match is greater than nine bytes, an initial bit pattern of b'111' is put in the three bits. This does not signify a length of 10 bytes, but instead a length that is greater than or equal to 10, which is included in the low-order nibble of the following byte.

Every other time that the length is greater than nine, an additional byte follows the initial 2-byte metadata. The first time that the additional byte is included, the low-order nibble is used as the additive length. The high-order nibble is "reserved" for the next metadata instance when the length is greater than nine. Therefore, the first time that the decoder encounters a length that is greater than nine, it reads the next byte from the data stream and the low-order nibble is extracted and used to compute the length for this metadata instance. The high-order nibble is remembered and used the next time that the decoder encounters a metadata length that is greater than nine. The third time that a length that is greater than nine is encountered, another extra byte is added after the 2-byte metadata, with the low-order nibble used for this length and the high-order nibble reserved for the fourth length that is greater than nine, and so on.

If the nibble from this "shared" byte is all "1s" (for example, b'1111'), another byte is added after the shared byte to hold more length. In this manner, a length of 24 is encoded as follows:

- b'111' (in the three bits in the original two bytes of metadata), plus
- b'1110' (in the nibble of the "shared" byte" of extended length)
- b'111' means 10 bytes plus b'1110', which is 14, which results in a total of 24.

If the length is more than 24, the next byte is also used in the length calculation. In this manner, a length of 25 is encoded as follows:

- b'111' (in the three bits in the original two bytes of metadata), plus
- b'1111' (in the nibble of the "shared" byte of extended length), plus
- b'00000000' (in the next byte).

This scheme is good for lengths of up to 278 (a length of 10 in the three bits in the original two bytes of metadata, plus a length of 15 in the nibble of the "shared" byte of extended length, plus a length of up to 254 in the extra byte).

A "full" (all b'1') bit pattern (b'111', b'1111', and b'11111111') means that there is more length in the following two bytes.

The final two bytes of length differ from the length information that comes earlier in the metadata. For lengths that are equal to 280 or greater, the length is calculated only from these last two bytes, and is not added to the previous length bits. The value in the last two bytes, a 16-bit integer, is three less than the metadata length. These last two bytes allow for a match length of up to 32,768 bytes + 3 bytes (the minimum match length).

The following table summarizes the length representation in metadata.

Note Length is computed from the bits that are included in the metadata plus the minimum match length of three.

Length representation in metadata

Match length	Length bits in the metadata
24	b'111' (three bits in the original two bytes of metadata)

Match length	Length bits in the metadata
	+ b'1110' (in the high-order or lower-order nibble, as appropriate, of the shared byte)
25	b'111' (three bits in the original two bytes of metadata) + b'1111' (in the high-order or lower-order nibble, as appropriate, of the shared byte) + b'00000000' (in the next byte)
26	b'111' (three bits in the original two bytes of metadata) + b'1111' (in the high-order or lower-order nibble, as appropriate, of the shared byte) + b'00000001' (in the next byte)
279	b'111' (three bits in the original two bytes of metadata) + b'1111' (in the high-order or lower-order nibble, as appropriate, of the shared byte) + b'11111110' (in the next byte)
280	b'111' (three bits in the original two bytes of metadata) b'1111' (in the high-order or lower-order nibble, as appropriate, of the shared byte) b'11111111' (in the next byte) 0x0115 (in the next two bytes). These two bytes represent a length of 277 + 3 (minimum match length). Note All of the length is included in the final two bytes and is not additive, as were the previous length calculations for lengths that are smaller than 280 bytes.
281	b'111' (three bits in the original two bytes of metadata) b'1111' (in the high-order or lower-order nibble, as appropriate, of the shared byte) b'11111111' (in the next byte) 0x0116 (in the next two bytes). This is 278 + 3 (minimum match length). Note All of the length is included in the final two bytes and is not additive, as were the previous length calculations for lengths that are smaller than 280 bytes.

A "full" bit pattern in that last half word does not mean that more metadata is coming after the last bytes.

The LZ77 compression algorithm produces a well-compressed encoding for small valued lengths, but as the length increases, the encoding becomes less well compressed. A match length of greater than 278 bytes requires a relatively large number of bits: 3+4+8+16. This includes three bits in the original two bytes of metadata, four bits in the nibble in the "shared" byte, eight bits in the next byte, and 16 bits in the final two bytes of metadata.

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(  
    hDrs: DRS_HANDLE,  
    rf: RepsFrom,  
    msgIn: DRS_MSG_GETCHGREQ_V10,  
    dwOutVersion: ULONG,  
    msgOut: DRS_MSG_GETCHGREPLY) : ULONG
```

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:

- hDrs: The [DRS_HANDLE](#) derived by sending [IDL_DRSBind](#) to the server.
- rf: [RepsFrom](#) for the server.
- msgIn: **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 and
    msgReplyV6.pNC^ ≠ SchemaNC()
then
    return ERROR_DS_DRA_SCHEMA_MISMATCH
endif
Remove sourcePrefixTable[lastElement] from sourcePrefixTable

else
    return ulResult
endif

/* If the source has the Recycle Bin optional feature enabled, then it must
 * be enabled locally, unless the Schema partition is being updating.
 */
if msgReplyV6.pNC^ ≠ SchemaNC() and
    ServerExtensions(hDrs).RB and not IsRecycleBinEnabled()
then
    EnableRecycleBin()
    return ERROR_DS_DRS_EXTENSIONS_CHANGED
endif

/* Process object updates. */
replEntinfList := msgReplyV6.pObjects^
while (ulResult = 0) and (not replEntinfList = null)
    /* Decrypt any encrypted attribute values. */
    ulResult := DecryptValuesIfNecessary (
        hDrs,
        sourcePrefixTable,
        replEntinfList.Entinf.AttrBlock)

    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 msgIn.ulFlags) then
            writableReplica := true
        else
            writableReplica := false
        endif
        continueProcessing := AdjustInstanceTypeAttrVal(
            msgReplyV6.pNC^,
            writableReplica,
            replEntinfList,
            prefixTable)
    endif
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

/* Enable link value updates for outbound replication
 * if inbound link value updates are detected from source. */
if (msgReplyV6.cNumValues > 0) then
  dc.fLinkValueStampEnabled = true
endif

/* Process link value updates. */
linkValueCount := 0
while (ulResult = 0) and (linkValueCount < msgReplyV6.cNumValues)
  ulResult := ProcessLinkValue(
    msgReplyV6.rgValues[linkValueCount],
    msgReplyV6.pNC^,
    prefixTable,
    msgIn.ulFlags,
    msgIn.ulMoreFlags)
  linkValueCount := linkValueCount + 1
endwhile

if (ulResult = ERROR_DS_DRA_MISSING_PARENT) then
  Send IDL_DRSGetNCChanges message again with the same input
  parameters specified in msgIn but this time with msgIn.ulFlags
  containing DRS_GET_ANC field set. It is an error for this
  condition to occur if (DRS_GET_ANC in msgIn.ulFlags) is true
else if (ulResult = ERROR_DS_DRA_RECYCLED_TARGET) then
  Send IDL_DRSGetNCChanges message again with the same input
  parameters specified in the msgIn but this time with msgIn.ulMoreFlags
  containing DRS_GET_TGT field set.
else if (msgIn.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,
      msgIn.partialAttrSetEx^)
  endif
endif
endif

```

```
return ulResult
```

4.1.10.6.2 EnableRecycleBin

```
procedure EnableRecycleBin()
```

Informative summary of behavior: The EnableRecycleBin procedure is invoked during inbound replication if the source has the **Recycle Bin optional feature** enabled but the destination does not. It adds a reference to the object representing the Recycle Bin optional feature to the [msDS-EnabledFeature](#) attribute of both the [nTDSDSA](#) object of the destination DC and the Cross-Ref-Container container. For more details, see [\[MS-ADTS\]](#) sections [3.1.1.9](#), [3.1.1.9.1](#), and [7.1.1.2.1](#).

```
Asynchronous Processing: Initiate a logical thread of control
to process the remainder of this request asynchronously
return 0
```

```
DSNAME rbObj := select one v from ConfigNC()
                where v!objectGuid = 766ddcd8-acd0-445e-f3b9-a7f9b6744f2a
ConfigNC()!msDS-EnabledFeature :=
    ConfigNC()!msDS-EnabledFeature + {rbObj}
DSAObj()!msDS-EnabledFeature :=
    DSAObj()!msDS-EnabledFeature + {rbObj}
return
```

4.1.10.6.3 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, 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

```



```

endif
return true /* Continue processing the replicated update. */

```

4.1.10.6.4 AdjustInstanceTypeAttrVal

```

procedure AdjustInstanceTypeAttrVal(
    ncReplicated: DSName,
    writableReplica: DSName,
    var replEntinfList: REPLENTINFLIST,
    prefixTable: PrefixTable) : boolean

```

Informative summary of behavior: The 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, then this object is
 * a subordinate reference. Take this opportunity
 * to ensure that ncReplicated!subRefs has an entry for this
 * sub-ref object.
 */
ncSubRef := replEntinfList.Entinf.pName^
if (not ncSubRef in ncReplicated!subRefs) then
    ncReplicated!subRefs := ncReplicated!subRefs + {ncSubRef}
endif
if ObjExists(ncSubRef)
    /* Ensure that all sub-ref objects have the flag IT_NC_ABOVE set. */
    if not IT_NC_ABOVE in ncSubRef!instanceType then
        ncSubRef!instanceType := ncSubRef!instanceType + {IT_NC_ABOVE}
    endif

    /* If the sub-ref object corresponds to a locally instantiated
     * child NC, then skip this update as the sub-ref object
     * will be updated when the child NC replicates in.

```

```

        */
        if PartialGCReplicaExists(ncSubRef) then
            return false /* Skip processing this entry. */
        endif
    endif

    /* If sub-ref object does not exist or exists but the child NC is not
     * locally instantiated, then continue processing this entry to
     * add or update the sub-ref object.
     */
    instanceTypeAdjustedVal := instanceTypeVal + {IT_NC_ABOVE,
        IT_UNINSTANT, IT_NC_HEAD}

else
    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
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

```

4.1.10.6.5 SetResetInstanceTypeBits

```

procedure SetResetInstanceTypeBits(y: DWORD): DWORD

```

The SetResetInstanceTypeBits procedure is an implementation-specific function that MAY [<29>](#) set or reset bits in y other than IT_WRITE and IT_NC_HEAD. It returns the updated value.

4.1.10.6.6 PerformModifyOperation

```

procedure PerformModifyOperation(
    data: ENTINF,
    updateObject: DSNAME,
    prefixTable: PrefixTable): integer

```

The PerformModifyOperation procedure performs a modify operation with the given [ENTINF](#) to modify updateObject, 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 updateObject. If the DN in data.pName.StringName is not equal to the DN of updateObject in updateObject.StringName, then let newParentName be the DSNAME of the parent object identified in data.pName.StringName, let newRDN be the RDN identified in data.pName.StringName, and call PerformModifyDNOOperation(updateObject!distinguishedName, newParentName, newRDN). The PerformModifyDNOOperation procedure call should NOT be performed as an originating update.

For each [ATTR attr](#) in data.AttrBlock, let *attribute* be the [ATTRTYP](#) returned by [LocalAttrFromRemoteAttrId](#)(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].

The PerformModifyOperation procedure is NOT to be performed as an originating update. The AttributeAndStamp values associated with the modified attributes should not be touched by this procedure. For more information about originating updates, see [\[MS-ADTS\]](#) section 3.1.1.1.9.

If the modify operation succeeds, the procedure returns 0. If the modify operation fails, the procedure returns a Windows error code.

4.1.10.6.7 NameObject

```
procedure NameObject(  
    replEntinfList: REPLENTINFLIST,  
    sourcePrefixTable: PrefixTable,  
    nc: DSName,  
    attributesAndStamps: set of AttributeAndStamp): DWORD
```

Informative summary of behavior: The NameObject procedure performs the necessary steps to identify an unused name for an object, whether it is being added new or renamed. 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 method returns a Windows error code if it encounters an error while updating the object.

```
parentObject: DSNAME  
newParentObject: 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. */
    newParentObject := 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 the value of newObjectDN to replEntinfList.Entinf.pName^.StringName
    and update the value in replEntinfList.Entinf.pName^.structLen and
    replEntinfList.Entinf.pName^.NameLen accordingly.

    return ERROR_SUCCESS;

```

4.1.10.6.8 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.

```

newObject: DSName
dwResult: DWORD
objectClassAttr: ATTRVAL
isDeletedAttr: ATTRVAL
isDeletedValue: boolean
ncNameValue: DSName
ncNameAtt: ATTRVAL
partitionsContainer: DSName
parentObject: DSName

/* Find an appropriate and unused name for the object, updating
 * replEntInflList as appropriate */
dwResult := NameObject(replEntInflList, nc, sourcePrefixTable, attributeAndStamps)

if dwResult # ERROR_SUCCESS then
    return dwResult
endif

/* Check if this is a cross-ref in the partitions container replicating in.*/
objectClassAttr := ENTINF_GetValue(
    replEntInflList.Entinf,
    objectClass,
    sourcePrefixTable)
ncNameAtt := ENTINF_GetValue(
    replEntInflList.Entinf,
    ncName,
    sourcePrefixTable)
ncNameVal := ValueFromATTRVal(ncNameAtt, Syntax(ncNameAtt), sourcePrefixTable)

partitionsContainer:= DescendantObject(ConfigNC(), "CN=Partitions,")
parentObject := replEntInflList.Entinf.pName^ stripped of the first RDN.

if(crossRef in ObjectClassAttr and parentObject = partitionsContainer)
    dwResult := AddSubRef(ncNameVal)
    if dwResult # 0 then
        return dwResult
    endif

    isDeletedAttr := ENTINF_GetValue(
        replEntInflList.Entinf,
        isDeleted,
        sourcePrefixTable)

    if (isDeletedAttr = null) then
        isDeletedValue := false
    else
        isDeletedValue :=
            ValueFromATTRVal(isDeletedAttr, Syntax(isDeleted), sourcePrefixTable)
    endif

    If(isDeleted Value)
        DelSubRef(ncNameVal)
    endif
endif

dwResult := PerformAddOperation(replEntInflList.Entinf, newObject,
    sourcePrefixTable, FALSE)

/* Update attribute stamps. */

```

```

if (dwResult = 0) then
  for each e in attributesAndStamps do
    SetAttrStamp(newObject, e.attribute, e.stamp)
  endfor
endif

return dwResult

```

4.1.10.6.9 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 the 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
stampRemote: AttributeStamp
stampLocal: AttributeStamp
attribute: ATTRTYP
nameAttrAndStamp: AttributeAndStamp
attrAndStamp: AttributeAndStamp
isDeletedAttrAndStamp: AttributeAndStamp
dwResult: DWORD

updateObject := replEntinfList.Entinf.pName^

/* Determine if attributesAndStamps 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. */

/* The NameObject function will find an appropriate, unused, local
 * name for the object and modify the replEntInfList appropriately */

dwResult := NameObject(replEntInfList, sourcePrefixTable,
                      nc, attributeAndStamps)
if dwResult # ERROR_SUCCESS then
    return dwResult
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

dwResult := PerformModifyOperation(replEntInfList.Entinf,
                                   updateObject,
                                   sourcePrefixTable
                                   )

if dwResult # ERROR_SUCCESS then
    return dwResult
endif

/* Update attribute stamps on the object to those corresponding to
 * the replicated updates. */
for each e in attributesAndStamps do
    SetAttrStamp(updateObject, e.attribute, e.stamp)
endfor

if updateObject!isDeleted = true then
    if(crossRef in updateObject!objectClass)
        /* If this is a cross-ref being deleted, then the respective
         * sub-ref object, if any, must also be deleted.*/
        DelSubRef (updateObject!ncName)
    endif
    /* There might be attribute values left on this object that do not

```

```

    * conform to the invariants of a tombstone or deleted-object (see
    * MS-ADTS section 3.1.1.5.5). Delete the object again to create an
    * originating change of any such attribute values that need it.
    * This originating change will affect the metadata of updateObject,
    * and can explicitly affect metadata just written to the database
    * in the above SetAttrStamp procedure. */
    dwResult := RemoveObj(updateObject,false)
else
    isDeletedAttrAndStamp := select one e from attributesAndStamps where
        (e.attribute = isDeleted)
    if(isDeletedAttrAndStamp != null and crossRef in updateObject!objectClass)
        /* If this is a cross-ref being undeleted, then we must also undelete
        * the respective sub-ref object. */
        AddSubRef (updateObject!ncName)
    endif
endif

if updateObject!isRecycled = true and IsRecycleBinEnabled() then
    /* There might be attribute values left on this object that do
    * not conform to the invariants of a recycled-object (see MS-ADTS
    * section 3.1.1.5.5). Recycle the object again to create an originating
    * change of any such attribute values that need it. This
    * originating change will affect the metadata of updateObject, and
    * can explicitly affect metadata just written to the database in
    * the above SetAttrStamp procedure. */
    dwResult := RecycleObj(updateObject)
endif

return dwResult

```

4.1.10.6.10 FindBestParentObject

```

procedure FindBestParentObject(
    parentObject: DSName,
    replEntinflList: 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 of the desired parent object.
- replEntinflList: The replicated update that should be applied.
- sourcePrefixTable: The prefix table from the server.
- nc: The DSName 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

```

4.1.10.6.11 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: The update to be applied.

parentObject: The [DSName](#) 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 existing attribute stamp for name and add in a new one
 * so that the update is seen as an originating update. */
attributesAndStamps := attributesAndStamps - {nameAttrStamp}
nameAttrStamp := originating update stamp
/* See MS-ADTS section 3.1.1.1.9 */
attributeAndStamps := attributeAndStamps + {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

```

4.1.10.6.12 MakeConflictDN

```

procedure MakeConflictDN(oldDN: DN, guid: GUID): DN

```

The MakeConflictDN procedure is used during name conflict resolution. For more information, see section [4.1.10.6.11](#).

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.

4.1.10.6.13 ProcessLinkValue

```
procedure ProcessLinkValue(  
    replValinf: REPLVALINF,  
    nc: DSName,  
    sourcePrefixTable: PrefixTable,  
    ulFlags: ULONG,  
    ulMoreFlags: ULONG): 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: The link value replicated update.
- nc: The [DSName](#) of the root of the NC replica where the replicated update is applied.
- sourcePrefixTable: The prefix table from the server.
- ulFlags: A [DRS_OPTIONS](#) bit field.
- ulMoreFlags: A [DRS_MORE_GETCHGREQ_OPTIONS](#) bit field.

```
updateObject: DSName  
targetObject: DSName  
isDeleted: boolean  
attribute: ATTRTYP  
attributeValue: attribute value  
attributeValues: set of attribute value  
newAttributeValue: attribute value  
localValueStamp: LinkValueStamp  
remoteValueStamp: LinkValueStamp  
  
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

    if (IsRecycleBinEnabled()) then
        isRecycled := updateObject!isRecycled
        if (isRecycled = true) then
            if (DRS_GET_ANC in ulFlags) then
                /* Local object is recycled, and it is up-to-date.
                Replicated update is not applied on a recycled object */
                return 0
            else
                /* Local object is recycled, but it may not be up-to-date. */
                return ERROR_DS_DRA_MISSING_PARENT
            endif
        endif
    else
        isDeleted := updateObject!isDeleted
        if (isDeleted = true) then
            if (DRS_GET_ANC in ulFlags) then
                /* Local object is deleted, and it is up-to-date.
                Replicated update is not applied on a deleted object.*/
                return 0
            else
                /* Local object is deleted, but it may not be up-to-date. */
                return ERROR_DS_DRA_MISSING_PARENT
            endif
        endif
    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. */

        newAttributeValue = ValueFromATTRVAL(

```

```

        sourcePrefixTable, Syntax(attribute), replValInf.pAval)
targetObject = GetDSNameFromAttrVal( replValInf.attrTyp, replValInf.pAval)
if ((IsRecycleBinEnabled() and targetObject!isRecycled) or
    (not IsRecycleBinEnabled() and targetObject!isDeleted)) then
    if (DRS_GET_TGT in ulMoreFlags) then
        /* nothing to do */
        return 0
    else
        return ERROR_DS_DRA_RECYCLED_TARGET
    endif
endif
if (not attributeValue = null) then
    /* Remove the old attribute value. */
    RemoveAttrVal(updateObject, attribute, attributeValue)
endif

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
endif
SetLinkStamp(updateObject, attribute, newAttributeValue,
    remoteValueStamp)
endif

return 0

```

4.1.10.6.14 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 for the server.
- msgReplyV6: The [IDL DRSGetNCChanges](#) response from the server.
- dsaServer: The [DSName](#) of the [nTDSDSA](#) object of the server.
- ulResult: A Windows error code that indicates 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}

```

4.1.10.6.15 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](#) 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

```

4.1.10.6.16 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 132391](#)), 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](#) 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, "Building and Using a
Security Context", and [MS-KILE] section 3.1.1.2, "Cryptographic
Material"

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

```

4.1.10.6.17 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.

```

pInBuffer: sequence of BYTE
pOutBuffer: sequence of BYTE
cbInBufferCompress: DWORD
cbInBufferDeCompress: 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
        compressionAlg := DRS_COMP_ALG_MSZIP
    else if (dwOutVersion = 7) then
        pInBuffer := msgOut.V7.CompressedAny.pbCompressedData
        cbInBufferCompress := msgOut.V7.CompressedAny.cbCompressedSize
        cbInBufferDeCompress :=
            msgOut.V7.CompressedAny.cbUncompressedSize
        compressionAlg := msgOut.V7.CompressionAlg
    endif

    DecompressMessage(pInBuffer, cbInBufferCompress, cbOutBufferCompress,
        compressionAlg, pOutputBuffer)

    /* 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, "Type Serialization
        Version 1" and "Type Serialization Version 2"

    return GetNCChangesV6Reply(msgOut, dwOutVersion)

```

```
endif
```

4.1.10.6.18 DecompressMessage

```
procedure DecompressMessage(  
  pInBuffer: sequence of BYTE,  
  cbInBufferCompress: DWORD,  
  cbInBufferDecompress: DWORD,  
  DRS_COMP_ALG_TYPE: compressionAlg  
  ref pOutputBuffer: sequence of BYTE)
```

Informative summary of behavior: Compression subdivides a data stream into sequences of bytes called chunks. The DecompressMessage 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.

```
pInBlock: ADDRESS OF COMPRESSED_DATA  
cbInputProcessed: DWORD  
cbDecompressedData: DWORD  
  
if (cbInBufferCompress = cbInBufferDecompress) then  
  /* No decompression required here. */  
  pOutBuffer := pInBuffer  
  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 (compressionAlg = DRS_COMP_ALG_MSZIP) then  
        pDecompressedData :=  
          Decompress pInBlock^.data in accordance  
            with [RFC1951].  
      else  
        pDecompressedData := new sequence of BYTE of length  
          pInBlock^.cbDecompressedSize  
        CompressOrDecompressWin2k3(pInBlock^.data,  
          pInBlock^.cbDecompressedSize,  
          pDecompressedData, FALSE)  
    endif  
  endwhile
```

```

        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

```

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 DN's:
- 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

Where V1 represents the [REPS_FROM](#) structure. V1.mtx_address represents the [MTX_ADDR](#) structure stored in the data field of the REPS_FROM structure.

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 DN's: 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 = 10
- pmsgIn = [DRS_MSG_GETCHGREQ_V10](#)

- 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](#)
- ulMoreFlags: 0

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 DN's:
- 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 DN's:
- 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 = 10`
- `pmsgIn = DRS_MSG_GETCHGREQ_V10`
 - 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](#)
- ulMoreFlags: 0

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: [REPLVALINE](#)
 - 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: <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 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 the 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 DN's:
- 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 = 10
- pmsgIn = [DRS_MSG_GETCHGREQ_V10](#)
 - 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](#)
- ulMoreFlags: 0

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*

- ImPwdHistory: *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: 0x0 V1.cbOtherDraOffset: 216 V1.cbOtherDra: 60
V1.ulReplicaFlags: 0x70 V1.rtSchedule: <ldp: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 the 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: 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 or ERROR_MORE_DATA if successful; another Windows error code if a failure occurred.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE, ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.

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, which are presented in little-endian byte order:

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

CL (DRS_NT4_CHGLOG_GET_CHANGE_LOG, 0x00000001): If set, the server returns the PDC change log.

SN (DRS_NT4_CHGLOG_GET_SERIAL_NUMBERS, 0x00000002): 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 emulator FSMO 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

ValidatedDRSInput(hDrs, 11)

pdwOutVersion^ := 1
pmsgOut^.V1.cbRestart := 0
pmsgOut^.V1.cbLog := 0
pmsgOut^.V1.ReplicationState.SamSerialNumber := 0
```



```

pmsgOut^.V1.ReplicationState.SamCreationTime := 0
pmsgOut^.V1.ReplicationState.BuiltinSerialNumber := 0
pmsgOut^.V1.ReplicationState.BuiltinCreationTime := 0
pmsgOut^.V1.ReplicationState.LsaSerialNumber := 0
pmsgOut^.V1.ReplicationState.LsaCreationTime := 0
pmsgOut^.V1.ActualNtStatus := 0
pmsgOut^.V1.pRestart := null
pmsgOut^.V1.pLog := null

/* Validate the request version */
if dwInVersion # 1 then
    return ERROR_DS_DRA_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^.V1.pLog := null
        pmsgOut^.V1.cbLog := 0
    end
end

```

```

    pmsgOut^.V1.pRestart := null
    pmsgOut^.V1.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^.V1.pLog := pChangeLog
    pmsgOut^.V1.cbLog := size in bytes of pmsgOut^.V1.pLog^
/* Construct a new cookie */
    lastReturnedSerialNumber :=
        dc.pdcChangeLog[lastIndexToBeReturned].SerialNumber
    pmsgOut^.V1.pRestart :=
        ADDRESS OF implementation-specific struct
        encapsulating lastReturnedSerialNumber and sequenceNumber
    pmsgOut^.V1.cbRestart := size in bytes of pmsgOut^.V1.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 Microsoft Windows® 2000 operating system 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: 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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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: 1 if the digests of the object sequences on the client and server are the same, 0 if they are different.

cNumGuids: The number of items in the **rgGuids** array. Zero if **dwStatusFlags** = 0.

rgGuids: The [objectGUID](#)s 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 [objectGUID](#)s 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 [objectGUID](#)s 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 *digest* 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 [objectGUID](#)s of members of the set, sorted into ascending order by [GUID](#) value.
3. Find the smallest integer *i* such that $S[i] \geq \text{startGUID}$. If there is no such *i*, the result sequence *s* is empty, otherwise the result sequence *s* is as follows:

$S[i .. \min(i+\text{count}, S.\text{length})-1]$

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 the informative summary of the GuidSequence procedure (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: The lower bound on the smallest [objectGUID](#) in the cluster that the client will check.

count: The 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: [<30>](#)

```
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
*pdwOutVersion = 1;

ValidateDRSInput(hDrs, 23)

pdwOutVersion^ := 1
pmsgOut^.V1.dwStatusFlags := 0
```



```

pmsgOut^.V1.cNumGuids := 0
pmsgOut^.V1.rgGuids := null

if dwInVersion ≠ 0x1 then
    return ERROR_DS_DRA_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, 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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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.

V2: Version 2 request. 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.

```
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.

```
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, which are presented in little-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	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: Unused. MUST be zero and ignored.

MT (DS_REPL_INFO_FLAG_IMPROVE_LINKED_ATTRS, 0x00000001): 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: Zero, or the value of **dwEnumerationContext** returned by the server on a previous call to **IDL_DRSGetReplInfo**. For an informative description of the sequencing issues associated with this field, see section [1.3.2](#).

4.1.13.1.4 DS_REPL_INFO Codes

DS_REPL_INFO codes indicate the type of replication state information being requested.

Value	Meaning
DS_REPL_INFO_NEIGHBORS 0x00000000	Replication state data for each NC and source server pair, for all NC replicas hosted by this DC.

Value	Meaning
DS_REPL_INFO_CURSORS_FOR_NC 0x00000001	A portion of the replication state for the NC replica of a given NC.
DS_REPL_INFO_METADATA_FOR_OBJ 0x00000002	Stamps for all the replicated attribute of the given object.
DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES 0x00000003	Replication state data regarding connection failures with inbound replication partners.
DS_REPL_INFO_KCC_DSA_LINK_FAILURES 0x00000004	Replication state data regarding link failures with inbound replication partners.
DS_REPL_INFO_PENDING_OPS 0x00000005	Replication tasks that are currently executing or that are queued to execute.
DS_REPL_INFO_METADATA_FOR_ATTR_VALUE 0x00000006	Stamps for a specific link attribute of the given object.
DS_REPL_INFO_CURSORS_2_FOR_NC 0x00000007	A portion of the replication state for the NC replica of a given NC.
DS_REPL_INFO_CURSORS_3_FOR_NC 0x00000008	A portion of the replication state for the NC replica of a given NC.
DS_REPL_INFO_METADATA_2_FOR_OBJ 0x00000009	Stamps for all the replicated attribute of the given object.
DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE 0x0000000A	Stamps for a specific link attribute of the given object.
DS_REPL_INFO_SERVER_OUTGOING_CALLS 0xFFFFFFFFA	A list of all outstanding RPC server call contexts.
DS_REPL_INFO_UPTODATE_VECTOR_V1 0xFFFFFFFFB	The replication state for the NC replica of a given NC.
DS_REPL_INFO_CLIENT_CONTEXTS 0xFFFFFFFFC	A list of all outstanding RPC client contexts.
DS_REPL_INFO_REPSTO 0xFFFFFFFFE	Replication state data for each NC and destination server (which is notified of changes) pair, for all NC replicas hosted by this DC.

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* 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;

```

pNeighbors: Neighbor information.

pCursors: Cursors for an NC replica.

pObjMetaData: Attribute stamps.

pConnectFailures: Connection failure data.

pLinkFailures: Link failure data.

pPendingOps: Pending operations in the replication queue.

pAttrValueMetaData: Link value stamps.

pCursors2: Cursors for an NC replica.

pCursors3: Cursors for an NC replica.

pObjMetaData2: Attribute stamps.

pAttrValueMetaData2: Link value stamps.

pServerOutgoingCalls: Outstanding requests from this DC to other DCs.

pUpToDateVec: Cursors for an NC replica.

pClientContexts: Active RPC client connections.

pRepsTo: 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_NEIGHBOR rgNeighbor[];
} DS_REPL_NEIGHBORSW;
```

cNumNeighbors: The count of items in the **rgNeighbor** array.

dwReserved: Unused. MUST be 0 and ignored.

rgNeighbor: A 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_NEIGHBORW;
```

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: Unused. MUST be 0 and ignored.

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, as specified in [\[MS-ERREF\]](#) section 2.2, 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: Unused. MUST be 0 and ignored.

rgCursor: A 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 update sequence number (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: The value a client uses to populate the **dwEnumerationContext** field of the request on a future call to [IDL_DRSGetReplInfo](#) to retrieve additional results. For an informative description of the sequencing issues associated with this field, see section [1.3.2](#).

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: The value a client uses to populate the **dwEnumerationContext** field of the request on a future call to [IDL_DRSGetReplInfo](#) to retrieve additional results. For an informative description of the sequencing issues associated with this field, see section [1.3.2](#).

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: Unused. MUST be 0 and ignored.

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: An 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: Unused. MUST be 0 and ignored.

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: An 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: Unused. MUST be 0 and ignored.

rgDsaFailure: An array of [DS_REPL_KCC_DSA_FAILUREW](#) structures.

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 time at which the DC entered an error state.

cNumFailures: The number of errors that have occurred.

dwLastResult: The Windows error code, as specified in [\[MS-ERREF\]](#) section 2.2, 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 number 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 time at which the operation was requested.

ulSerialNumber: The unique ID associated with the operation.

ulPriority: The priority of the operation.

OpType: An integer that indicates the type of operation, as defined in [DS_REPL_OP_TYPE \(section 5.45\)](#).

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 number of items in **rgMetaData** array.

dwEnumerationContext: The value a client uses to populate the **dwEnumerationContext** field of the request on a future call to [IDL_DRSGetReplInfo](#) to retrieve additional results. For an informative description of the sequencing issues associated with this field, see section [1.3.2](#).

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: The binary_value portion of the attribute value if the attribute is of syntax Object(DN-Binary), or the string_value portion of the attribute value if the attribute is of syntax Object(DN-String); null otherwise.

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 number of items in the **rgMetaData** array.

dwEnumerationContext: The value a client uses to populate the **dwEnumerationContext** field of the request on a future call to [IDL_DRSGetReplInfo](#) to retrieve additional results. For an informative description of the sequencing issues associated with this field, see section [1.3.2](#).

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: The `binary_value` portion of the attribute value if the attribute is of syntax `Object(DN-Binary)`, or the `string_value` portion of the attribute value if the attribute is of syntax `Object(DN-String)`; null otherwise.

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.

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 number of items in the **rgContext** array.

dwReserved: Unused. MUST be 0 and ignored.

rgContext: A 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;
    DSTMIME 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 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 MUST be 0.

pid: The process ID 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 number of items in the **rgCall** array.

dwReserved: Unused. MUST be 0 and ignored.

rgCall: A 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;
```

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 thread ID of the thread that is using the context.

dwBindingTimeoutMins: If the context is set to be canceled, the time-out in minutes.

dstimeCreated: The date and 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
18	IDL_DRSWriteSPN

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.

4.1.13.2.4 GetUpToDateVector

```
procedure GetUpToDateVector(nc: DSName): sequence of ReplUpToDateVector
```

Returns a sequence of [ReplUpToDateVector \(section 5.162\)](#), sorted in ascending order by the **uuidDSA** field. The entries are retrieved from nc!replUpToDateVector plus an additional entry with **uuidDSA** set to the invocation ID of this server, **usnHighPropUpdate** set to rootDSE!highestCommittedUSN, and **timeLastSyncSuccess** set to the current time.

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 in the definition of DS_REPL_INFO in section [4.1.13.1.4](#).

```
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
fAccessGranted: boolean
infoTypeValid: boolean
defaultNC: DSName
object: DSName
enumerationContext: DWORD
baseIndex: DWORD
endIndex: DWORD
ncs: set of DSName
nc: DSName
i, j: DWORD
r: RepsFrom
q: 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
stamp: LinkValueStamp
```

```

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
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
v: attribute value

```

```
ValidatedDRSInput(hDrs, 19)
```

```

if dwInVersion = 1 then
    infoType = pmsgIn^ V1.InfoType
else
    infoType = pmsgIn^ V2.InfoType
endif
pdwOutVersion^ := infoType

```

```

if infoType = DS_REPL_INFO_NEIGHBORS then
    pmsgOut^.pNeighbors := null
else if infoType = DS_REPL_INFO_CURSORS_FOR_NC then
    pmsgOut^.pCursors := null
else if infoType = DS_REPL_INFO_METADATA_FOR_OBJ then
    pmsgOut^.pObjMetaData := null
else if infoType = DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES then
    pmsgOut^.pConnectFailures := null
else if infoType = DS_REPL_INFO_KCC_DSA_LINK_FAILURES then
    pmsgOut^.pLinkFailures := null
else if infoType = DS_REPL_INFO_PENDING_OPS then
    pmsgOut^.pPendingOps := null
else if infoType = DS_REPL_INFO_METADATA_FOR_ATTR_VALUE then
    pmsgOut^.pAttrValueMetaData := null
else if infoType = DS_REPL_INFO_CURSORS_2_FOR_NC then
    pmsgOut^.pCursors2 := null
else if infoType = DS_REPL_INFO_CURSORS_3_FOR_NC then
    pmsgOut^.pCursors3 := null
else if infoType = DS_REPL_INFO_METADATA_2_FOR_OBJ then
    pmsgOut^.pObjMetaData2 := null
else if infoType = DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE then
    pmsgOut^.pAttrValueMetaData2 := null
else if infoType = DS_REPL_INFO_SERVER_OUTGOING_CALLS then
    pmsgOut^.pServerOutgoingCalls := null
else if infoType = DS_REPL_INFO_UPTODATE_VECTOR_V1 then
    pmsgOut^.pUpToDateVec := null
else if infoType = DS_REPL_INFO_CLIENT_CONTEXTS then
    pmsgOut^.pClientContexts := null
else if infoType = DS_REPL_INFO_REPSTO then
    pmsgOut^.pRepsTo := null
endif

```

```

/* 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, a starting index into the result set is needed 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
    if msgIn.dwEnumerationContext = 0xffffffff then
        /* No more data is available. */
        return ERROR_NO_MORE_ITEMS
    endif
    baseIndex := msgIn.dwEnumerationContext
endif

/* Perform the necessary access checks. */
defaultNC := DefaultNC()
fAccessGranted := false
infoTypeValid := false
object := msgIn.pszObjectDN
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}) and

```

```

        object ≠ null) 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
    if object = null then
        return ERROR_INVALID_PARAMETER
    endif
    if not ObjExists(object) then
        if object.dn = null then
            return ERROR_DS_DRA_BAD_DN
        else
            return ERROR_DS_OBJ_NOT_FOUND
        endif
    endif
endif
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 fAccessGranted 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 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 = NULLGUID 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^.uuidSourceDsaObjGuid := r.uuidDsa
                    pNeighbor^.pszSourceDsaDN :=
                        GetDNFromObjectGuid(r.uuidDsa)
                    pNeighbor^.pszNamingContext := nc!distinguishedName
                    /* If a naming context is specified in the request,
                     * the uuidNamingContextObjGuid field of the response
                     * is set to the NULL GUID. */
                    if object ≠ null then
                        pNeighbor^.uuidNamingContextObjGuid := NULLGUID
                    else
                        pNeighbor^.uuidNamingContextObjGuid := nc!objectGUID
                    endif

                    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^.ftimeLastSyncAttempt := r.timeLastAttempt
                    pNeighbor^.dwLastSyncResult := r.ulResultLastAttempt
                    pNeighbor^.cNumConsecutiveSyncFailures :=
                        r.cConsecutiveFailures
                    j := j + 1
                endif
                i := i + 1
            endif
        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 := q.naDsa
      pNeighbor^.dwReplicaFlags := q.options
      pNeighbor^.ftimeLastSyncSuccess := q.timeLastSuccess
      pNeighbor^.ftimeLastSyncAttempt := q.timeLastAttempt
      pNeighbor^.dwLastSyncResult := q.ulResultLastAttempt
      pNeighbor^.cNumConsecutiveSyncFailures :=
        q.cConsecutiveFailures
      pNeighbor^.uuidSourceDsaObjGuid := q.uuidDsa
      pNeighbor^.pszSourceDsaDN := GetDNFromObjectGuid(q.uuidDsa)
      pNeighbor^.pszNamingContext := nc!distinguishedName
      /* If a naming context is specified in the request,
       * the uuidNamingContextObjGuid field of the response
       * is set to the NULL GUID. */
      if object ≠ null then
        pNeighbor^.uuidNamingContextObjGuid := NULLGUID
      else
        pNeighbor^.uuidNamingContextObjGuid := nc!objectGUID
      endif

      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

  /* Enumerate all the replicated attributes */
  attrSeq := ReplicatedAttributes()
  i := 0
  j := 0
  while (i < attrSeq.length)
    attr := attrSeq[i]
    s := AttrStamp(object, attr)

    if (IsForwardLinkAttribute(attr) and
      dwInVersion = 2 and
      DS_REPL_INFO_FLAG_IMPROVE_LINKED_ATTRS in msgIn.ulFlags)
      then
        ls := null
        foreach v in GetAttrVals(object, attr, true)
          stamp := LinkStamp(object, attr, 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. */
          if stamp ≠ null and LinkValueStampCompare(stamp, ls) > 0 then
            ls := stamp;

```



```

        endif
    endfor
    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 := NULLGUID
    s.usnOriginating := ls.usnOriginating
endif

if s ≠ null then
    if i ≥ baseIndex
        if infoType = DS_REPL_INFO_METADATA_FOR_OBJ then
            pObjMetaData := ADR(pmsgOut^.pObjMetaData^.rgMetaData[j])
            pObjMetaData^.pszAttributeName := attr
            pObjMetaData^.dwVersion := s.dwVersion
            pObjMetaData^.ftimeLastOriginatingChange := 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^.ftimeLastOriginatingChange := 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
if infoType = DS_REPL_INFO_METADATA_FOR_OBJ then
    pmsgOut^.pObjMetaData^.cNumEntries = j
else
    pmsgOut^.pObjMetaData2^.cNumEntries = j
endif
endif
endif

/* DS_REPL_INFO_CURSORS_FOR_NC */
if infoType = DS_REPL_INFO_CURSORS_FOR_NC then

    /* The NC root object must be specified */
    nc := object

    /* Parameter validation */

```

```

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 := GetUpToDatenessVector(nc)
i := baseIndex
j := 0
while i < utd.length
    pCursor := ADR(pmsgOut^.pCursors^.rgCursor[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

    /* The NC root object must be specified. */
    nc := object

    /* Parameter validation. */
    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

    i := baseIndex
    j := 0
    utd := GetUpToDatenessVector(nc)

    /* 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^.rgCursor[j])
            pCursor2^.uuidSourceDsaInvocationID := utd[i].uuidDsa
            pCursor2^.usnAttributeFilter := utd[i].usnHighPropUpdate
            pCursor2^.ftimeLastSyncSuccess := utd[i].timeLastSyncSuccess
        else
            pCursor3 := ADR(pmsgOut^.pCursors3^.rgCursor[j])

```

```

        pCursor3^.uuidSourceDsaInvocationID := utd[i].uuidDsa
        pCursor3^.usnAttributeFilter := utd[i].usnHighPropUpdate
        pCursor3^.ftimeLastSyncSuccess := utd[i].timeLastSyncSuccess
        pCursor3^.pszSourceDsaDN :=
            GetDNFromInvocationID(utd[i].uuidDsa)
    endif
    j := j + 1
    i := i + 1
endwhile
if infoType = DS_REPL_INFO_CURSORS_2_NC then
    pmsgOut^.pCursors2^.cNumCursors := j
else
    pmsgOut^.pCursors3^.cNumCursors := j
endif

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. */
    If infoType = DS_REPL_INFO_CURSORS_2_NC then
        pmsgOut^.pCursor2^.dwEnumerationContext := i
    else
        pmsgOut^.pCursors3^.dwEnumerationContext := i
    endif
else
    /* No more data is available. */
    If infoType = DS_REPL_INFO_CURSORS_2_NC then
        pmsgOut^.pCursor2^.dwEnumerationContext := 0xffffffff
    else
        pmsgOut^.pCursors3^.dwEnumerationContext := 0xffffffff
    endif
endif
endif

/* DS_REPL_INFO_UPTODATE_VECTOR_V1 */
if infoType = DS_REPL_INFO_UPTODATE_VECTOR_V1 then

    /* The NC root object must be specified. */
    nc := object
    /* Parameter validation. */

    if (nc = null) then
        return ERROR_INVALID_PARAMETER
    endif

    utd := GetUpToDateVector(nc)
    for i := 0 to utd.length - 1
        pCursor := ADR(pmsgOut^.pUpToDateVec^.rgCursors[i])
        pCursor^.uuidSourceDsaInvocationID := utd[i].uuidDsa
        pCursor^.usnAttributeFilter := 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

```

```

/* 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
    attrsSeq := select all a in attrs where
        GetAttrVals(object, a, true) # null
    attr := attrsSeq[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
        endif
        j := j + 1
    endwhile
endif

```

```

else
    pAttrValueMetaData2 :=
        ADR(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 infoType = DS_REPL_INFO_METADATA_FOR_ATTR_VALUE then
    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^.pAttrValueMetaData^.dwEnumerationContext := i
    else
        /* No more data is available. */
        pmsgOut^.pAttrValueMetaData^.dwEnumerationContext :=
            0xffffffff
    endif
    pmsgOut^.pAttrValueMetaData^.cNumEntries = j
else
    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
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_DSA_LINK_FAILURES */
if infoType = DS_REPL_INFO_KCC_DSA_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
    endfor
endif

```

```

        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^.rgCall[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

```

4.1.13.4 Examples of the IDL_DRSGetReplInfo Method

4.1.13.4.1 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_NEIGHBORS to find replication neighbors for a specified NC

In this example, the domain administrator wants to see which source DCs DC1 receives replication updates from for the domain NC CONTOSO.COM. The domain administrator does so by issuing a request to DC1 with **pszObjectDN** set to the DN of the domain NC.

4.1.13.4.1.1 Initial State

Querying the NC root object for domain NC CONTOSO.COM on DC1:

- `ldap_search_s("DC=contoso,DC=com", baseObject, "(objectClass=domainDNS)", [objectClass, repsFrom])`
- Getting 1 entry:
- Dn: DC=contoso,DC=com
 - 3> objectClass: top; domain; domainDNS;
 - 1> repsFrom: dwVersion: 2 v1.cb: 492 v1.cConsecutive Failures: 0 v1.timeLastSuccess: 12924402382 v1.timeLastAttempt: 12924402382 v1.ulResultLastAttempt: 0

```
v1.cbOtherDraOffset: 216v1.cbOtherDra: 276v1.ulReplicaFlags: 112 v1.rtSchedule:
<skipped> v1.usnvec.usnHighObjUpdate: 19332 v1.usnvec.usnHighPropUpdate: 19332
v1.pszUuidDsaObj: 12626d52-1da7-4a40-a490-987c0880c3fe v1.pszUuidInvocId: 44a2959c-
bb0d-4b2e-b106-fd8235288ee4 v1.pszUuidTransportObj: 00000000-0000-0000-0000-
000000000000 v1.cbPASDataOffset: 0 v1~PasData: (none) v2~pdsa_rpc_inst
v2.pszDSIServer 12626d52-1da7-4a40-a490-987c0880c3fe._msdcs.contoso.com
v2.pszDSIAnnotation (null) v2.pszDSIInstance 12626d52-1da7-4a40-a490-
987c0880c3fe._msdcs.contoso.com v2.pguidDSIInstance (null);
```

4.1.13.4.1.2 Client Request

The client invokes the [IDL DRSGetReplInfo \(section 4.1.13\)](#) method against DC1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 2
- pmsgIn = DRS_MSG_GETREPLINFO_REQ_V2
 - InfoType = DS_REPL_INFO_NEIGHBORS
 - pszObjectDN = "DC=contoso,DC=com"
 - uuidSourceDsaObjGuid = 00000000-0000-0000-0000-000000000000
 - ulFlags = 0x0
 - pszAttributeName = (null)
 - pszValueDN = (null)
 - dwEnumerationContext = 0

4.1.13.4.1.3 Server Response

A return code of 0 with the following values:

- pdwOutVersion = DS_REPL_INFO_NEIGHBORS
- pmsgOut = DS_REPL_NEIGHBORSW
 - cNumNeighbors = 1
 - dwReserved = 0
 - rgNeighbor = DS_REPL_NEIGHBORW[]
 - rgNeighbor[0]
 - pszNamingContext = DC=contoso,dc=com
 - pszSourceDsaDN = CN=NTDS Settings,CN=DC2,CN=Servers,CN=Default-First-Site-Name,CN= Sites,CN=Configuration,DC=contoso,DC=com
 - pszSourceDsaAddress = 12626d52-1da7-4a40-a490-987c0880c3fe._msdcs.contoso.com
 - pszAsyncIntersiteTransportDN = (null)
 - dwReplicaFlags = 0x0

- dwReserved = 0
- uuidNamingContextObjGuid = 00000000-0000-0000-0000-000000000000
- uuidSourceDsaObjGuid = 12626d52-1da7-4a40-a490-987c0880c3fe
- uuidSourceDsaInvocationID = 44a2959c-bb0d-4b2e-b106-fd8235288ee4
- uuidAsyncIntersiteTransportObjGuid = 00000000-0000-0000-0000-000000000000
- usnLastObjChangeSynced = 20002
- usnAttributeFilter = 20002
- ftimeLastSyncSuccess.dwLowDateTime = 0x4aaeb00
- ftimeLastSyncSuccess.dwHighDateTime = 0x1cb2ad2
- ftimeLastSyncAttempt.dwLowDateTime = 0x4aaeb00
- ftimeLastSyncAttempt.dwHighDateTime = 0x1cb2ad2
- dwLastSyncResult = 0
- cNumConsecutiveSyncFailures = 0

4.1.13.4.1.4 Final State

The final state is the same as the initial state; there is no change.

4.1.13.4.2 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_NEIGHBORS to find which naming contexts a DC receives updates for from a replication neighbor

In this example, the domain administrator checks which NCs DCA1 receives replication updates for from DC1. The domain administrator does so by issuing a request to DCA1 with **pszObjectDN** set to null and **uuidSourceDsaObjGuid** set to the DSA GUID of DC1.

4.1.13.4.2.1 Initial State

Querying the nTDSDSA object for DC1 on DCA1:

- 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, objectGUID])
- Getting 1 entry:
- >> 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> objectGUID: 2e235fab-353c-46fc-8afd-437e9d0188b3;

Querying the NC root object for config NC CN=Configuration,DC=contoso,DC=com on DCA1:

- `ldap_search_s("CN=Configuration,DC=contoso,DC=com", baseObject, "(objectClass=*)", [objectClass, repsFrom])`
- Getting 1 entry:
- `>> Dn: CN=Configuration,DC=contoso,DC=com`
 - `2> objectClass: top; configuration;`
 - `1> repsFrom: dwVersion: 2 v1.cb: 492 v1.cConsecutive Failures: 0 v1.timeLastSuccess: 12924750622 v1.timeLastAttempt: 12924750622 v1.ulResultLastAttempt: 0 v1.cbOtherDraOffset: 216v1.cbOtherDra: 276v1.ulReplicaFlags: 805306448 v1.rtSchedule: <skipped> v1.usnvec.usnHighObjUpdate: 24573 v1.usnvec.usnHighPropUpdate: 24573 v1.pszUuidDsaObj: 2e235fab-353c-46fc-8afd-437e9d0188b3 v1.pszUuidInvocId: 2e235fab-353c-46fc-8afd-437e9d0188b3 v1.pszUuidTransportObj: 00000000-0000-0000-0000-000000000000 v1.cbPASDataOffset: 0 v1~PasData: (none) v2~pdsa_rpc_inst v2.pszDSIServer 2e235fab-353c-46fc-8afd-437e9d0188b3._msdcs.contoso.com v2.pszDSIAnnotation (null) v2.pszDSIInstance 2e235fab-353c-46fc-8afd-437e9d0188b3._msdcs.contoso.com v2.pguidDSIInstance (null);`

Querying the NC root object for schema NC CN=Schema,CN=Configuration,DC=contoso,DC=com on DCA1:

- `ldap_search_s("CN=Schema,CN=Configuration,DC=contoso,DC=com", baseObject, "(objectClass=*)", [objectClass, repsFrom])`
- Getting 1 entry:
- `>> Dn: CN=Schema,CN=Configuration,DC=contoso,DC=com`
 - `2> objectClass: top; dMD;`
 - `1> repsFrom: dwVersion: 2 v1.cb: 492 v1.cConsecutive Failures: 0 v1.timeLastSuccess: 12924750622 v1.timeLastAttempt: 12924750622 v1.ulResultLastAttempt: 0 v1.cbOtherDraOffset: 216v1.cbOtherDra: 276v1.ulReplicaFlags: 2952790096 v1.rtSchedule: <skipped> v1.usnvec.usnHighObjUpdate: 24573 v1.usnvec.usnHighPropUpdate: 24573 v1.pszUuidDsaObj: 2e235fab-353c-46fc-8afd-437e9d0188b3 v1.pszUuidInvocId: 2e235fab-353c-46fc-8afd-437e9d0188b3 v1.pszUuidTransportObj: 00000000-0000-0000-0000-000000000000 v1.cbPASDataOffset: 0 v1~PasData: (none) v2~pdsa_rpc_inst v2.pszDSIServer 2e235fab-353c-46fc-8afd-437e9d0188b3._msdcs.contoso.com v2.pszDSIAnnotation (null) v2.pszDSIInstance 2e235fab-353c-46fc-8afd-437e9d0188b3._msdcs.contoso.com v2.pguidDSIInstance (null);`

Querying the NC root object for NC DC=ForestDnsZones,DC=contoso,DC=com on DCA1:

- `ldap_search_s("DC=ForestDnsZones,DC=contoso,DC=com ", baseObject, "(objectClass=*)", [objectClass, repsFrom])`
- Getting 1 entry:
- `>> Dn: DC=ForestDnsZones,DC=contoso,DC=com`
 - `3> objectClass: top; domain; domainDNS;`
 - `1> repsFrom: dwVersion: 2 v1.cb: 492 v1.cConsecutive Failures: 0 v1.timeLastSuccess: 12924750622 v1.timeLastAttempt: 12924750622 v1.ulResultLastAttempt: 0 v1.cbOtherDraOffset: 216v1.cbOtherDra: 276v1.ulReplicaFlags: 805306448 v1.rtSchedule:`

```
<skipped> v1.usnvec.usnHighObjUpdate: 24573 v1.usnvec.usnHighPropUpdate: 24573
v1.pszUuidDsaObj: 2e235fab-353c-46fc-8afd-437e9d0188b3 v1.pszUuidInvocId: 2e235fab-
353c-46fc-8afd-437e9d0188b3 v1.pszUuidTransportObj: 00000000-0000-0000-0000-
000000000000 v1.cbPASDataOffset: 0 v1~PasData: (none) v2~pdsa_rpc_inst
v2.pszDSIServer 2e235fab-353c-46fc-8afd-437e9d0188b3._msdcs.contoso.com
v2.pszDSIAnnotation (null) v2.pszDSIInstance 2e235fab-353c-46fc-8afd-
437e9d0188b3._msdcs.contoso.com v2.pguidDSIInstance (null);
```

4.1.13.4.2.2 Client Request

The client invokes the [IDL DRSGetReplInfo \(section 4.1.13\)](#) method against DCA1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 2
- pmsgIn = DRS_MSG_GETREPLINFO_REQ_V2
 - InfoType = DS_REPL_INFO_NEIGHBORS
 - pszObjectDN = (null)
 - uuidSourceDsaObjGuid = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - ulFlags = 0x0
 - pszAttributeName = (null)
 - pszValueDN = (null)
 - dwEnumerationContext = 0

4.1.13.4.2.3 Server Response

A return code of 0 with the following values:

- pdwOutVersion = DS_REPL_INFO_NEIGHBORS
- pmsgOut = DS_REPL_NEIGHBORSW
 - cNumNeighbors = 3
 - dwReserved = 0
 - rgNeighbor = DS_REPL_NEIGHBORW[]
 - rgNeighbor[0]
 - pszNamingContext = CN=Configuration,DC=contoso,DC=com
 - pszSourceDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN= Sites,CN=Configuration,DC=contoso,DC=com
 - pszSourceDsaAddress = 2e235fab-353c-46fc-8afd-437e9d0188b3._msdcs.contoso.com
 - pszAsyncIntersiteTransportDN = (null)
 - dwReplicaFlags = 0x0

- dwReserved = 0
- uuidNamingContextObjGuid = 64f4ed75-28b1-42f3-b7c9-6ac234db9a9e
- uuidSourceDsaObjGuid = 2e235fab-353c-46fc-8afd-437e9d0188b3
- uuidSourceDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
- uuidAsyncIntersiteTransportObjGuid = 00000000-0000-0000-0000-000000000000
- usnLastObjChangeSynced = 24523
- usnAttributeFilter = 24523
- ftimeLastSyncSuccess.dwLowDateTime = 0xf7e80900
- ftimeLastSyncAttempt.dwHighDateTime = 0x1cb2de9
- ftimeLastSyncAttempt.dwLowDateTime = 0xf7e80900
- ftimeLastSyncAttempt.dwHighDateTime = 0x1cb2de9
- dwLastSyncResult = 0
- cNumConsecutiveSyncFailures = 0
- rgNeighbor[1]
 - pszNamingContext = CN=Schema,CN=Configuration,DC=contoso,DC=com
 - pszSourceDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - pszSourceDsaAddress = 2e235fab-353c-46fc-8afd-437e9d0188b3._msdcs.contoso.com
 - pszAsyncIntersiteTransportDN = (null)
 - dwReplicaFlags = 0x0
 - dwReserved = 0
 - uuidNamingContextObjGuid = f3ba2060-2d67-43e6-a334-54a8f1ecc78a
 - uuidSourceDsaObjGuid = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - uuidSourceDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - uuidAsyncIntersiteTransportObjGuid = 00000000-0000-0000-0000-000000000000
 - usnLastObjChangeSynced = 24523
 - usnAttributeFilter = 24523
 - ftimeLastSyncSuccess.dwLowDateTime = 0xf7e80900
 - ftimeLastSyncSuccess.dwHighDateTime = 0x1cb2de9
 - ftimeLastSyncAttempt.dwLowDateTime = 0xf7e80900
 - ftimeLastSyncAttempt.dwHighDateTime = 0x1cb2de9

- dwLastSyncResult = 0
- cNumConsecutiveSyncFailures = 0
- rgNeighbor[2]
 - pszNamingContext = DC=ForestDnsZones,DC=contoso,DC=com
 - pszSourceDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - pszSourceDsaAddress = 2e235fab-353c-46fc-8afd-437e9d0188b3._msdcs.contoso.com
 - pszAsyncIntersiteTransportDN = (null)
 - dwReplicaFlags = 0x0
 - dwReserved = 0
 - uuidNamingContextObjGuid = 7fafd728-d866-4cf3-915f-78ff680603d4
 - uuidSourceDsaObjGuid = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - uuidSourceDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - uuidAsyncIntersiteTransportObjGuid = 00000000-0000-0000-0000-000000000000
 - usnLastObjChangeSynced = 24523
 - usnAttributeFilter = 24523
 - ftimeLastSyncSuccess.dwLowDateTime = 0xf7e80900
 - ftimeLastSyncSuccess.dwHighDateTime = 0x1cb2de9
 - ftimeLastSyncAttempt.dwLowDateTime = 0xf7e80900
 - ftimeLastSyncAttempt.dwHighDateTime = 0x1cb2de9
 - dwLastSyncResult = 0
 - cNumConsecutiveSyncFailures = 0

4.1.13.4.2.4 Final State

The final state is the same as the initial state; there is no change.

4.1.13.4.3 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_REPSTO to find replication neighbors for a specified NC

In this example, the domain administrator verifies which DCs DC1 sends replication updates to for the domain NC CONTOSO.COM. The domain administrator does this by issuing a request to DC1 with **pszObjectDN** set to the DN of the domain NC.

4.1.13.4.3.1 Initial State

Querying the NC root object for domain NC CONTOSO.COM on DC1:

- `ldap_search_s("DC=contoso,DC=com", baseObject, "(objectClass=domainDNS)", [objectClass, repsTo])`
- Getting 1 entry:
- `>> Dn: DC=contoso,DC=com`
 - `3> objectClass: top; domain; domainDNS;`
 - `1> repsTo: dwVersion: 2 v1.cb: 492 v1.cConsecutive Failures: 0 v1.timeLastSuccess: 12924828300 v1.timeLastAttempt: 12924828300 v1.ulResultLastAttempt: 0 v1.cbOtherDraOffset: 216v1.cbOtherDra: 276v1.ulReplicaFlags: 16 v1.rtSchedule: <skipped> v1.usnvec.usnHighObjUpdate: 0 v1.usnvec.usnHighPropUpdate: 0 v1.pszUuidDsaObj: 12626d52-1da7-4a40-a490-987c0880c3fe v1.pszUuidInvocId: 00000000-0000-0000-0000-000000000000 v1.pszUuidTransportObj: 00000000-0000-0000-0000-000000000000 v1.cbPASDataOffset: 0 v1~PasData: (none) v2~pdsa_rpc_inst v2.pszDSIServer 12626d52-1da7-4a40-a490-987c0880c3fe._msdcs.contoso.com v2.pszDSIAnnotation (null) v2.pszDSIInstance 12626d52-1da7-4a40-a490-987c0880c3fe._msdcs.contoso.com v2.pguidDSIInstance (null);`

4.1.13.4.3.2 Client Request

The client invokes the [IDL DRSGetReplInfo \(section 4.1.13\)](#) method against DC1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- `dwInVersion = 2`
- `pmsgIn = DRS_MSG_GETREPLINFO_REQ_V2`
 - `InfoType = DS_REPL_INFO_REPSTO`
 - `pszObjectDN = DC=contoso,DC=com`
 - `uuidSourceDsaObjGuid = 00000000-0000-0000-0000-000000000000`
 - `ulFlags = 0x0`
 - `pszAttributeName = (null)`
 - `pszValueDN = (null)`
 - `dwEnumerationContext = 0`

4.1.13.4.3.3 Server Response

A return code of 0 with the following values:

- `pdwOutVersion = DS_REPL_INFO_REPSTO`
- `pmsgOut = DS_REPL_NEIGHBORSW`
 - `cNumNeighbors = 1`
 - `dwReserved = 0`
 - `rgNeighbor = DS_REPL_NEIGHBORW[]`
 - `rgNeighbor[0]`

- pszNamingContext = DC=contoso,DC=com
- pszSourceDsaDN = CN=NTDS Settings,CN=DC2,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
- pszSourceDsaAddress = 12626d52-1da7-4a40-a490-987c0880c3fe._msdcs.contoso.com
- pszAsyncIntersiteTransportDN = (null)
- dwReplicaFlags = 0x0
- dwReserved = 0
- uuidNamingContextObjGuid = 00000000-0000-0000-0000-000000000000
- uuidSourceDsaObjGuid = 12626d52-1da7-4a40-a490-987c0880c3fe
- uidSourceDsaInvocationID = 00000000-0000-0000-0000-000000000000
- uuidAsyncIntersiteTransportObjGuid = 00000000-0000-0000-0000-000000000000
- usnLastObjChangeSynced = 0
- usnAttributeFilter = 0
- ftimeLastSyncSuccess.dwLowDateTime = 0x6a6bee00
- ftimeLastSyncSuccess.dwHighDateTime = 0x1cb2ea9
- ftimeLastSyncAttempt.dwLowDateTime = 0x6a6bee00
- ftimeLastSyncAttempt.dwHighDateTime = 0x1cb2ea9
- dwLastSyncResult = 0
- cNumConsecutiveSyncFailures = 0

4.1.13.4.3.4 Final State

The final state is the same as the initial state; there is no change.

4.1.13.4.4 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_CURSORS_3_FOR_NC

In this example, the domain administrator wants to view the replication state on DC2 relative to the config NC CN=Configuration,DC=contoso,DC=com. The domain administrator does this by issuing a request to DC1 with **pszObjectDN** set to the DN of the config NC.

4.1.13.4.4.1 Initial State

Querying the NC root object for the config NC CN=Configuration,DC=contoso,DC=com on DC2:

- ldap_search_s("CN=Configuration,DC=contoso,DC=com", baseObject, "(objectClass=configuration)", [objectClass, replUpToDateVector])
- Getting 1 entry:
- >> Dn: CN=Configuration,DC=contoso,DC=com

- 2> objectClass: top; configuration;
- 1> replUpToDateVector: dwVersion: 2, dwReserved1: 0, V2.cNumCursors: 2, V2.dwReserved2: 0, rgCursors: {uuidDsa: 2e235fab-353c-46fc-8afd-437e9d0188b3, usnHighPropUpdate: 22378, timeLastSyncSuccess: 07/26/2010 16:00:19}, {uuidDsa: e4dfc4c0-381c-48c9-a563-cb27db448753, usnHighPropUpdate: 18177, timeLastSyncSuccess: 07/26/2010 16:02:32};

4.1.13.4.4.2 Client Request

The client invokes the [IDL DRSGetReplInfo \(section 4.1.13\)](#) method against DC2 with the following parameters ([DRS_HANDLE](#) to DC2 omitted):

- dwInVersion = 2
- pmsgIn = DRS_MSG_GETREPLINFO_REQ_V2
 - InfoType = DS_REPL_INFO_CURSORS_3_FOR_NC
 - pszObjectDN = CN=Configuration,DC=contoso,dc=com
 - uuidSourceDsaObjGuid = 00000000-0000-0000-0000-000000000000
 - ulFlags = 0x0
 - pszAttributeName = (null)
 - pszValueDN = (null)
 - dwEnumerationContext = 0

4.1.13.4.4.3 Server Response

A return code of 0 with the following values:

- pdwOutVersion = DS_REPL_INFO_CURSORS_3_FOR_NC
- pmsgOut = DS_REPL_CURSORS_3W
 - cNumCursors = 3
 - dwEnumerationContext = 0xffffffff
 - rgCursor = DS_REPL_CURSOR_3W[]
 - rgCursor[0]
 - usnAttributeFilter = 22378
 - uuidSourceDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - ftimeLastSyncSuccess.dwLowDateTime = 0x517f0380
 - ftimeLastSyncSuccess.dwHighDateTime = 0x1cb2d16
 - pszSourceDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com

- rgCursor[1]
 - usnAttributeFilter = 43601
 - uuidSourceDsaInvocationID = 44a2959c-bb0d-4b2e-b106-fd8235288ee4
 - ftimeLastSyncSuccess.dwLowDateTime = 0xaf135000
 - ftimeLastSyncSuccess.dwHighDateTime = 0x1cb2d16
 - pszSourceDsaDN = CN=NTDS Settings,CN=DC2,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
- rgCursor[2]
 - usnAttributeFilter = 18177
 - uuidSourceDsaInvocationID = e4dfc4c0-381c-48c9-a563-cb27db448753
 - ftimeLastSyncSuccess.dwLowDateTime = 0xa0c53400
 - ftimeLastSyncSuccess.dwHighDateTime = 0x1cb2d16
 - pszSourceDsaDN = CN=NTDS Settings,CN=DCA1,CN=Servers,CN=Default-Second-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com

4.1.13.4.4.4 Final State

The final state is the same as the initial state; there is no change.

4.1.13.4.5 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_METADATA_2_FOR_OBJ

In this example, the domain administrator wants to view the replication state of the organizational unit OU1 on DC1.

4.1.13.4.5.1 Initial State

Querying OU1 on DC1:

- ldap_search_s("OU=OU1,DC=contoso,DC=com", baseObject, "(objectClass=*)", [*])
- Getting 1 entry:
- >> Dn: OU=OU1,DC=contoso,DC=com
 - 1> distinguishedName: OU=OU1,DC=contoso,DC=com;
 - 3> dSCorePropagationData: 7/27/2010 10:20:24 PM Pacific Daylight Time; 7/27/2010 10:20:23 PM Pacific Daylight Time; 0x0 = (), 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> name:OU1;
 - 1> objectCategory: CN=Organizational-Unit,CN=Schema,CN=Configuration,DC=contoso,DC=com;

- 2> objectClass: top; organizationalUnit;
- 1> objectGUID: 1a0c2e8f-2747-4e38-80fb-074e2dd3df8c;
- 1> ou: OU1;
- 1> uSNChanged: 25426;
- 1> uSNCreated: 25424;
- 1> whenChanged: 7/27/2010 10:20:23 PM Pacific Daylight Time;
- 1> whenCreated: 7/27/2010 10:20:23 PM Pacific Daylight Time;

4.1.13.4.5.2 Client Request

The client invokes the [IDL DRSGetReplInfo \(section 4.1.13\)](#) method against DC2 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 2
- pmsgIn = DRS_MSG_GETREPLINFO_REQ_V2
 - InfoType = DS_REPL_INFO_METADATA_2_FOR_OBJ
 - pszObjectDN = OU=OU1,DC=contoso,dc=com
 - uuidSourceDsaObjGuid = 00000000-0000-0000-0000-000000000000
 - ulFlags = 0x0
 - pszAttributeName = (null)
 - pszValueDN = (null)
 - dwEnumerationContext = 0

4.1.13.4.5.3 Server Response

A return code of 0 with the following values:

- pdwOutVersion = DS_REPL_INFO_METADATA_2_FOR_OBJ
- pmsgOut = DS_REPL_OBJ_META_DATA_2
 - cNumEntries = 7
 - dwReserved = 0
 - rgMetaData = DS_REPL_ATTR_META_DATA_2[]
 - rgMetaData[0]
 - pszAttributeName = objectClass
 - dwVersion = 1
 - ftimeLastOriginatingChange.dwLowDateTime = 0x94270580

- `ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2e14`
- `uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3`
- `usnOriginatingChange = 25424`
- `usnLocalChange = 25424`
- `pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com`
- `rgMetaData[1]`
 - `pszAttributeName = ou`
 - `dwVersion = 1`
 - `ftimeLastOriginatingChange.dwLowDateTime = 0x94270580`
 - `ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2e14`
 - `uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3`
 - `usnOriginatingChange = 25424`
 - `usnLocalChange = 25424`
 - `pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com`
- `rgMetaData[2]`
 - `pszAttributeName = instanceType`
 - `dwVersion = 1`
 - `ftimeLastOriginatingChange.dwLowDateTime = 0x94270580`
 - `ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2e14`
 - `uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3`
 - `usnOriginatingChange = 25424`
 - `usnLocalChange = 25424`
 - `pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com`
- `rgMetaData[3]`
 - `pszAttributeName = whenCreated`
 - `dwVersion = 1`
 - `ftimeLastOriginatingChange.dwLowDateTime = 0x94270580`
 - `ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2e14`
 - `uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3`

- usnOriginatingChange = 25424
- usnLocalChange = 25424
- pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
- rgMetaData[4]
 - pszAttributeName = nTSecurityDescriptor
 - dwVersion = 2
 - ftimeLastOriginatingChange.dwLowDateTime = 0x94270580
 - ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2e14
 - uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - usnOriginatingChange = 25426
 - usnLocalChange = 25426
 - pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
- rgMetaData[5]
 - pszAttributeName = name
 - dwVersion = 1
 - ftimeLastOriginatingChange.dwLowDateTime = 0x94270580
 - ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2e14
 - uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - usnOriginatingChange = 25424
 - usnLocalChange = 25424
 - pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
- rgMetaData[6]
 - pszAttributeName = objectCategory
 - dwVersion = 1
 - ftimeLastOriginatingChange.dwLowDateTime = 0x94270580
 - ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2e14
 - uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - usnOriginatingChange = 25424
 - usnLocalChange = 25424

- pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com

4.1.13.4.5.4 Final State

The final state is the same as the initial state; there is no change.

4.1.13.4.6 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE to view the replication metadata for all values of a link value attribute

In this example, the domain administrator requires viewing the replication state of the link value attribute member of the group GroupB on DC2.

4.1.13.4.6.1 Initial State

Querying the GroupB on DC2:

- ldap_search_s("CN=GroupB,CN=Users,DC=contoso,DC=com", baseObject, "(objectClass=group)", [objectClass, member])
- Getting 1 entry:
- >> Dn: CN=GroupB,CN=Users,DC=contoso,DC=com
 - 3> member: CN=GroupC,CN=Users,DC=contoso,DC=com;
CN=GroupA,CN=Users,DC=contoso,DC=com; CN=Kim
Akers,CN=Users,DC=contoso,DC=com;
- 2> objectClass: top; group;

4.1.13.4.6.2 Client Request

The client invokes the [IDL_DRSGetReplInfo \(section 4.1.13\)](#) method against DC2 with the following parameters ([DRS_HANDLE](#) to DC2 omitted):

- dwInVersion = 2
- pmsgIn = DRS_MSG_GETREPLINFO_REQ_V2
 - InfoType = DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE
 - pszObjectDN = CN=GroupB,CN=Users,DC=contoso,dc=com
 - uuidSourceDsaObjGuid = 00000000-0000-0000-0000-000000000000
 - ulFlags = 0x0
 - pszAttributeName = member
 - pszValueDN = (null)
 - dwEnumerationContext = 0

4.1.13.4.6.3 Server Response

A return code of 0 with the following values:

- pdwOutVersion = DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE
- pmsgOut = DS_REPL_ATTR_VALUE_META_DATA_2
 - cNumEntries = 3
 - dwEnumerationContext = 0xffffffff
 - rgMetaData = DS_REPL_VALUE_META_DATA_2[]
 - rgMetaData[0]
 - pszAttributeName = member
 - pszObjectDn = CN=Kim Akers,CN=Users,DC=contoso,DC=com
 - cbData = 0
 - pbData = null
 - ftimeDeleted.dwLowDateTime = 0x0
 - ftimeDeleted.dwHighDateTime = 0x0
 - ftimeCreated.dwLowDateTime = 0xc3a4dd80
 - ftimeCreated.dwHighDateTime = 0x1cb2ab5
 - dwVersion = 1
 - ftimeLastOriginatingChange.dwLowDateTime = 0xc3a4dd80
 - ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2ab5
 - uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - usnOriginatingChange = 15399
 - usnLocalChange = 19212
 - pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - rgMetaData[1]
 - pszAttributeName = member
 - pszObjectDn = CN=GroupA,CN=Users,DC=contoso,DC=com
 - cbData = 0
 - pbData = null
 - ftimeDeleted.dwLowDateTime = 0x0
 - ftimeDeleted.dwHighDateTime = 0x0
 - ftimeCreated.dwLowDateTime = 0x2fb77680
 - ftimeCreated.dwHighDateTime = 0x1cb2e13

- dwVersion = 1
- ftimeLastOriginatingChange.dwLowDateTime = 0x2fb77680
- ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2e13
- uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
- usnOriginatingChange = 25384
- usnLocalChange = 46509
- pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
- rgMetaData[2]
 - pszAttributeName = member
 - pszObjectDn = CN=GroupC,CN=Users,DC=contoso,DC=com
 - cbData = 0
 - pbData = null
 - ftimeDeleted.dwLowDateTime = 0x0
 - ftimeDeleted.dwHighDateTime = 0x0
 - ftimeCreated.dwLowDateTime = 0x2fb77680
 - ftimeCreated.dwHighDateTime = 0x1cb2e13
 - dwVersion = 1
 - ftimeLastOriginatingChange.dwLowDateTime = 0x2fb77680
 - ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2e13
 - uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - usnOriginatingChange = 25385
 - usnLocalChange = 46508
 - pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com

4.1.13.4.6.4 Final State

The final state is the same as the initial state; there is no change.

4.1.13.4.7 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE to view the replication metadata for a specific value of a link value attribute

In this example, the domain administrator wants to view the replication state of link value attribute member of the group GroupB on DC2. Specifically, the domain administrator is interested in the

member value corresponding to Kim Akers' account. The domain administrator does so by issuing a request to DC2 with **pszObjectDN** set to the DN of group B, **pszAttributeName** set to member, and **pszValueDN** set to the DN of Kim Akers' user account.

4.1.13.4.7.1 Initial State

Querying the GroupB on DC2:

- `ldap_search_s("CN=GroupB,CN=Users,DC=contoso,DC=com", baseObject, "(objectClass=group)", [objectClass, member])`
- Getting 1 entry:
- `>> Dn: CN=GroupB,CN=Users,DC=contoso,DC=com`
 - `3> member: CN=GroupC,CN=Users,DC=contoso,DC=com;`
`CN=GroupA,CN=Users,DC=contoso,DC=com; CN=Kim`
`Akers,CN=Users,DC=contoso,DC=com;`
 - `2> objectClass: top; group;`

4.1.13.4.7.2 Client Request

The client invokes the [IDL DRSGetReplInfo \(section 4.1.13\)](#) method against DC2 with the following parameters ([DRS_HANDLE](#) to DC2 omitted):

- `dwInVersion = 2`
- `pmsgIn = DRS_MSG_GETREPLINFO_REQ_V2`
 - `InfoType = DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE`
 - `pszObjectDN = CN=GroupB,CN=Users,DC=contoso,dc=com`
 - `uuidSourceDsaObjGuid = 00000000-0000-0000-0000-000000000000`
 - `ulFlags = 0x0`
 - `pszAttributeName = member`
 - `pszValueDN = CN=Kim Akers,CN=Users,DC=contoso,DC=com`
 - `dwEnumerationContext = 0`

4.1.13.4.7.3 Server Response

A return code of 0 with the following values:

- `pdwOutVersion = DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE`
- `pmsgOut = DS_REPL_ATTR_VALUE_META_DATA_2`
 - `cNumEntries = 1`
 - `dwEnumerationContext = 0x1`
 - `rgMetaData = DS_REPL_VALUE_META_DATA_2[]`

- rgMetaData[0]
 - pszAttributeName = member
 - pszObjectDn = CN=Kim Akers,CN=Users,DC=contoso,DC=com
 - cbData = 0
 - pbData = null
 - ftimeDeleted.dwLowDateTime = 0x0
 - ftimeDeleted.dwHighDateTime = 0x0
 - ftimeCreated.dwLowDateTime = 0xc3a4dd80
 - ftimeCreated.dwHighDateTime = 0x1cb2ab5
 - dwVersion = 1
 - ftimeLastOriginatingChange.dwLowDateTime = 0xc3a4dd80
 - ftimeLastOriginatingChange.dwHighDateTime = 0x1cb2ab5
 - uuidLastOriginatingDsaInvocationID = 2e235fab-353c-46fc-8afd-437e9d0188b3
 - usnOriginatingChange = 15399
 - usnLocalChange = 19212
 - pszLastOriginatingDsaDN = CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com

4.1.13.4.7.4 Final State

The final state is the same as the initial state; there is no change.

4.1.13.4.8 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES

In this example, the domain administrator verifies whether DC1 has any replication failures.

4.1.13.4.8.1 Initial State

DC2 is a replication neighbor of DC1. DC2 is offline and DC1 is unable to contact DC2 to query its replication state.

4.1.13.4.8.2 Client Request

The client invokes the [IDL_DRSGetReplInfo \(section 4.1.13\)](#) method against DC1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 2
- pmsgIn = DRS_MSG_GETREPLINFO_REQ_V2
 - InfoType = DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES

- pszObjectDN = (null)
- uuidSourceDsaObjGuid = 00000000-0000-0000-0000-000000000000
- ulFlags = 0x0
- pszAttributeName = (null)
- pszValueDN = (null)
- dwEnumerationContext = 0

4.1.13.4.8.3 Server Response

A return code of 0 with the following values:

- pdwOutVersion = DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES
- pmsgOut = DS_REPL_KCC_DSA_FAILURESW
 - cNumEntries = 1
 - dwReserved = 0
 - rgDsaFailure = DS_REPL_KCC_DSA_FAILUREW[]
 - rgDsaFailure[0]
 - pszDsaDN = CN=NTDS Settings,CN=DC2,CN=Servers,CN=Default-First-Site-Name,CN= Sites,CN=Configuration,DC=contoso,DC=com
 - uuidDsaObjGuid = 12626d52-1da7-4a40-a490-987c0880c3fe
 - ftimeFirstFailure.dwLowDateTime = 0xcefc4100
 - ftimeFirstFailure.dwHighDateTime = 0x1cb2d21
 - cNumFailures = 2
 - dwLastResult = 1722

4.1.13.4.8.4 Final State

The final state is the same as the [initial state \(section 4.1.13.4.8.1\)](#); there is no change.

4.1.13.4.9 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_PENDING_OPS

In this example, the domain administrator verifies whether DC1 has any pending operations in its replication queue.

4.1.13.4.9.1 Initial State

DC2 is a replication neighbor of DC1. DC1 is syncing updates from DC2.

4.1.13.4.9.2 Client Request

The client invokes the [IDL DRSGetReplInfo \(section 4.1.13\)](#) method against DC1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwInVersion = 2
 - pmsgIn = DRS_MSG_GETREPLINFO_REQ_V2
 - InfoType = DS_REPL_INFO_PENDING_OPS
 - pszObjectDN = (null)
 - uuidSourceDsaObjGuid = 00000000-0000-0000-0000-000000000000
 - ulFlags = 0x0
 - pszAttributeName = (null)
 - pszValueDN = (null)
 - dwEnumerationContext = 0

4.1.13.4.9.3 Server Response

A return code of 0 with the following values:

- pdwOutVersion = DS_REPL_INFO_PENDING_OPS
- pmsgOut = DS_REPL_PENDING_OPSPW
 - ftimeCurrentOpStarted.dwLowDateTime = 0x2546bc80
 - ftimeCurrentOpStarted.dwHighDateTime = 0x1cb2ea7
 - cNumPendingOps = 1
 - rgPendingOp = DS_REPL_OPW[]
 - rgPendingOp[0]
 - ftimeEnqueued.dwLowDateTime = 0x2546bc80
 - ftimeEnqueued.dwHighDateTime = 0x2546bc80
 - ulSerialNumber = 2343
 - ulPriority = 250
 - OpType = DS_REPL_OP_TYPE_SYNC
 - ulOptions = 524291
 - pszNamingContext = DC=contoso,DC=com
 - pszDsaDN = CN=NTDS Settings,CN=DC2,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - pszDsaAddress = 12626d52-1da7-4a40-a490-987c0880c3fe._msdcs.contoso.com

- uuidNamingContextObjGuid = 8f3cea57-61ff-46cb-aa17-6c1683c33020
- uuidDsaObjGuid = 12626d52-1da7-4a40-a490-987c0880c3fe

4.1.13.4.9.4 Final State

The final state is the same as the [initial state \(section 4.1.13.4.9.1\)](#); there is no change.

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: 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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE, ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.

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: Unused. 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 2.2.

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 disabling both originating and replicated updates to the AD LDS DC.
[<31>](#)

```
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

ValidatedDRSInput(hDrs, 25)

pmsgOut^.V1.dwOpError := 0

if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
if pmsgIn = null then
    return ERROR_INVALID_PARAMETER
endif
if pmsgIn^.V1.dwReserved ≠ 0 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_DS_DRA_ACCESS_DENIED
endif

dc.fEnableUpdates := FALSE

if ret = ERROR_SUCCESS then
    ret := Mark database as read-only
endif
pmsgOut^.V1.dwOpError := ret
pdwMsgOutVersion^ := 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_MOVEREREPLY* 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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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.

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. [<32>](#)

```
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: Unused. MUST be 0 and ignored.

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.

```
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: Unused. MUST be 0 and ignored.

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.

4.1.15.1.5 DRS_MSG_MOVEREPLY_V1

The **DRS_MSG_MOVEREPLY_V1** structure defines a response message received from the [IDL DRSInterDomainMove](#) method. This response version is obsolete. [<33>](#)


```
typedef struct {
    ENTINF** ppResult;
    SCHEMA_PREFIX_TABLE PrefixTable;
    ULONG* pError;
} DRS_MSG_MOVEREPLY_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 non-zero if a fatal error occurred.

4.1.15.1.6 DRS_MSG_MOVEREPLY_V2

The **DRS_MSG_MOVEREPLY_V2** structure defines a response message received from the [IDL DRSInterDomainMove](#) method.

```
typedef struct {
    ULONG win32Error;
    [unique] DSNAME* pAddedName;
} DRS_MSG_MOVEREPLY_V2;
```

win32Error: 0 if successful, or non-zero 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 **non-replicated 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 2.5.3 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 done by the "source" DC to 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

ValidateDRSInput(hDrs, 10)

pdwOutVersion^ := 2
msgOut^.V2.win32error := ERROR_DS_GENERIC_ERROR
msgOut^.V2.pAddedName := null
if dwInVersion ≠ 2 then
    return ERROR_DS_DRA_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

/* 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(O, prefixTable)
    if AttrIsBacklink(attribute) or AttrIsNonReplicated(attribute) or
    AttrIsConstructed(attribute) then
        ENTINF_SetValue(O, attribute, null, prefixTable)
    end
end

```

```

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(O, attribute, null, prefixTable)
endif
endifor

if ENTINF_GetValue(O, userAccountControl, prefixTable) ≠ null then
    /* Reset lockout bit. */
    ENTINF_SetValue(O, userAccountControl,
        ENTINF_GetValue(O, userAccountControl) - {ADS_UF_LOCKOUT},
        prefixTable)
endif

if ENTINF_GetValue(O, pwdLastSet, prefixTable) ≠ null and
    ENTINF_GetValue(O, pwdLastSet, prefixTable) ≠ 0 then
    /* If pwdLastSet is set to non-zero, then change it to -1. */
    ENTINF_SetValue(O, pwdLastSet, (LONGLONG)-1, prefixTable)
endif

/* Append objectSid to sidHistory. */
ENTINF_SetValue(O, sidHistory,
    ENTINF_GetValue(O, sidHistory, prefixTable)
    + {ENTINF_GetValue(O, objectSid, prefixTable)})

/* Compute the new proxiedObjectName value. */
if ENTINF_GetValue(O, proxiedObjectName, prefixTable) ≠ null and
    GetProxyType(ENTINF_GetValue(O, 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(O, proxiedObjectName,
        prefixTable))+1
else
    /* No valid proxiedObjectName, so start a new one. */
    proxyEpoch := 1
endif

/* Stamp the new proxiedObjectName value into ENTINF. */
ENTINF_SetValue(O,
    proxiedObjectName,
    MakeProxyValue(msgIn.pSrcNC^,
        PROXY_TYPE_MOVED_OBJECT,
        proxyEpoch),
    prefixTable)

if existingObj ≠ null then
    /* Purge existing object, it is about to be overwritten. */
    Expunge(existingObj)

```

```

endif

ImpersonateAuthorizationInfo(clientCreds)
O.pName := msgIn.pDstName
dwErr :=
    PerformAddOperation(
        O,
        msgOut^.V2.pAddedName^,
        AbstractPTFromConcretePT(msgIn.PrefixTable),
        TRUE)

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: 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, or a Windows error code if a failure occurs.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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 number of items in the **rgpszToSites** array (the count of "to" sites).

rgpszToSites: The RDNs of the [site](#) objects of the "to" sites.

dwFlags: Unused. MUST be 0 and ignored.

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_SITESREPLY_ELEMENT_V1* rgCostInfo;
    DWORD dwFlags;
} DRS_MSG_QUERY_SITESREPLY_V1;

```

cToSites: The number of items in the **rgCostInfo** array.

rgCostInfo: The sequence of computed site costs, in the same order as the **rgpszToSites** field in the request message.

dwFlags: Unused. MUST be 0 and ignored.

4.1.16.1.5 DRS_MSG_QUERY_SITESREPLY_ELEMENT_V1

The **DRS_MSG_QUERY_SITESREPLY_ELEMENT_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

```

Informative summary of behavior: The ValidateSiteRDN procedure returns 0 if s is a valid RDN for a [site](#) object, and returns an appropriate error otherwise. A valid RDN has the following characteristics:

- Is not null.
- Does not have 0 length.
- Does not have a length greater than 64.
- Contains no occurrences of the equal sign (=) or comma (,).

```

if s = null or s.length = 0 then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
if s.Length > 64 then
    return ERROR_DS_NAME_TOO_LONG
endif
if s contains (=) or s contains (,) then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
return 0

```

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. [<34>](#)

```
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)

msgIn: DRS_MSG_QUERY_SITESREQ_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
ul : ULONG

ValidateDRSInput(hDrs, 24)

pdwOutVersion^ := 1
pmsgOut^.V1.cToSites := 0
pmsgOut^.V1.rgCostInfo := null
pmsgOut^.V1.dwFlags := 0

/* Perform input validation,
 * initialize siteContainer, ipObject, fromSite. */
if dwInVersion ≠ 1 then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
ul := ValidateSiteRDN(msgIn.pwszFromSite)
if 0 ≠ ul then
    return ul
endif
if msgIn.cToSites > 0 and msgIn.rgszToSites = null then
    return ERROR_INVALID_PARAMETER
endif
for i := 0 to msgIn.cToSites - 1
    ul := ValidateSiteRDN(msgIn.rgszToSites[i])
    if 0 ≠ ul then
        return ul
    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.pwszFromSite
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, "IP Transport Container", for
 * the definition of the NTDSTRANSOPT_OPT_BRIDGES_REQUIRED option. */
if NTDSTRANSOPT_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
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
else
/* Fully transitive network, ignore siteLinkBridge objects. */
aSetC := MinWeightPath(vSet, aSet)
endif

/* Construct result message. */
pdwOutVersion^ := 1
pmsgOut^.V1.cToSites := msgIn.cToSites
pmsgOut^.V1.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^.V1.rgCostInfo[i].dwErrorCode := ERROR_DS_OBJ_NOT_FOUND
        pmsgOut^.V1.rgCostInfo[i].dwCost := 0xffffffff
    else
        min := select one t from aSetC where
            t.initial = fromSite and t.final = toSite
        if min != null then
            pmsgOut^.V1.rgCostInfo[i].dwErrorCode := 0
            pmsgOut^.V1.rgCostInfo[i].dwCost := min.cost
        else
            pmsgOut^.V1.rgCostInfo[i].dwErrorCode := 0
            pmsgOut^.V1.rgCostInfo[i].dwCost := 0xffffffff
        endif
    endif
endfor

```



```

endif
endfor

return 0

```

4.1.16.4 Examples of IDL_DRSQuerySitesByCost Method

4.1.16.4.1 Nontransitive Communication Using siteLinkBridge

Determines the nontransitive communication cost "from" site DC1 "to" sites DC2, DC3, and DC4. A site graph is displayed in the following figure.

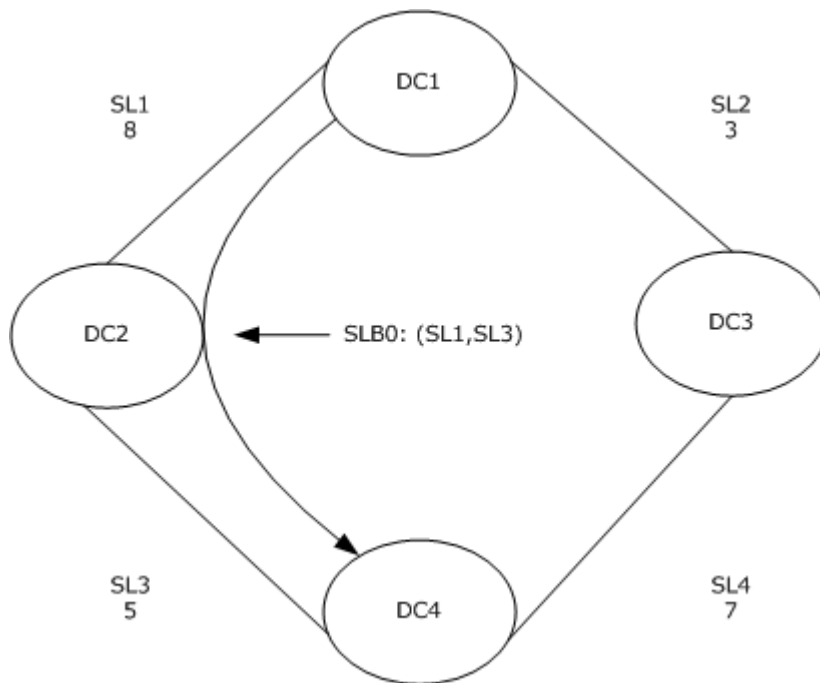


Figure 2: Site graph for a nontransitive network

VERTEX	ARC	ARC WEIGHT
DC1	SL1 SL2 SLB0 (SL1, SL3)	8 3 13
DC2	SL1 SL3	8 5
DC3	SL2 SL4	3 7
DC4	SL3 SL4 SLB0 (SL1, SL3)	7 13

4.1.16.4.1.1 Initial State

Querying the site object for domain NC CONTOSO.COM by performing an LDAP search with Base DN "CN=Configuration,DC=contoso,DC=com".

- ldap_search_s(Id, "CN=Configuration,DC=contoso,DC=com", 2, " (objectclass=site)", attrList, 0, &msg)
- Result <0>: (null)
- Matched DNs:
- Getting 4 entries:
- >> Dn: CN=DC1,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - 1> cn: DC1;
 - 1> distinguishedName: CN=DC1,CN=Sites,CN=Configuration,DC=contoso,DC=com;
 - 1> instanceType: 0x4 = (WRITE);
 - 1> location: DC1;
 - 1> name: DC1;
 - 1> objectCategory: CN=Site,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; site;
 - 1> objectGUID: 3540d101-be2d-4630-b75e-1343c2a39dc8;
 - 1> showInAdvancedViewOnly: TRUE;
 - 1> systemFlags: 0x42000000 = (CONFIG_ALLOW_RENAME | DISALLOW_MOVE_ON_DELETE);
 - 1> uSNChanged: 41007;
 - 1> uSNCreated: 36885;
 - 1> whenChanged: 06/08/2010 19:04:05 Pacific Standard Time;
 - 1> whenCreated: 06/08/2010 13:53:19 Pacific Standard Time;
- >> Dn: CN=DC2,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - 1> cn: DC2;
 - 1> distinguishedName: CN=DC2,CN=Sites,CN=Configuration,DC=contoso,DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> location: DC2;
 - 1> name: DC2
 - 1> objectCategory: CN=Site,CN=Schema,CN=Configuration,DC=contoso,DC=com;

- 2> objectClass: top; site;
- 1> objectGUID: 7dd0525e-f00a-4c1d-9eec-d6df02625a59;
- 1> showInAdvancedViewOnly: TRUE;
- 1> systemFlags: 0x42000000 = (CONFIG_ALLOW_RENAME | DISALLOW_MOVE_ON_DELETE);
- 1> uSNChanged: 40991;
- 1> uSNCreated: 40991
- 1> whenChanged: 06/08/2010 18:39:43 Pacific Standard Time;
- 1> whenCreated: 06/08/2010 18:39:43 Pacific Standard Time;
- >> Dn: CN=DC3,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - 1> cn: DC3;
 - 1> distinguishedName: CN=DC3,CN=Sites,CN=Configuration,DC=contoso,DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> location: DC3;
 - 1> name: DC3;
 - 1> objectCategory: CN=Site,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; site;
 - 1> objectGUID: dbdff472-a414-44c2-8206-a619e5eee583;
 - 1> showInAdvancedViewOnly: TRUE;
 - 1> systemFlags: 0x42000000 = (CONFIG_ALLOW_RENAME | DISALLOW_MOVE_ON_DELETE);
 - 1> uSNChanged: 40997;
 - 1> uSNCreated: 40997;
 - 1> whenChanged: 06/08/2010 18:53:31 Pacific Standard Time;
 - 1> whenCreated: 06/08/2010 18:53:31 Pacific Standard Time
- >> Dn: CN=DC4,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - 1> cn: DC4;
 - 1> distinguishedName: CN=DC4,CN=Sites,CN=Configuration,DC=contoso,DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> location: DC4;

- 1> name: DC4;
- 1> objectCategory: CN=Site,CN=Schema,CN=Configuration,DC=contoso,DC=com;
- 2> objectClass: top; site;
- 1> objectGUID: c15325f5-881b-417a-80cf-8e3530885613
- 1> showInAdvancedViewOnly: TRUE;
- 1> systemFlags: 0x42000000 = (CONFIG_ALLOW_RENAME | DISALLOW_MOVE_ON_DELETE);
- 1> uSNChanged: 41002;
- 1> uSNCreated: 41002;
- 1> whenChanged: 06/08/2010 18:59:28 Pacific Standard Time;
- 1> whenCreated: 06/08/2010 18:59:28 Pacific Standard Time;

Querying the siteLink object for domain NC CONTOSO.COM by performing an LDAP search with Base DN "CN=Configuration,DC=contoso,DC=com".

- ldap_search_s(Id, "CN=Configuration,DC=contoso,DC=com", 2, "(objectclass=sitelink)", attrList, 0, &msg)
- Result <0>: (null)
- Matched DNs:
- Getting 4 entries:
- >> Dn: CN=SL1, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com
 - 1> cn: SL1;
 - 1> cost: 8;
 - 1> distinguishedName: CN=SL1, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> instanceType: 0x4 = (WRITE);
 - 1> name: SL1;
 - 1> objectCategory: CN=Site-Link,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; siteLink;
 - 1> objectGUID: bd4ba671-90fb-4f4b-ab5d-76c9451d300c;
 - 1> replInterval: 180;
 - 2> siteList : CN=DC2,CN=Sites, CN=Configuration, DC=contoso, DC=com; CN=DC1, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> systemFlags: 0x40000000 = (CONFIG_ALLOW_RENAME);

- 1> uSNChanged: 41010;
- 1> uSNCreated: 36896;
- 1> whenChanged: 06/08/2010 19:04:37 Pacific Standard Time;
- 1> whenCreated: 06/08/2010 14:01:17 Pacific Standard Time;
- >> Dn: CN=SL2, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com
 - 1> cn: SL2;
 - 1> cost: 3;
 - 1> distinguishedName: CN=SL2, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> name: SL2;
 - 1> objectCategory: CN=Site-Link,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; siteLink;
 - 1> objectGUID: a8906f5f-0c46-4276-87c6-34e60c6c0d63;
 - 1> replInterval: 180;
 - 2> siteList: CN=DC3, CN=Sites, CN=Configuration, DC=contoso, DC=com; CN=DC1, CN=Sites, CN=Configuration,DC=contoso,DC=com;
 - 1> systemFlags: 0x40000000 = (CONFIG_ALLOW_RENAME);
 - 1> uSNChanged: 41014;
 - 1> uSNCreated: 41014;
 - 1> whenChanged: 06/08/2010 19:05:29 Pacific Standard Time;
 - 1> whenCreated: 06/08/2010 19:05:29 Pacific Standard Time;
- >> Dn: CN=SL3, CN=IP, CN=Intersite Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com
 - 1> cn: SL3;
 - 1> cost: 5;
 - 1> distinguishedName: CN=SL3, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);

- 1> name: SL3;
- 1> objectCategory: CN=Site-Link, CN=Schema, CN=Configuration, DC=contoso, DC=com;
- 2> objectClass: top; siteLink;
- 1> objectGUID: 33f2a214-bea7-4061-8ecf-eca598837bc3;
- 1> replInterval: 180;
- 2> siteList: CN=DC4,CN=Sites, CN=Configuration, DC=contoso, DC=com; CN=DC2, CN=Sites, CN=Configuration, DC=contoso, DC=com;
- 1> systemFlags: 0x40000000 = (CONFIG_ALLOW_RENAME);
- 1> uSNChanged: 41017;
- 1> uSNCreated: 41017;
- 1> whenChanged: 06/08/2010 19:05:51 Pacific Standard Time;
- 1> whenCreated: 06/08/2010 19:05:51 Pacific Standard Time;
- >> Dn: CN=SL4, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com
 - 1> cn: SL4;
 - 1> cost: 7;
 - 1> distinguishedName: CN=SL4, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> name: SL4;
 - 1> objectCategory: CN=Site-Link,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; siteLink;
 - 1> objectGUID: 3c3e2aa6-03b3-4aab-a0b2-a689a7636619;
 - 1> replInterval: 180;
 - 2> siteList: CN=DC4, CN=Sites, CN=Configuration, DC=contoso, DC=com; CN=DC3, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> systemFlags: 0x40000000 = (CONFIG_ALLOW_RENAME);
 - 1> uSNChanged: 41020;
 - 1> uSNCreated: 41020;
 - 1> whenChanged: 06/08/2010 19:06:13 Pacific Standard Time;
 - 1> whenCreated: 06/08/2010 19:06:13 Pacific Standard Time;

Querying the siteLinkBridge object for domain NC CONTOSO.COM by performing an LDAP search with Base DN "CN=Configuration,DC=contoso,DC=com".

- ldap_search_s(Id, "CN=Configuration,DC=contoso,DC=com", 2, "(objectclass=sitelinkbridge)", attrList, 0, &msg)
- Result <0>: (null)
- Matched DN's:
- Getting 1 entry:
- >> Dn: CN=SLB0, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com
 - 1> cn: SLB0;
 - 1> distinguishedName: CN=SLB0, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> dSCorePropagationData (2): 06/08/2010 17:06:09 Pacific Standard Time; 0x1 = (NEW_SD);
 - 1> instanceType: 0x4 = (WRITE);
 - 1> name: SLB0;
 - 1> objectCategory: CN=Site-Link-Bridge,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; siteLinkBridge;
 - 1> objectGUID: 6ed39e2c-0bb4-4fe7-9cb1-5b4e82d1a5e2;
 - 2> siteLinkList: CN=SL1, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso,DC=com; CN=SL3, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> systemFlags: 0x40000000 = (CONFIG_ALLOW_RENAME);
 - 1> uSNChanged: 36899;
 - 1> uSNCreated: 36899;
 - 1> whenChanged: 06/08/2010 14:05:25 Pacific Standard Time;
 - 1> whenCreated: 06/08/2010 14:05:25 Pacific Standard Time;

4.1.16.4.1.2 Client Request

A client invokes the [IDL DRSQuerySitesByCost](#) method against Contoso with the following parameters ([DRS_HANDLE](#) to DC1 is omitted):

- InVersion = 1
- pmsgIn = DRS_MSG_QUERY_SITESREQ_V1
 - pwszFromSite = "DC1"

- cToSites =3
- rgpszToSites = {"DC2", "DC3", "DC4"}
- dwFlags =0

4.1.16.4.1.3 Server Response

- pdwOutVersion^ = 1
- pmsgOut = DRS_MSG_QUERY_SITES_REPLY_V1
 - cToSites =3
 - rgCostInfo[0]: DRS_MSG_QUERY_SITES_REPLY_ELEMENT_V1
 - dwErrorCode =0
 - dwCost: =8
 - rgCostInfo[1]: DRS_MSG_QUERY_SITES_REPLY_ELEMENT_V
 - dwErrorCode =0
 - dwCost: =3
 - rgCostInfo[2]: DRS_MSG_QUERY_SITES_REPLY_ELEMENT_V1
 - dwErrorCode =0
 - dwCost: =13
- dwFlags =0

4.1.16.4.1.4 Final State

The final state is the same as the initial state; there is no change.

4.1.16.4.2 Transitive Communication

Determines the transitive communication cost from site DC1 to sites DC2, DC3, and DC4. A site graph is displayed in the following figure.

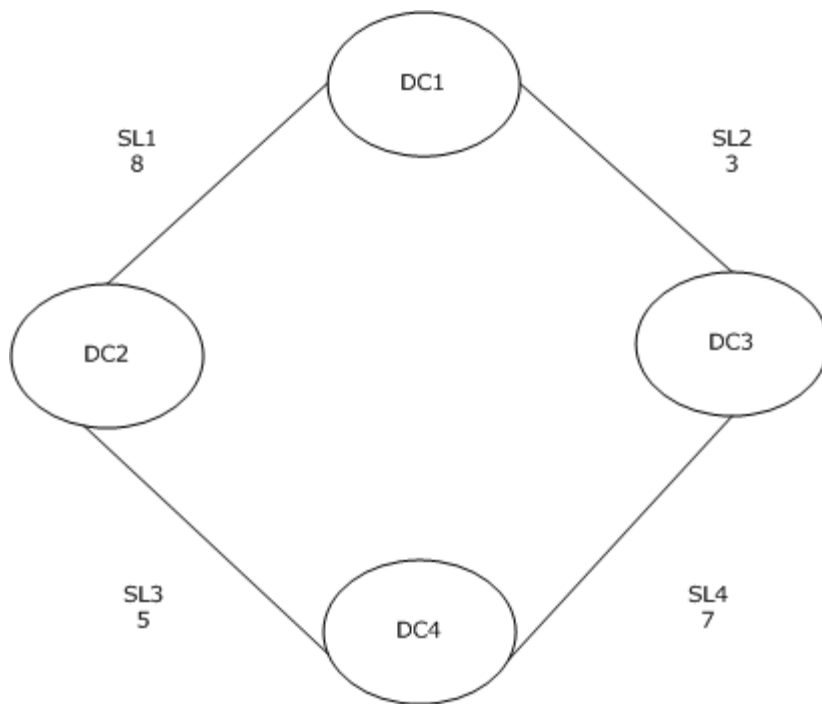


Figure 3: Site graph for a transitive network

VERTEX	ARC	ARC WEIGHT
DC	SL1 SL2	8 3
DC2	SL1 SL3	8 5
DC3	SL2 SL4	3 7
DC4	SL3 SL4	5 7

4.1.16.4.2.1 Initial State

Querying the site object for domain NC CONTOSO.COM by performing an LDAP search with Base DN "CN=Configuration,DC=contoso,DC=com".

- `ldap_search_s(ldap, "CN=Configuration,DC=contoso,DC=com", 2, " (objectclass=site)", attrList, 0, &msg)`
- Result <0>: (null)
- Matched DN;
- Getting 4 entries;

- >> Dn: CN=DC1,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - 1> cn: DC1;
 - 1> distinguishedName: CN=DC1,CN=Sites,CN=Configuration,DC=contoso,DC=com;
 - 1> instanceType: 0x4 = (WRITE);
 - 1> location: DC1;
 - 1> name: DC1;
 - 1> objectCategory: CN=Site,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; site;
 - 1> objectGUID: 3540d101-be2d-4630-b75e-1343c2a39dc8;
 - 1> showInAdvancedViewOnly: TRUE;
 - 1> systemFlags: 0x42000000 = (CONFIG_ALLOW_RENAME | DISALLOW_MOVE_ON_DELETE);
 - 1> uSNChanged: 41007;
 - 1> uSNCreated: 36885;
 - 1> whenChanged: 06/08/2010 19:04:05 Pacific Standard Time;
 - 1> whenCreated: 06/08/2010 13:53:19 Pacific Standard Time;
- >> Dn: CN=DC2,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - 1> cn: DC2;
 - 1> distinguishedName: CN=DC2,CN=Sites,CN=Configuration,DC=contoso,DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> location: DC2;
 - 1> name: DC2;
 - 1> objectCategory: CN=Site,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; site;
 - 1> objectGUID: 7dd0525e-f00a-4c1d-9eec-d6df02625a59;
 - 1> showInAdvancedViewOnly: TRUE;
 - 1> systemFlags: 0x42000000 = (CONFIG_ALLOW_RENAME | DISALLOW_MOVE_ON_DELETE);
 - 1> uSNChanged: 40991;
 - 1> uSNCreated: 40991;
 - 1> whenChanged: 06/08/2010 18:39:43 Pacific Standard Time;

- 1> whenCreated: 06/08/2010 18:39:43 Pacific Standard Time;
- >> Dn: CN=DC3,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - 1> cn: DC3;
 - 1> distinguishedName: CN=DC3,CN=Sites,CN=Configuration,DC=contoso,DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> location: DC3;
 - 1> name: DC3;
 - 1> objectCategory: CN=Site,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; site;
 - 1> objectGUID: dbdff472-a414-44c2-8206-a619e5eee583;
 - 1> showInAdvancedViewOnly: TRUE;
 - 1> systemFlags: 0x42000000 = (CONFIG_ALLOW_RENAME | DISALLOW_MOVE_ON_DELETE);
 - 1> uSNChanged: 40997;
 - 1> uSNCreated: 40997;
 - 1> whenChanged: 06/08/2010 18:53:31 Pacific Standard Time;
 - 1> whenCreated: 06/08/2010 18:53:31 Pacific Standard Time;
- >> Dn: CN=DC4,CN=Sites,CN=Configuration,DC=contoso,DC=com
 - > cn: DC4;
 - 1> distinguishedName: CN=DC4,CN=Sites,CN=Configuration,DC=contoso,DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> location: DC4;
 - 1> name: DC4;
 - 1> objectCategory: CN=Site,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; site;
 - 1> objectGUID: c15325f5-881b-417a-80cf-8e3530885613;
 - 1> showInAdvancedViewOnly: TRUE;
 - 1> systemFlags: 0x42000000 = (CONFIG_ALLOW_RENAME | DISALLOW_MOVE_ON_DELETE);
 - 1> uSNChanged: 41002l

- 1> uSNCreated: 41002;
- 1> whenChanged: 06/08/2010 18:59:28 Pacific Standard Time;
- 1> whenCreated: 06/08/2010 18:59:28 Pacific Standard Time;

Querying the siteLink object for domain NC CONTOSO.COM by performing an LDAP search with Base DN "CN=Configuration,DC=contoso,DC=com".

- ldap_search_s(ld, "CN=Configuration,DC=contoso,DC=com", 2, "(objectclass=sitelink)", attrList, 0, &msg)
- Result <0>: (null)
- Matched DNs;
- Getting 4 entries:
- >> Dn: CN=SL1, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com
 - 1> cn: SL1;
 - 1> cost: 8;
 - 1> distinguishedName: CN=SL1, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> instanceType: 0x4 = (WRITE);
 - 1> name: SL1;
 - 1> objectCategory: CN=Site-Link,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; siteLink;
 - 1> objectGUID: bd4ba671-90fb-4f4b-ab5d-76c9451d300c;
 - 1> replInterval: 180;
 - 2> siteList : CN=DC2,CN=Sites, CN=Configuration, DC=contoso, DC=com; CN=DC1, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> systemFlags: 0x40000000 = (CONFIG_ALLOW_RENAME);
 - 1> uSNChanged: 41010;
 - 1> uSNCreated: 36896;
 - 1> whenChanged: 06/08/2010 19:04:37 Pacific Standard Time;
 - 1> whenCreated: 06/08/2010 14:01:17 Pacific Standard Time;
- >> Dn: CN=SL2, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> cn: SL2;
 - 1> cost: 3;

- 1> distinguishedName: CN=SL2, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
- 1> dSCorePropagationData: 0x0 = ();
- 1> instanceType: 0x4 = (WRITE);
- 1> name: SL2;
- 1> objectCategory: CN=Site-Link,CN=Schema,CN=Configuration,DC=contoso,DC=com;
- 2> objectClass: top; siteLink;
- 1> objectGUID: a8906f5f-0c46-4276-87c6-34e60c6c0d63;
- 1> replInterval: 180;
- 2> siteList: CN=DC3, CN=Sites, CN=Configuration, DC=contoso, DC=com; CN=DC1, CN=Sites, CN=Configuration,DC=contoso,DC=com;
- 1> systemFlags: 0x40000000 = (CONFIG_ALLOW_RENAME);
- 1> uSNChanged: 41014;
- 1> uSNCreated: 41014;
- 1> whenChanged: 06/08/2010 19:05:29 Pacific Standard Time;
- 1> whenCreated: 06/08/2010 19:05:29 Pacific Standard Time;
- >> Dn: CN=SL3, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com
 - 1> cn: SL3;
 - 1> cost: 5;
 - 1> distinguishedName: CN=SL3, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> name: SL3;
 - 1> objectCategory: CN=Site-Link, CN=Schema, CN=Configuration, DC=contoso, DC=com;
 - 2> objectClass: top; siteLink;
 - 1> objectGUID: 33f2a214-bea7-4061-8ecf-eca598837bc3;
 - 1> replInterval: 180;
 - 2> siteList: CN=DC4,CN=Sites, CN=Configuration, DC=contoso, DC=com; CN=DC2, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> systemFlags: 0x40000000 = (CONFIG_ALLOW_RENAME);

- 1> uSNChanged: 41017;
- 1> uSNCreated: 41017;
- 1> whenChanged: 06/08/2010 19:05:51 Pacific Standard Time;
- 1> whenCreated: 06/08/2010 19:05:51 Pacific Standard Time;
- >> Dn: CN=SL4, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com
 - 1> cn: SL4;
 - 1> cost: 7;
 - 1> distinguishedName: CN=SL4, CN=IP, CN=Inter-Site Transports, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> dSCorePropagationData: 0x0 = ();
 - 1> instanceType: 0x4 = (WRITE);
 - 1> name: SL4;
 - 1> objectCategory: CN=Site-Link,CN=Schema,CN=Configuration,DC=contoso,DC=com;
 - 2> objectClass: top; siteLink;
 - 1> objectGUID: 3c3e2aa6-03b3-4aab-a0b2-a689a7636619;
 - 1> replInterval: 180;
 - 2> siteList: CN=DC4, CN=Sites, CN=Configuration, DC=contoso, DC=com; CN=DC3, CN=Sites, CN=Configuration, DC=contoso, DC=com;
 - 1> systemFlags: 0x40000000 = (CONFIG_ALLOW_RENAME);
 - 1> uSNChanged: 41020;
 - 1> uSNCreated: 41020;
 - 1> whenChanged: 06/08/2010 19:06:13 Pacific Standard Time;
 - 1> whenCreated: 06/08/2010 19:06:13 Pacific Standard Time;

4.1.16.4.2.2 Client Request

A client invokes the [IDL DRSQuerySitesByCost](#) method against Contoso with the following parameters ([DRS_HANDLE](#) to DC1 is omitted):

- InVersion = 1
- pmsgIn = DRS_MSG_QUERY_SITESREQ_V1
 - pwszFromSite = "DC1"
 - cToSites = 3
 - rgszToSites = {"DC2", "DC3", "DC4"}

- dwFlags = 0

4.1.16.4.2.3 Server Response

- pdwOutVersion^ = 1
- pmsgOut = DRS_MSG_QUERY_SITES_REPLY_V1
 - cToSites = 3
 - rgCostInfo[0]: DRS_MSG_QUERY_SITES_REPLY_ELEMENT_V1
 - dwErrorCode = 0
 - dwCost: = 8
 - rgCostInfo[1]: DRS_MSG_QUERY_SITES_REPLY_ELEMENT_V1
 - dwErrorCode = 0
 - dwCost: = 3
 - rgCostInfo[2]: DRS_MSG_QUERY_SITES_REPLY_ELEMENT_V1
 - dwErrorCode = 0
 - dwCost: = 10
- dwFlags = 0

4.1.16.4.2.4 Final State

The final state is the same as the initial state; there is no change.

4.1.17 IDL_DRSRemoveDsDomain (Opnum 15)

The **IDL_DRSRemoveDsDomain** method removes the representation (also known as 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, because this is the only version supported.

pmsgIn: A pointer to the request message.

pdwOutVersion: A pointer to the version of the response message. The value must be 1 because that is the only version supported.

pmsgOut: A pointer to the response message.

Return Values: 0 if successful, or a Windows error code if a failure occurs.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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: Unused. MUST be 0 and ignored.

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_DRSSAddEntry method when IDL_DRSSAddEntry 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  
rt: ULONG
```

```

ValidatedDRSInput(hDrs, 15)

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

/* 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, so exit
     * with an error. */
    if not HasNCReplicated(ConfigNC()) then
        return ERROR_DS_ROLE_NOT_VERIFIED;
    endif
endif

if (not AccessCheckObject(cr, RIGHT_DELETE)) and
    (not AccessCheckObject(cr.parent, RIGHT_DS_DELETE_CHILD)) then
    return ERROR_ACCESS_DENIED
endif

rt:= RemoveObj(cr,false)

```

```

if rt != 0 then
    return rt
endif

DelSubRef(cr!ncName)
return 0

```

4.1.18 IDL_DRSRemoveDsServer (Opnum 14)

The **IDL_DRSRemoveDsServer** method removes the representation (also known as 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: A pointer to the request message.

pdwOutVersion: A pointer to the version of the response message. The value must be 1 because that is the only version supported.

pmsgOut: A pointer to the response message.

Return Values: 0 if successful or a Windows error code if a failure occurs.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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.fLastDCInDomain.

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. Otherwise, false.

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 method also computes whether the DC is the last replica of its default domain NC.

The behavior of this method has two variants. If pmsgIn^.V1.fCommit is false, the method is read-only with regard to abstract state; that is, it does not make any changes to the directory contents. In this mode, the main purpose of the method is to compute pmsgOut^.V1.fLastDcInDomain (and so there is little point to calling the method in this mode without setting pmsgIn^.V1.DomainDN). For example, prior to removing the DC's metadata, a client application might try 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 method 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_DRSSAddEntry method when IDL_DRSSAddEntry 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
server: DSName
ntdsdsa: DSName
otherNtdsdsa: DSName
spnsToRemove: set of unicodestring
computerDn: unicodestring
computer: DSName
objectsToDelete: set of DSName
rt: ULONG
ValidateDRSInput(hDrs, 14)

serverDn := pmsgIn^.V1.ServerDN
domainDn := pmsgIn^.V1.DomainDN

pdwOutVersion^ := 1
```

```

pmsgOut^.V1.fLastDcInDomain = false

/* 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 cannot be empty. */
if domainDn = "" then
    return ERROR_INVALID_PARAMETER
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

ntdsdsa := DescendantObject([dn: serverDn], "CN=NTDS Settings,")
if ntdsdsa = null then
    return ERROR_DS_CANT_FIND_DSA_OBJ
endif

/* Perform the actual DC metadata removal. */

/* Locate the computer object for the DC's account. */
server := ntdsdsa!parent
computerDn := server!serverReference
computer := null
if computerDn # null then
    computer := GetDSNameFromDN(computerDn)
endif

/* Remove the subtree of objects rooted at the DC's ntdsDsa object.*/

if not AccessCheckObject(ntdsdsa, RIGHT_DS_DELETE_TREE) then
    return ERROR_ACCESS_DENIED
endif

rt := RemoveObj(ntdsdsa,true)
if rt # 0 then

```

```

    return rt
endif

/* 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_DELETE)) and
            (not AccessCheckObject(r.parent, RIGHT_DS_DELETE_CHILD)) then
            return ERROR_ACCESS_DENIED
        endif

        RemoveObj(r, false)
    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: A pointer to the request message.

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

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE,

ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.

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.

V2: The version 2 request (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.

```
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 used 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.

```
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 used 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_NONGC_RO_REP and DRS_SPECIAL_SECRET_PROCESSING in ulOptions if the specified NC replica is a read-only full replica on a read-only DC. The server adds a value to [repsFrom](#), and the value has replicaFlags derived from ulOptions (see below), serverAddress equal to pszSourceDsaAddress (pszDsaSrc if V1), and 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. */
hDrsSrc: DRS_HANDLE

ValidateDRSInput(hDrs, 5)

/* Validate the version */
if dwVersion != 1 and dwVersion != 2 then
    return ERROR_DS_DRA_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_DS_DRA_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_NC
endif

if options - {DRS_ASYNC_OP, DRS_CRITICAL_ONLY, DRS_ASYNC_REP,
    DRS_WRIT_REP, DRS_INIT_SYNC, DRS_PER_SYNC, DRS_MAIL_REP,
    DRS_NONGC_RO_REP, DRS_SPECIAL_SECRET_PROCESSING, DRS_DISABLE_AUTO_SYNC,
    DRS_DISABLE_PERIODIC_SYNC, DRS_USE_COMPRESSION, DRS_NEVER_NOTIFY,
    DRS_TWOWAY_SYNC} ≠ {} then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif

if AmIRODC() and DRS_WRIT_REP in options then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif

if AmIRODC() and DRS_MAIL_REP in options then
    return ERROR_DS_DRA_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_INSTANCE_TYPE
    endif
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 DRS_ASYNC_REP in options then
    if msgIn.pSourceDsaDN = null
        or not ObjExists(msgIn.pSourceDsaDN^)
        then
            return ERROR_DS_DRA_INVALID_PARAMETER
        endif
    endif
endif

if DRS_MAIL_REP in options then
    if msgIn.pTransportDN = null
        or not ObjExists(msgIn.pTransportDN^)
        then
            return ERROR_DS_DRA_INVALID_PARAMETER
        endif
    endif
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_NONGC_RO_REP,
    DRS_SPECIAL_SECRET_PROCESSING }
rf.schedule := msgIn.rtSchedule^
rf.serverAddress := msgIn.pszSourceDsaAddress^
rf.timeLastAttempt := current time

nc!repsFrom := nc!repsFrom + {rf}
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.uuidDsaObjDest := 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

    hDrsSrc := BindToDSA(msgIn.pSourceDsaDN)
    if hDrsSrc # null then
        ret := IDL_DRSUpdateRefs(hDrsSrc, 1, ADR(updRefs))
        UnbindFromDSA(hDrsSrc)
    endif
endif

```

```
endif
```

Perform a replication cycle as a client of IDL_DRSGetNCChanges. Call ReplicateNCRequestMsg (section 4.1.10.4.1) to form the first request and send it. If not DRS_MAIL_REP in msgIn.ulOptions, then wait for the response, process it (section 4.1.10.6), send the next request, etc. until the replication cycle is complete. If there are any failures from this replication attempt, err should be assigned an appropriate error value.

```
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: A pointer to the request message.

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

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE, ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.

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: A 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 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 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. This operation returns an error and the expunge does not occur if the [repsFrom](#) or [repsTo](#) attributes are present on the NC replica. However, if ulOptions contains DRS_REF_OK, it is permitted for [repsTo](#) to be present. 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
cr: DSName
srcDSA: DSName
hDrsSrc: DRS_HANDLE
rf: RepsFrom
msgIn: DRS_MSG_REPDEL_V1
updRefs: DRS_MSG_UPDREFS /* See IDL_DRSUpdateRefs structures. */

ValidatedDRSInput(hDrs, 6)

msgIn := pmsgDel^.V1

/* Validate the NC */
if msgIn.pNC = null then
```

```

    return ERROR_DS_DRA_INVALID_PARAMETER
endif

nc := msgIn.pNC^

if not ObjExists(nc) then
    return ERROR_DS_DRA_BAD_NC
endif

if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
    return ERROR_DS_DRA_ACCESS_DENIED
endif

options := msgIn.ulOptions

/* Any request that includes invalid options is rejected. */
if options - {DRS_ASYNC_OP, DRS_WRIT_REP, DRS_MAIL_REP, DRS_ASYNC_REP,
    DRS_IGNORE_ERROR, DRS_LOCAL_ONLY, DRS_NO_SOURCE, DRS_REF_OK} ≠ {} then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif

if DRS_NO_SOURCE in options then
    /* Expunging local copy of an NC. */

    /* Do not permit removal of nonroot or uninstantiated NCs. */
    if (IT_NC_HEAD not in nc!instanceType or
        IT_UNINSTANT in nc!instanceType) then
        return ERROR_DS_DRA_BAD_NC
    endif

    /* NC must not replicate from any other DC. */
    if (select one v from nc!repsFrom where (true)) ≠ null then
        return ERROR_DS_DRA_INVALID_PARAMETER
    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_DS_DRA_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 and pertaining to the same NC.
    * If the subtree includes a sub-ref object for a locally instantiated NC,
    * remove the IT_NC_ABOVE flag from the sub-ref object instanceType
    * attribute.

```

```

*
*/
foreach o in (select all v from subtree nc where GetObjectNC(v) = nc)
  if(IT_NC_HEAD in o!instanceType and
    IT_UNINSTANT not in o!instanceType) then
    o!instanceType = o!instanceType - {IT_NC_ABOVE}
  else
    Expunge(o)
  endif
endfor

/* If the root of the NC being expunged is a sub-ref object itself, then it
* may need to be preserved.
*/

/* Check whether there is still a crossref object for the given nc. */
cr := select one v from subtree ConfigNC()
    where v!ncName = nc and crossRef in v!objectClass

if(cr == NULL)
  if(IT_NC_ABOVE in nc!instanceType) then
    nc!instanceType = {IT_NC_ABOVE, IT_UNINSTANT, IT_NC_HEAD}
  endif
  RemoveObj(nc, false)
else
  if(IT_NC_ABOVE in nc!instanceType) then
    nc!instanceType = {IT_NC_ABOVE, IT_UNINSTANT, IT_NC_HEAD}
  else
    Expunge(nc)
  endif
endif

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_DS_DRA_INVALID_PARAMETER
  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 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 server DC
    * specified by msgIn.pszDsaSrc to remove this DC

```

```

        * from its 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
    srcDSA := GetDSNameFromNetworkAddr(msgIn.pszDsaSrc)
    hDrsSrc := BindToDSA(srcDSA)
    if hDrsSrc ≠ null then
        ret := IDL_DRSUpdateRefs(hDrsSrc, 1, ADR(updRefs))
        UnbindFromDSA(hDrsSrc)
    endif

endif
return 0
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.

```

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: 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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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;
    [ref] DSNAME* pNC;
} DRS_MSG_REPLICA_DEMOTIONREQ_V1;
```

dwFlags: Zero or more of the following bit flags, which are presented in little-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	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: Unused. MUST be zero.

T (DS_REPLICA_DEMOTE_TRY_ALL_SRCS, 0x00000001): MUST be set.

uuidHelperDest: Unused. Must be NULL GUID and ignored.

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 2.2.

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 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
hDRS: DRS_HANDLE
msgReq: DRS_MSG_GETCHGREQ_V10
msgUpd: DRS_MSG_GETCHGREPLY_V6
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 do not 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(targetDSAs[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
            /* successfully given away */
            bGivenAway := true
        endif
        UnbindFromDSA(hDRS)
    endif
    i := i + 1
endwhile
/* if execution got here, the role was given away */
return ERROR_SUCCESS

```

4.1.21.2.3 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(targetDSAs[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,
1,ADR(msgSyncReq))
    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
      msgAddReq.pSourceDsaDN := null
      msgAddReq.pTransportDN := null
      ret := IDL_DRSReplicaAdd(hDRS, 2,ADR(msgAddReq))
    endif
    UnbindFromDSA(hDRS)
    if ret = ERROR_SUCCESS then
      /* success! */
      bReplicated := true
    endif
  endif
  i := i + 1
endwhile
/* if execution got here, then the changes were successfully replicated off */
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

ValidatedDRSInput(hDrs, 26)

pdwOutVersion^ := 1
pmsgOut^.V1.dwOpError := ERROR_DS_CODE_INCONSISTENCY

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_DS_DRA_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^.V1.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: A pointer to the request message.

Return Values: 0 if successful, or a Windows error code if a failure occurs.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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: A pointer to the [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 (presented in little-endian byte order).

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

X: Unused. MUST be zero and ignored.

UF (DRS_UPDATE_FLAGS, 0x00000001): Updates the flags associated with the server.

UA (DRS_UPDATE_ADDRESS, 0x00000002): Updates the transport-specific address associated with the server.

US (DRS_UPDATE_SCHEDULE, 0x00000004): Updates the replication schedule associated with the server.

ulOptions: The **DRS_OPTIONS** flags for execution of 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 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

ValidateDRSInput(hDrs, 7)

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}
≠ {}

```

```

        or msgIn.ulOptions - {DRS_ASYNC_OP} ≠ {} then
        return ERROR_DS_DRA_INVALID_PARAMETER
    endif

    /* Validate the specified NC */
    options := msgIn.ulOptions
    nc := msgIn.pNC^

    if not ObjExists(nc) 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

    /* Find the specified repsFrom. */
    if (msgIn.uuidSourceDRA ≠ null ) then
        rf := select one v from nc!repsFrom
            where (v.uuidDsa = msgIn.uuidSourceDRA)
    else
        rf := select one v from nc!repsFrom
            where (v.serverAddress = msgIn.pszSourceDRA)
    end if

    if rf = null then
        return ERROR_DS_DRA_NO_REPLICA
    endif

    /* Update the specified repsFrom. */
    nc!repsFrom := nc!repsFrom - {rf}
    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: A pointer to the request message.

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

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE, ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.

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: A pointer to [DSName](#) of the root of an NC replica on the server.

uuidDsaSrc: The 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

ValidatedDRSInput(hDrs, 2)

/* Validate the version */
if dwVersion ≠ 1 then
    return ERROR_DS_DRA_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_DS_DRA_INVALID_PARAMETER
endif

/* Validate the specified NC. */
nc := msgIn.pNC^
if not ObjExists(nc) then
    return ERROR_DS_DRA_BAD_NC
endif

if (DRS_SYNC_BYNAME in options and msgIn.pszDsaSrc = null)
    or (not DRS_SYNC_BYNAME in options and msgIn.uuidDsaSrc = null)
    or (not DRS_SYNC_BYNAME in options and msgIn.uuidDsaSrc = NULLGUID) then
    return ERROR_DS_DRA_INVALID_PARAMETER
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.naDsa = 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 (section 4.1.10.4.1) 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
        (section 4.1.10.6), 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 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: A pointer to the request message.

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

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE, ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.

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,
    var uTDVec: UPTODATE_VECTOR_V1_EXT
): ULONG
```

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*. The procedure returns either an implementation-specific value from the client implementation of the **IDL_DRSGetReplInfo** method, or the value returned by the remote server's **IDL_DRSGetReplInfo** method.

4.1.24.2.2 ObjectExistsAtDC

```
procedure ObjectExistsAtDC(o: DSName, dsa: DSName): boolean
```

The ObjectExistsAtDC procedure checks that the object *o* exists on the DC whose [nTDSDSA](#) object has the [DSName](#) *dsa* by verifying that the DC holds an object *o'* whose [objectGUID](#) value is equal to that of object *o*. 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 there is no requirement on whether IDL_DRSReplicaVerifyObjects should or should not expunge or 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)

err: ULONG
msgIn: DRS_MSG_REPVERIFYOBJ_V1
nc, refDsa, o: DSName
uTDServer, uTDRef, uTDMerge: UPTODATE_VECTOR_V1_EXT
sPrime: set of DSName

ValidateDRSInput(hDrs, 22)

/* Perform input validation and access check */
if dwInVersion ≠ 0x1 then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
msgIn := pmsgVerify^V1
if msgIn.pNC = null then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
nc := msgIn.pNC^
if not FullReplicaExists(nc) and
    not PartialGCReplicaExists(nc) then
    return ERROR_DS_DRA_BAD_NC
endif
if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
    return ERROR_DS_DRA_ACCESS_DENIED
endif
refDsa := select one object o from subtree ConfigNC() where
    o!objectGUID = msgIn.uuidDsaSrc and nTDSDSA in o!objectClass
if refDsa = null then
    if IsAdlds() then
        return ERROR_DS_DRA_INVALID_PARAMETER
```

```

else
    return ERROR_DS_DNS_LOOKUP_FAILURE
endif
endif

/* Compute the set S' */
uTDServer := nc!replUpToDateVector
err := GetRemoteUTD(refDsa, nc, uTDRef)
if err ≠ 0 then
    return err
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 not 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: [<35>](#35)

4.1.24.4 Examples of the IDL_DRSReplicaVerifyObjects Method

A client that has bound to DC1 is removing all lingering objects on this directory server with respect to DC2.

4.1.24.4.1 Initial State

A client has bound to DC1.CONTOSO.COM using the [IDL_DRSBind](#) method and received a [DRS_HANDLE](#) to DC1.

Consider the following objects under the Users container, "CN=Users,DC=CONTOSO,DC=COM", listed by their [DSName](#):

Users at DC1	Users at DC2	Notes
<GUID=f5ef2f4b-a3db-464c-8403-b27aa00b0d5d>;<SID=S-1-5-21-1583212203-607051668-819563750-1107>;CN=Kim Akers, CN=Users,DC=CONTOSO,DC=COM	<GUID=f5ef2f4b-a3db-464c-8403-b27aa00b0d5d>;<SID=S-1-5-21-1583212203-607051668-819563750-1107>;CN=Kim Akers, CN=Users,DC=CONTOSO,DC=COM	Objects are identical.
<GUID=89430510-48eb-4e68-aeb1-98a9471f1938>;<SID=S-1-5-21-1583212203-607051668-819563750-1111>; CN=Josh Bailey,CN=Users,DC=CONTOSO,DC=COM		"Josh Bailey" was created on DC1 and has not been

Users at DC1	Users at DC2	Notes
		replicated to DC2 yet.
<GUID=833a118e-035f-4702-b67e-9e7c1ada2f57>;<SID=S-1-5-21-1583212203-607051668-819563750-1108>;CN= Eva Corets,CN=Users,DC=CONTOSO,DC=COM		"Eva Corets" is a lingering object on DC1.
<GUID=3cb4b6cf-f220-472a-bd2f-5f1399232ca6>;<SID=S-1-5-21-1583212203-607051668-819563750-1109>;CN= Jim Daly,CN=Users,DC=CONTOSO,DC=COM	<GUID=3cb4b6cf-f220-472a-bd2f-5f1399232ca6>;<SID=S-1-5-21-1583212203-607051668-819563750-1109>;CN= Jim Daly,CN=Users,DC=CONTOSO,DC=COM	The mail attribute of "Jim Daly" has been modified on DC1 but this change has not replicated to DC2 yet.
	<GUID=46c1b351-da31-49f2-8437-8d82df024972>;<SID=S-1-5-21-1583212203-607051668-819563750-1604>; CN=Ebru Ersan,CN=Users,DC=CONTOSO,DC=COM	"Ebru Ersan" was created on DC2 and has not been replicated to DC1 yet.
	<GUID=8df1f9bb-7551-46c3-b9c2-c905e9463542>;<SID=S-1-5-21-1583212203-607051668-819563750-1110>; CN= Kari Furse,CN=Users,DC=CONTOSO,DC=COM	"Kari Furse" is a lingering object on DC2.

Relevant entries of the DS_REPL_ATTR_META_DATA structure for each object listed above are also captured below to further demonstrate the differences between DC1 and DC2.

Relevant metadata entries for "CN=Kim Akers,CN=Users,DC=CONTOSO,DC=COM" at DC1:

usnLocalChange	uidLastOriginatingDsaInvocationID	usnOriginatingChange	ftimeLastOriginatingChange	dwVersion	pszAttributeNames
13964	4875e25f-11a9-4c70-abf4-5fb39529f84b	13964	5/21/2010 18:08:30	1	whenCreated

Relevant metadata entries for "CN=Josh Bailey,CN=Users,DC=CONTOSO,DC=COM" at DC1:

usnLocalChange	uidLastOriginatingDsaInvocationID	usnOriginatingChange	ftimeLastOriginatingChange	dwVersion	pszAttributeNames
14112	4875e25f-11a9-4c70-abf4-5fb39529f84b	14112	5/21/2010 19:11:09	1	whenCreated

Relevant metadata entries for "CN=Eva Corets,CN=Users,DC=CONTOSO,DC=COM" at DC1:

usnLocalC hange	uidLastOriginatingDsaI nvocationID	usnOriginatin gChange	ftimeLastOriginati ngChange	dwVer sion	pszAttribut eName
9071	4875e25f-11a9-4c70- abf4-5fb39529f84b	9071	1/15/2009 11:05:42	1	whenCreate d

Relevant metadata entries for "CN=Jim Daly,CN=Users,DC=CONTOSO,DC=COM" at DC1:

usnLocalC hange	uidLastOriginatingDsaI nvocationID	usnOriginatin gChange	ftimeLastOriginati ngChange	dwVer sion	pszAttribut eName
14085	4875e25f-11a9-4c70- abf4-5fb39529f84b	14085	5/21/2010 19:06:32	1	whenCreate d
14118	4875e25f-11a9-4c70- abf4-5fb39529f84b	14118	5/21/2010 19:12:51	1	mail

Relevant metadata entries for "CN=Kim Akers,CN=Users,DC=CONTOSO,DC=COM" at DC2:

usnLocalC hange	uidLastOriginatingDsaI nvocationID	usnOriginatin gChange	ftimeLastOriginati ngChange	dwVer sion	pszAttribut eName
12324	4875e25f-11a9-4c70- abf4-5fb39529f84b	13964	5/21/2010 18:08:30	1	whenCreate d

Relevant metadata entries for "CN=Jim Daly,CN=Users,DC=CONTOSO,DC=COM" at DC2:

usnLocalC hange	uidLastOriginatingDsaI nvocationID	usnOriginatin gChange	ftimeLastOriginati ngChange	dwVer sion	pszAttribut eName
12432	4875e25f-11a9-4c70- abf4-5fb39529f84b	14085	5/21/2010 19:06:32	1	whenCreate d

Relevant metadata entries for "CN=Ebru Ersan,CN=Users,DC=CONTOSO,DC=COM" at DC2:

usnLocalC hange	uidLastOriginatingDsaI nvocationID	usnOriginatin gChange	ftimeLastOriginati ngChange	dwVer sion	pszAttribut eName
12451	7526f625-51db-4022- 8150-59c0286efd82	12451	5/21/2010 19:19:14	1	whenCreate d

Relevant metadata entries for "CN=Kari Furse,CN=Users,DC=CONTOSO,DC=COM" at DC2:

usnLocalC hange	uidLastOriginatingDsaI nvocationID	usnOriginatin gChange	ftimeLastOriginati ngChange	dwVer sion	pszAttribut eName
441	4875e25f-11a9-4c70- abf4-5fb39529f84b	5099	11/1/2008 04:29:47	1	whenCreate d

The [UPTODATE VECTOR V1 EXT](#) structures on DC1 and DC2 are also needed for the [IDL_DRSReplicaVerifyObjects](#) method:

- On DC1:

dwVersion: 1

dwReserved1: 0

cNumCursors: 2

dwReserved2: 0

rgCursors: An array of UPTODATE_CURSOR_V1:

- First entry:

uuidDsa: 4875e25f-11a9-4c70-abf4-5fb39529f84b

usnHighPropUpdate: 14621

- Second entry:

uuidDsa: 7526f625-51db-4022-8150-59c0286efd82

usnHighPropUpdate: 12448

- On DC2:

dwVersion: 1

dwReserved1: 0

cNumCursors: 2

dwReserved2: 0

rgCursors: An array of [UPTODATE_CURSOR_V1](#):

- First entry:

uuidDsa: 4875e25f-11a9-4c70-abf4-5fb39529f84b

usnHighPropUpdate: 14107

- Second entry:

uuidDsa: 7526f625-51db-4022-8150-59c0286efd82

usnHighPropUpdate: 12992

- Finally, also relevant to **IDL_DRSReplicaVerifyObjects** is the nTDSDSA object for DC2 as seen on DC1:
 - 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: e845e047-3850-4a82-8811-a0b9250863c6;

4.1.24.4.2 Client Request

A client invokes the [IDL_DRSReplicaVerifyObjects](#) method on DC1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted):

- dwVersion: 1
- pmsgVerify:

pNC: Pointer to the [DSName](#) structure for DC=CONTOSO,DC=COM

uuidDsaSrc: e845e047-3850-4a82-8811-a0b9250863c6

ulOptions: 0

4.1.24.4.3 Server Response

The server returns a code of 0.

4.1.24.4.4 Final State

The IDL_DRSReplicaVerifyObjects method has removed all lingering objects on DC1 (but not on DC2). The following table compares the Users container on DC1 and DC2 after the IDL_DRSReplicaVerifyObjects method has been successfully returned.

Users at DC1	Users at DC2	Notes
<GUID=f5ef2f4b-a3db-464c-8403-b27aa00b0d5d>;<SID=S-1-5-21-1583212203-607051668-819563750-1107>;CN=Kim Akers, CN=Users,DC=CONTOSO,DC=COM	<GUID=f5ef2f4b-a3db-464c-8403-b27aa00b0d5d>;<SID=S-1-5-21-1583212203-607051668-819563750-1107>;CN=Kim Akers, CN=Users,DC=CONTOSO,DC=COM	Objects are identical.
GUID=89430510-48eb-4e68-aeb1-98a9471f1938>;<SID=S-1-5-21-1583212203-607051668-819563750-1111>; CN=Josh Bailey,CN=Users,DC=CONTOSO,DC=COM		"Josh Bailey" was created on DC1 and has not been replicated to DC2 yet.
		"Eva Corets" was a lingering object on DC1 and has been expunged.
<GUID=3cb4b6cf-f220-472a-bd2f-5f1399232ca6>;<SID=S-1-5-21-1583212203-607051668-819563750-1109>;CN= Jim Daly,CN=Users,DC=CONTOSO,DC=COM	<GUID=3cb4b6cf-f220-472a-bd2f-5f1399232ca6>;<SID=S-1-5-21-1583212203-607051668-819563750-1109>;CN= Jim Daly,CN=Users,DC=CONTOSO,DC=COM	The mail attribute of "Jim Daly" has been modified on DC1 but this change has not

Users at DC1	Users at DC2	Notes
		replicated to DC2 yet.
	<GUID=46c1b351-da31-49f2-8437-8d82df024972>;<SID=S-1-5-21-1583212203-607051668-819563750-1604>; CN=Ebru Ersan,CN=Users,DC=CONTOSO,DC=COM	"Ebru Ersan" was created on DC2 and has not been replicated to DC1 yet.
	<GUID=8df1f9bb-7551-46c3-b9c2-c905e9463542>;<SID=S-1-5-21-1583212203-607051668-819563750-1110>; CN= Kari Furse,CN=Users,DC=CONTOSO,DC=COM	"Kari Furse" is a lingering object on DC2.

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: A 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.

Exceptions Thrown: This method might throw the following exception beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE.

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)

ValidateDRSInput(hDrs, 1)

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: A pointer to the request message.

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

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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: A pointer to the [DSNAME](#) of the root of an NC replica on the server.

pszDsaDest: The transport-specific [NetworkAddress](#) of a DC.

uuidDsaObjDest: The 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. If these options are combined, the Delete operation is done before the Add operation; 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 both **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

ValidatedDRSInput(hDrs, 4)

if dwInVersion  $\neq$  1 then
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
msgIn := pmsgUpdRefs^.V1
options := msgIn.ulOptions

if msgIn.pNC = null or
    (msgIn.pszDsaDest = null) or
    (msgIn.uuidDsaObjDest = null) or
    (options  $\cap$  {DRS_ADD_REF, DRS_DEL_REF} = null)
    return ERROR_DS_DRA_INVALID_PARAMETER
endif
nc := msgIn.pNC^

if ((IT_WRITE in nc!instanceType)  $\neq$  (DRS_WRIT_REP in options)) or
    not ObjExists(nc) then
    return ERROR_DS_DRA_BAD_NC
endif

if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
    return ERROR_DS_DRA_ACCESS_DENIED
endif

/* Perform repsTo value add, delete, or combination of add/delete to the specified NC
replica,
* the result value is a Windows error code or 0
result := UpdateRefs(pmsgIn^.V1)
if(result  $\neq$  ERROR_SUCCESS) then
    return result
endif
```

```

/* If DRS_GETCHG_CHECK is specified, ERROR_DS_DRA_REF_NOT_FOUND and
 * ERROR_DS_DRA_REF_ALREADY_EXISTS are ignored. */
if DRS_GETCHG_CHECK in options and
(err = ERROR_DS_DRA_REF_NOT_FOUND or err = ERROR_DS_DRA_REF_ALREADY_EXISTS) then
    err := 0
endif

return err

```

4.1.26.3 Examples of the IDL_DRSUpdateRefs Method

4.1.26.3.1 Adding a repsTo Entry

This example shows how to add a new **repsTo** entry by calling [IDL_DRSUpdateRefs \(section 4.1.26\)](#) with the *DRS_ADD_REF* parameter.

4.1.26.3.1.1 Initial State

The [repsTo](#) attribute on the NC root object for domain NC CONTOSO.COM on DC1 does not contain a value:

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

Result <0>: (null)

Matched DNs:

Getting 1 entry:

```
>> Dn: DC=CONTOSO,DC=COM
```

4.1.26.3.1.2 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: Pointer to the DSNAME structure for DC=CONTOSO,DC=COM
- pszDsaDest : "5fe84f18-3765-4ca3-b895-47802a7ab74f._msdcs.CONTOSO.COM"
- uuidDsaObjDest: 5fe84f18-3765-4ca3-b895-47802a7ab74f
- ulOptions: DRS_WRIT_REP | DRS_ADD_REF

4.1.26.3.1.3 Server Response

Return code of 0.

4.1.26.3.1.4 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 entry:

```
>> Dn: DC=CONTOSO,DC=COM
```

```
1> repsTo: dwVersion = 2,
```

- V2.cb: 592, V2.cConsecutiveFailures: 0, V2.timeLastSuccess: 12924245513,
- V2.timeLastAttempt: 0, V2.ulResultLastAttempt: 0,
- V2.cbOtherDraOffset: 216,
- V2.cbOtherDra: 376, V2.ulReplicaFlags: 16, V2.rtSchedule: <ldp:skipped>,
- V2.usnvec.usnHighObjUpdate: 0, V2.usnvec.usnHighPropUpdate: 0,
- V2.uuidDsaObj: 5fe84f18-3765-4ca3-b895-47802a7ab74f
- V2.uuidInvocId: 00000000-0000-0000-0000-000000000000
- V2.uuidTransportObj: 00000000-0000-0000-0000-000000000000
- V2.cbPASDataOffset: 0
- V2~PasData: (none)
- v2~pdsa_rpc_inst
- v2.pszDSIServer 5fe84f18-3765-4ca3-b895-47802a7ab74f._msdcs.CONTOSO.COM
- v2.pszDSIAnnotation (null)
- v2.pszDSIInstance 5fe84f18-3765-4ca3-b895-47802a7ab74f._msdcs.CONTOSO.COM
- v2.pguidDSIInstance (null);

4.1.26.3.2 Replacing a repsTo Entry

This example shows how to semantically update an existing **repsTo** entry by calling [IDL DRSUpdateRefs](#) with the *DRS_ADD_REF* and *DRS_DEL_REF* parameters.

4.1.26.3.2.1 Initial State

The ldap search

```
ldap_search_s(ld, "DC=CONTOSO,DC=com", 0, "(objectclass=*)",[repsTo])
```

returns

Getting 1 entry:

>> Dn: DC=CONTOSO,DC=COM

repsTo (2): dwVersion = 2

- v2.cb: 592, v2.cConsecutive Failures: 0, v2.timeLastSuccess: 12924315918,
- V2.timeLastAttempt: 1292431591 , V2.ulResultLastAttempt:0,
- V2.cbOtherDraOffset: 216,
- V2.cbOtherDra: 376, V2.ulReplicaFlags: 16, V2.rtSchedule: <ldp:skipped> ,
- V2.usnvec.usnHighObjUpdate: 0, v2.usnvec.usnHighPropUpdate:0
- V2.pszUuidDsaObj: 5fe84f18-3765-4ca3-b895-47802a7ab74f
- V2.pszUuidInvocId: 00000000-0000-0000-0000-000000000000
- V2.pszUuidTransportObj: 00000000-0000-0000-0000-000000000000
- V2.cbPASDataOffset: 0 v2~PasData: (none)
- V2~pdsa_rpc_inst
- V2.pszDSIServer 5fe84f18-3765-4ca3-b895-47802a7ab74f._msdcs.CONTOSO.COM
- V2.pszDSIAnnotation (null)
- V2.pszDSIInstance 5fe84f18-3765-4ca3-b895-47802a7ab74f._msdcs.CONTOSO.COM
- V2.pguidDSIInstance (null)

4.1.26.3.2.2 Client Request

A client invokes the [IDL DRSUpdateRefs \(section 4.1.26\)](#) method against DC1 with the following parameters ([DRS_HANDLE](#) to DC1 omitted).

- dwVersion = 1
- pmsgUpdRefs = 0x0006fe08 ; Pointer to the following structure:
 - pNC: Pointer to the DSNAME structure for DC=CONTOSO,DC=COM
 - pszDsaDest : "5fe84f18-3765-4ca3-b895-47802a7ab74f._msdcs.contoso.com"
 - uuidDsaObjDest: _GUID { 5fe84f18-3765-4ca3-b895-47802a7ab74f }
 - ulOptions: DRS_WRIT_REP | DRS_DEL_REF | DRS_ADD_REF

4.1.26.3.2.3 Server Response

Return code of 0.

4.1.26.3.2.4 Final State

The ldap search


```
ldap_search_s(ld, "DC=CONTOSO,DC=com", 0, "(objectclass=*)",[repsTo])
```

returns

Getting 1 entry:

```
>> Dn: DC=CONTOSO,DC=COM
```

repsTo (2): dwVersion = 2,

- v2.cb: 592, v2.cConsecutive Failures: 0, v2.timeLastSuccess: 12924320155
- V2.timeLastAttempt: 0, V2.ulResultLastAttempt: 0,
- V2.cbOtherDraOffset: 216.
- V2.cbOtherDra: 376, V2.ulReplicaFlags: 16, V2.rtSchedule: <ldp:skipped>,
- V2.usnvec.usnHighObjUpdate: 0, v2.usnvec.usnHighPropUpdate:0
- V2.pszUuidDsaObj: 5fe84f18-3765-4ca3-b895-47802a7ab74f
- V2.pszUuidInvocId: 00000000-0000-0000-0000-000000000000
- V2.pszUuidTransportObj: 00000000-0000-0000-0000-000000000000
- V2.cbPASDataOffset: 0 v2~PasData: (none)
- v2~pdsa_rpc_inst
- v2.pszDSIServer 5fe84f18-3765-4ca3-b895-47802a7ab74f._msdcs.CONTOSO.COM
- v2.pszDSIAnnotation (null)
- v2.pszDSIIInstance 5fe84f18-3765-4ca3-b895-47802a7ab74f._msdcs.CONTOSO.COM
- v2.pguidDSIIInstance (null);

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: 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.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): ERROR_INVALID_HANDLE, ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.

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	Verify DSName values.
DRS_VERIFY_SID 0x00000001	Verify objectSid values.
DRS_VERIFY_SAM_ACCOUNT_NAMES 0x00000002	Verify SAMAccountName values.

Value	Meaning
DRS_VERIFY_FPOS 0x00000003	Verify foreign principal object names.

cNames: The number 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 used 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 number of items in the **rpEntInf** array.

rpEntInf: An 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 used 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
referredDomain: unicodestring

ValidatedDRSInput(hDrs, 8)

pdwOutVersion^ := 1
pmsgOut^.V1.error := 0
pmsgOut^.V1.cNames := 0
pmsgOut^.V1.rpEntInf := null
pmsgOut^.V1.PrefixTable.PrefixCount := 0
pmsgOut^.V1.PrefixTable.pPrefixEntry := null

/* Perform input validation and access check */
if dwInVersion ≠ 0x1 then
    return ERROR_DS_DRA_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 ≠ DRS_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_DS_DRA_INVALID_PARAMETER
endif
if msgIn.cNames > 0 and msgIn.rpNames = null then
    return ERROR_DS_DRA_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, referredDomain) 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 :=
                    referredDomain
                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].attrTyp)
        msgOut.rpEntInf[i].AttrBlock.pAttr[j].attrTyp := 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))
        endfor
    endfor
endfor

```

```

        msgOut.rpEntInf[i].AttrBlock.pAttr[j].AttrVal.valCount :=
            k + 1
    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 and 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: A pointer to the request message.

pdwOutVersion: A pointer to the version of the response message. The value must be 1 because that is the only version supported.

pmsgOut: A pointer to the response message.

Return Values: 0 if successful, or a Windows error code if a failure occurs.

Exceptions Thrown: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)): `ERROR_INVALID_HANDLE`, `ERROR_DS_DRS_EXTENSIONS_CHANGED`, `ERROR_DS_DIFFERENT_REPL_EPOCHS`, and `ERROR_INVALID_PARAMETER`.

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: Unused. MUST be 0 and ignored.

pwszAccount: The DN of the object to modify.

cSPN: The number 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 DRSSWriteSPN](#) 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 DRSSWriteSPN](#) 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 Method-Specific Abstract Types and Procedures

4.1.28.2.1 ExecuteWriteSPNRemotely

```
procedure ExecuteWriteSPNRemotely(  
    DWORD dwInVersion,  
    DRS_MSG_SPNREQ *pmsgIn,  
    DWORD *pdwOutVersion,  
    DRS_MSG_SPNREPLY *pmsgOut): ULONG
```

This procedure is executed only on an RODC. It finds a DC that holds a **full NC replica** of the domain NC of the RODC, performs the [IDL_DRSWriteSPN](#) RPC method call with the given parameters against the DC in the client's security context, and returns the value returned by that RPC call.

4.1.28.3 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  
err: DWORD  
operation: DS_SPN_OPERATION  
cSPN: integer  
spnSet: set of unicodestring  
instanceName: unicodestring  
  
ValidateDRSInput(hDrs, 13)
```

```

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

/* Input parameter validation */
if dwInVersion ≠ 1 then
    pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
    return ERROR_INVALID_PARAMETER
endif

/* Input parameter validation */
if ClientUUID(hDrs) ≠ NTDSAPI_CLIENT_GUID
    pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
    return ERROR_INVALID_PARAMETER
endif

/* RODCs do not perform originating updates */
if AmIRODC() then
    return ExecuteWriteSPNRemotely(dwInVersion,
                                    pmsgIn, pdwOutVersion, pmsgOut);
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_FUNCTION
    return ERROR_INVALID_FUNCTION
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_OBJ_NOT_FOUND
    return ERROR_DS_OBJ_NOT_FOUND
endif

/* Perform access checks */

```

```

err = AccessCheckWriteToSpnAttribute(account, spnSet)
if err ≠ ERROR_SUCCESS then
    pmsgOut^.V1.retVal := err
    return err
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. This interface is available only when [msDS-UpdateScript](#) contains a valid value, where the validation criterion is implementation-specific.

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: 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, 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: Unused. MUST be 0 and ignored.

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_DSASExecuteScript](#).

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_DSASExecuteScript](#). 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_DSASExecuteScript** 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

/* Returned message will be version 1 */
pdwOutVersion^ := 1
pmsgOut^V1.dwOperationStatus := ERROR_DS_INTERNAL_FAILURE
pmsgOut^V1.pwErrMessage := null
pmsgOut^V1.cbPassword := 0
pmsgOut^V1.pbPassword := null
pmsgOut^V1.cbHashBody := 0
pmsgOut^V1.pbHashBody := null
pmsgOut^V1.cbHashSignature := 0
pmsgOut^V1.pbHashSignature := null

/* 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

/* 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.pwErrMessage := human-readable description of the error
    return 0

```

```

endif

if not AccessCheckCAR(pc, DS-Execute-Intentions-Script) then
    pmsgOut^.Vl.dwOperationStatus := ERROR_DS_AUTHORIZATION_FAILED
    pmsgOut^.Vl.pwErrMessage := human-readable description of the error
    return 0
endif

if GetKeyLengthHandleT(hRpc) < 128 then
    pmsgOut^.Vl.dwOperationStatus := ERROR_DS_STRONG_AUTH_REQUIRED
    pmsgOut^.Vl.pwErrMessage := human-readable description of the error
    return 0
endif

/* Validate stored script */
if not PrepareScriptVerifyScript(pc) then
    pmsgOut^.Vl.dwOperationStatus := ERROR_DS_INVALID_SCRIPT
    pmsgOut^.Vl.pwErrMessage := human-readable description of the error
    return 0
endif

/* Generate and return password for subsequent call to
 * IDL_DSASExecuteScript() */
pc!msDS-ExecuteScriptPassword := PrepareScriptGeneratePassword()

/* Return password and hashes */
byteseq := pc!msDS-ExecuteScriptPassword
pmsgOut^.Vl.pbPassword := byteseq
pmsgOut^.Vl.cbPassword := byteseq.length

byteseq := PrepareScriptHashBody(pc)
pmsgOut^.Vl.pbHashBody := byteseq
pmsgOut^.Vl.cbHashBody := byteseq.length

byteseq := PrepareScriptHashSignature(pc)
pmsgOut^.Vl.pbHashSignature := byteseq
pmsgOut^.Vl.cbHashSignature := byteseq.length

pmsgOut^.Vl.dwOperationStatus := 0
return 0

```

4.2.2 IDL_DSASExecuteScript (Opnum 1)

The **IDL_DSASExecuteScript** method executes a maintenance script.

```

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
);

```

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

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, 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: Unused. MUST be 0 and ignored.

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_DSAExecuteScript(  
    [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: DSName
msgIn: DSA_MSG_EXECUTE_SCRIPT_REQ_V1

/* returned message is version 1 */
pdwOutVersion^ := 1

pmsgOut^.V1.dwOperationStatus := ERROR_DS_INTERNAL_FAILURE
pmsgOut^.V1.pwErrMsg := null

/* Validate the version */
if dwInVersion ≠ 1 then
    return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1

/* Only 1 instance of this call can be running. */
if ExecuteScriptInProgress() then
    pmsgOut^.V1.dwOperationStatus := ERROR_ACCESS_DENIED
    pmsgOut^.V1.pwErrMsg := 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.pwErrMsg := 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.pwErrMsg := 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 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 an 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, which MUST be retrieved using the method described in [\[MS-RPCE\]](#) section 3.3.3.4.3, has the access indicated by the access right **right** to the attribute **attr** on that object per algorithm, as specified in [\[MS-DYTP\]](#) section 2.5.3.2. The procedure 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, which MUST be retrieved using the method described in [\[MS-RPCE\]](#) section 3.3.3.4.3, has the access indicated by the **control access right right** on that object per algorithm, as specified in [\[MS-DYTP\]](#) section 2.5.3.2. It 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, which MUST be retrieved using the method described in [\[MS-RPCE\]](#) section 3.3.3.4.3, has the access indicated by the access right **right** on that object per algorithm, as specified in [\[MS-DYTP\]](#) section 2.5.3.2. The procedure 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, which MUST be retrieved using the method described in [\[MS-RPCE\]](#) section 3.3.3.4.3, 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 ERROR_SUCCESS
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 ERROR_DS_INVALID_ATTRIBUTE_SYNTAX
                    endif
                else
                    return ERROR_DS_INVALID_ATTRIBUTE_SYNTAX
                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 ERROR_DS_INVALID_ATTRIBUTE_SYNTAX
                    endif
                else
                    return ERROR_DS_INVALID_ATTRIBUTE_SYNTAX
                endif
            endif
        endfor
        return ERROR_SUCCESS
    endif

    return ERROR_DS_INSUFF_ACCESS_RIGHTS
endif

```

5.6 AddSubRef

```

procedure AddSubRef(childNC: DSName): DWORD

```

Informative summary of behavior: This procedure creates a **sub-ref object** to the NC named by *childNC*, if appropriate.


```

parentNC, parentObj: DSName
srAtt: ENTINF
err: DWORD

err:= 0

/* Find the parent NC */
parentNC := GetObjectNC(ChildNC)

/* If the parent NC is not instantiated locally, return */
if not FullReplicaExists(parentNC) and
    not PartialGCReplicaExists(parentNC) then
    return err
endif
/* If child does not exist, create it */

if !ObjExists(childNC) then
    /* Create a subordinate reference object */
    ENTINF_SetValue(srAtt, instanceType,
        {IT_NC_HEAD | IT_UNINSTANT | IT_NC_ABOVE}, dc.prefixTable)
    ENTINF_SetValue(srAtt, objectClass, top, prefixTable)
    ENTINF_SetValue(srAtt, objectCategory, SchemaObj(top), dc.prefixTable)
    ENTINF_SetValue(srAtt, distinguishedName, childNC, prefixTable)
    err := PerformAddOperation(srAtt, childNC, dc.prefixTable, TRUE)
    if(err != 0)
        return err
    endif
else
    if (childNC!isDeleted)
        /*The cross-ref is being undeleted. Undelete the sub-ref object also.*/
        UndeleteObject(childNC)
    endif
endif

/* Ensure that the subordinate reference object is listed in the parent's
    subRefs attribute*/
if (not childNC in parentNC!subRefs) then
    parentNC!subRefs := parentNC!subRefs + {childNC}
endif
return err

```

5.7 AmIRODC

```

procedure AmIRODC() : boolean

```

The AmIRODC procedure returns true if the DC is an RODC.

```

return DSAObj()!objectCategory = SchemaObj(nTDSDSARO)

```

5.8 AmILHServer

```

procedure AmILHServer() : boolean

```

The AmILHServer procedure returns true if the local machine is Windows Server® 2008 operating system or later.

```
/* DS_BEHAVIOR_WIN2008 defined in [MS-ADTS]
 * section 7.1.4.2, "msDS-Behavior-Version: DC Functional Level" */
return DSAObj(!msDS-Behavior-Version ≥ DS_BEHAVIOR_WIN2008
```

5.9 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.

5.10 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 number of items in the pAttr array.

pAttr: An array of attributes and their values.

5.11 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. On each subsequent originating update, if the current value of **dwVersion** is less than 0xFFFFFFFF, then increment it by 1; otherwise set it to 0.
- **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

Version Comparison: The following procedure is used for comparing the **dwVersion** fields of two AttributeStamps:

```
procedure CompareVersions(x: DWORD, y: DWORD): int
```

Informative summary of behavior: This procedure compares two **dwVersions** and returns an integer that is used in AttributeStamp following comparisons.

```
    if x = y then
        return 0
    elseif x > 0x7FFFFFFF then
        if y = (x - 0x80000000) then
            return 1
        elseif (y < (x - 0x7FFFFFFF)) or (x < y) then
            return -1
        else
            return 1
        endif
    elseif x < 0x7FFFFFFF then
        if y = (x + 0x80000000) then
            return -1
        elseif (x < y) and (y < (x - 0x7FFFFFFF)) then
            return -1
        else
            return 1
        endif
    else
        if y = 0xFFFFFFFF then
            return -1
        elseif x < y then
            return -1
        else
            return 1
        endif
    endif
endif
```

AttributeStamp Comparison: Given two **AttributeStamps** *x* and *y*, let *d* be the result of the procedure CompareVersions (x.dwVersion, y.dwVersion).

x is said to be equal to *y* if any of the following is true:

- *x* is null and *y* is null
- *d* = 0 and *x*.timeChanged = *y*.timeChanged and *x*.uuidOriginating = *y*.uuidOriginating

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

x is said to be less than *y* if any of the following is true:

- x is null and y is not 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.

5.12 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 [5.16.2](#), for example, "Object(DS-DN)" and "Object(DN-Binary)".

5.13 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.

5.14 ATTRTYP

ATTRTYP is a concrete type for a compact representation of an **OID**.

This type is declared as follows:

```
typedef ULONG ATTRTYP;
```

Section [5.16.4](#) specifies the procedures that map between ATTRTYP and OID with the aid of a [SCHEMA PREFIX TABLE](#).

5.15 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

```

5.16 ATTRVAL

The **ATTRVAL** structure defines a concrete type for the value of a single attribute.

```

typedef struct {
    [range(0,26214400)] 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.

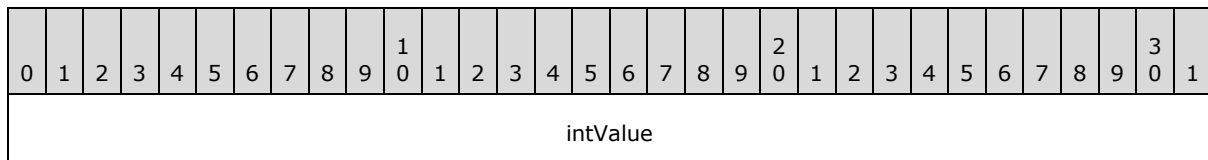
5.16.1 Concrete Value Representations

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 \(section 5.14\)](#)
- [DSNAME \(section 5.49\)](#)
- [DSTIME \(section 5.50\)](#)
- [SYNTAX ADDRESS \(section 5.185\)](#)
- [SYNTAX DISTNAME BINARY \(section 5.186\)](#)

5.16.1.1 INT32

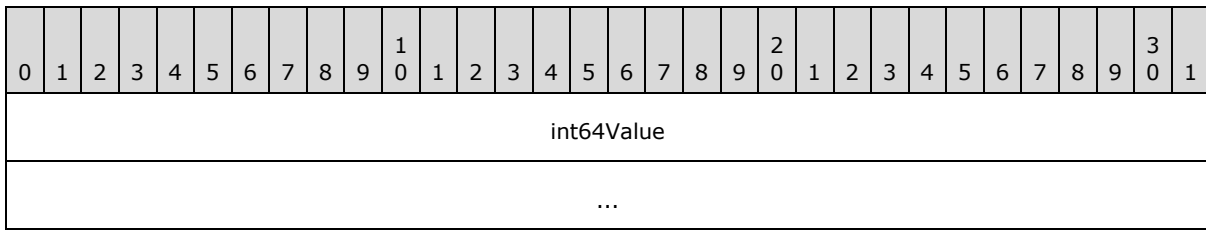
The INT32 type is a 4-byte integer in little-endian form. See [\[MS-DTYP\]](#) section 2.2.22 for its definition.



intValue (4 bytes): A 32-bit signed number in little-endian byte order.

5.16.1.2 INT64

The INT64 type is an 8-byte integer in little-endian form. See [\[MS-DTYP\]](#) section 2.2.23 for its definition.



int64Value (8 bytes): A 64-bit signed number in little-endian byte order.

5.16.1.3 OctetString

The OctetString represents an array of bytes. The number of bytes in the array equals the valLen field of the [ATTRVAL](#) structure.

5.16.1.4 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.

5.16.1.5 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.

5.16.1.6 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.

5.16.1.7 SID

The SID type is an NT security identifier structure, as specified in [\[MS-DTYP\]](#) section 2.4.2.

The data is stored in little-endian byte order.

5.16.2 Abstract Value Representations

The abstract data model utilizes a representation of data values that is used by LDAP, minus the 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. Some syntaxes share an OID, so the syntaxes in the table 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
Enumeration (2.5.5.9)	INTEGER	[RFC2252] section 6.16

LDAP syntax name (OID)	[RFC2252] name	Reference section in [RFC2252] or in this document
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 5.16.2.1
Object(DN-String) (2.5.5.14)	-	Section 5.16.2.2
Object(DN-Binary) (2.5.5.7)	-	Section 5.16.2.3
Object(Access-Point) (2.5.5.14)	-	Section 5.16.2.4
Object(OR-Name) (2.5.5.7)	-	Section 5.16.2.5
String(NT-Sec-Desc) (2.5.5.15)	-	Section 5.16.2.6
String(SID) (2.5.5.17)	-	Section 5.16.2.7
String(Teletex) (2.5.5.4)	-	Section 5.16.2.8

The LDAP syntaxes that are not defined in [\[RFC2252\]](#) are described in the following sections.

5.16.2.1 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.

5.16.2.2 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:

- *S* is a string literal that MUST be present.
- Each *:* is a string literal that MUST be present.
- *char_count* is the number of characters in the *string_value* string.
- *object_DN* is an **object reference** in the format of [Object\(DS-DN\)](#).

5.16.2.3 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:

- *B* is a string literal that MUST be present.
- Each *:* is a string literal that MUST be present.
- *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\(DS-DN\)](#).

5.16.2.4 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:

- *#X500* is a string literal that MUST be present.
- *:* is a string literal that MUST be present.
- *presentation_address* is a value encoded in the Object(Presentation-Address) syntax.
- *object_DN* is an object reference in the format of [Object\(DS-DN\)](#).

5.16.2.5 Object(OR-Name)

A value with the Object(OR-Name) syntax is a UTF-8 string in the following format:

object_DN

where: *object_DN* is an object reference in the format of [Object\(DS-DN\)](#).

5.16.2.6 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.

5.16.2.7 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.

5.16.2.8 String(Teletex)

A value with the String(Teletex) syntax is a UTF-8 string restricted to characters with values between 0x20 and 0x7e, inclusive.

5.16.3 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 [ATTRVAL](#)s.

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 [5.16.2](#) 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](#). These procedures have the following signatures:

```
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 [5.16.2](#).

- `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 [5.16.4](#), 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.
- **payload** represents the data in the 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](#).

5.16.3.1 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          ....
```

5.16.3.2 Enumeration and Integer

The Enumeration and the 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                                     ....

```

5.16.3.3 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.....

```

5.16.3.4 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.

```

5.16.3.5 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.

5.16.3.6 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:

31 32 33 34 35 36 37 38 39          123456789
```

5.16.3.7 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 64 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.
```

5.16.3.8 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](#) procedure.

Example:

```
LDAP value: 2.5.6.5
This corresponds to ATTRTYP value: 0x00010005.

valLen: 4
payload:

05 00 01 00          ....
```

5.16.3.9 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.....
```

5.16.3.10 Object(DS-DN)

The LDAP representation of Object(DS-DN) is defined in section [5.16.2.1](#). 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=01050000000000051500000089598d33d3c56b6894e1f2e6
f4010000>;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 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...
```

5.16.3.11 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 [5.16.2.3](#). 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 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.1.,.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.....

```

5.16.3.12 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 [5.16.2.2](#). 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 such 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.
69 00 63 00 6f 00 64 00 65 00 i.c.o.d.e.

```

5.16.3.13 Object(OR-Name)

The LDAP representation of the attribute value corresponds to [SYNTAX_DISTNAME_BINARY](#). The *object_DN* 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 [5.16.2.5](#).

5.16.3.14 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 [5.16.2.4](#). 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.

5.16.3.15 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: 010500000000000515000000089598d33d3c56b6894e1f2e6f4010000

valLen: 28

payload:

```

```

01 05 00 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.....

```

5.16.3.16 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)
0100048c7000000080000000000000001400000004005c00030000000050028000001000001000000531a72ab2f1ed
011981900aa0040529b010100000000000050a00000000121800ff010f000102000000000005200000002002000000
12140094000200010100000000000050b000000010200001cd509a01845935900020000010200001cd509a01845935
900020000

```

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 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....

```

5.16.4 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 of an object identifier consists of three components, because the end-of-contents component is not present. Only the third component (contents octets) is used here; other components are omitted.

Note The BER encoding of an OID is described in [ITU690](#) section 8.19. To avoid ambiguity, the non-encoded 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: unicodestring,
    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: uncodestring) : 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) : uncodestring
```

Converts a binary OID to its string representation, as described in [\[ITUX690\]](#) 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: uncodestring) : sequence of BYTE
```

Converts an OID string representation to a binary OID, as described in [\[ITUX690\]](#) 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 non-numeric 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 that it is known to not be 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

    procedure 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
"\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
```

5.17 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 number of items in the pAVal array.

pAVal: The sequence of attribute values.

5.18 ATTRVALFromValue

```
procedure ATTRVALFromValue(v: Value,
                           s: Syntax,
                           var t: PrefixTable) : ATTRVAL
```

The ATTRVALFromValue procedure 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 [5.16.3](#) for the specification of this procedure.

5.19 BindToDSA()

```
procedure BindToDSA(dsa: DSNAME): DRS_HANDLE
```

The BindToDSA procedure 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.

5.20 BOOL

A concrete type for a Boolean value, as specified in [\[MS-DTYP\]](#) section 2.2.3.

5.21 BYTE

A concrete type for a single byte, as specified in [\[MS-DTYP\]](#) section 2.2.6.

5.22 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: A pointer to an array of [CHANGELOG_ENTRY](#).

5.23 CHANGELOG_ENTRY

CHANGELOG_ENTRY is a concrete type that is defined in [\[MS-NRPC\]](#) section 3.5.5.6.4, 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.

5.24 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 2.5.3.

5.25 ClientAuthorizationInfo

ClientAuthorizationInfo is an abstract type that represents a client's security context that contains authorization information for a client.

5.26 ClientExtensions

```
procedure ClientExtensions(hDrs: DRS_HANDLE): DRS_EXTENSIONS_INT
```

The ClientExtensions server procedure 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.

5.27 ClientUUID

```
procedure ClientUUID(hDrs: DRS_HANDLE): UUID
```

The ClientUUID procedure returns the GUID that identifies the caller presented in the [IDL_DRSBind](#) call that created hDrs.

5.28 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
```

5.29 ConfigNC

```
procedure ConfigNC(): DSName
```

The ConfigNC procedure returns the dsname of [dc.configNC](#).

5.30 dc, DC

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,  
    invocationId: GUID,
```

usn: 64-bit integer,
prefixTable: [PrefixTable](#),
defaultNC: domain NC replica,
configNC: config NC replica,
schemaNC: schema NC replica,
partialDomainNCs: set of partial domain NC replica,
appNCs: set of application NC replica,
cdcChangeLog: [PdcChangeLog](#),
nt4ReplicationState: [NT4ReplicationState](#),
ldapConnections: [LDAPConnections](#),
replicationQueue: [ReplicationQueue](#),
kccFailedConnections: [KCCFailedConnections](#),
kccFailedLinks: [KCCFailedLinks](#),
rpcClientContexts: [RPCClientContexts](#),
rpcOutgoingContexts: [RPCOutgoingContexts](#),
fLinkValueStampEnabled: boolean,
nt4EmulatorEnabled: boolean,
fEnableUpdates: boolean
]

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
```

5.31 DefaultNC

```
procedure DefaultNC(): DSName
```

The DefaultNC procedure returns the dsname of the [dc](#).defaultNC.

5.32 DelSubRef

```
procedure DelSubRef(childNC: DSName)
```

Informative summary of behavior: This procedure deletes a sub-ref object for the NC *childNC*, if it exists.

```
parentNC: DSName
rt: ULONG

/* If the sub-ref object is not instantiated, delete it */
if(IT_UNINSTANT in childNC!instanceType)
then
    rt:=RemoveObj(childNC,false)
    /* Ignore rt because there are no possible errors returned by RemoveObj
    while deleting a subref object. RemoveObj always returns success in this
    procedure */
else
    /* Otherwise, just prevent continuation referrals from being
    * generated by removing childNC from the parent's subRefs list.
    */
    parentNC := GetObjectNC(ChildNC)
    parentNC!subRefs := parentNC!subRefs - {childNC}
endif
```

5.33 DescendantObject

```
procedure DescendantObject(
    ancestor: DSName, rdns: unicodestring): 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.

5.34 DN

DN is an abstract type that is a *unicodestring* (section [3.4.3](#)) that contains a DN of the form specified in [\[RFC2253\]](#).

5.35 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](#)]

5.36 DomainNameFromNT4AccountName

```
procedure DomainNameFromNT4AccountName(
    nt4AccountName: unicodestring): unicodestring
```

If nt4AccountName is a name in Microsoft Windows NT® 4.0 operating system account name format, that is, two components separated by a backslash (for example, "DOMAIN\username"), 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.

5.37 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.

5.38 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																															

ConfigObjGUID
...
...
...

cb (4 bytes): The count of bytes in the fields **dwFlags** through **dwReplEpoch**, inclusive. [<36>](#) [<37>](#) This field allows the DRS_EXTENSIONS_INT structure to be extended by including new fields to the end of the structure.

dwFlags (4 bytes): The **dwFlags** field contains individual bit flags that describe the capabilities of the DC that produced the DRS_EXTENSIONS_INT structure. [<38>](#)

The following table lists the bit flags, which are presented in little-endian byte order.

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

BAS (DRS_EXT_BASE, 0x00000001): Unused. SHOULD be 1 and MUST be ignored.

AS (DRS_EXT_ASYNCREPL, 0x00000002): If present, signifies that the DC supports [DRS_MSG_REPADD_V2](#) and DRS_MSG_GETCHGREQ_V2.

RM (DRS_EXT_REMOVEAPI, 0x00000004): If present, signifies that the DC supports [IDL_DRSRemoveDsServer](#) and [IDL_DRSRemoveDsDomain](#).

MV (DRS_EXT_MOVEREQ_V2, 0x00000000): If present, signifies that the DC supports [DRS_MSG_MOVEREQ_V2](#).

DF (DRS_EXT_GETCHG_DEFLATE, 0x00000010): If present, signifies that the DC supports [DRS_MSG_GETCHGREPLY_V2](#).

DC (DRS_EXT_DCINFO_V1, 0x00000020): If present, signifies that the DC supports [IDL_DRSDomainControllerInfo](#).

UO (DRS_EXT_RESTORE_USN_OPTIMIZATION, 0x00000040): Unused. SHOULD be 1 and MUST be ignored.

AE (DRS_EXT_ADDENTRY, 0x00000080): If present, signifies that the DC supports [IDL_DRSAddEntry](#).

KE (DRS_EXT_KCC_EXECUTE, 0x00000100): If present, signifies that the DC supports [IDL_DRSExecuteKCC](#).

AE2 (DRS_EXT_ADDENTRY_V2, 0x00000200): If present, signifies that the DC supports [DRS_MSG_ADDENTRYREQ_V2](#).

LVR (DRS_EXT_LINKED_VALUE_REPLICATION, 0x00000400): If present, signifies that the DC supports link value replication, and this support is enabled.

DC2 (DRS_EXT_DCINFO_V2, 0x00000800): If present, signifies that the DC supports [DRS_MSG_DCINFOREPLY_V2](#).

INR (DRS_EXT_INSTANCE_TYPE_NOT_REQ_ON_MOD, 0x00001000): Unused. SHOULD be 1 and MUST be ignored.

CB (DRS_EXT_CRYPTO_BIND, 0x00002000): 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, 0x00004000): If present, signifies that the DC supports [IDL DRSGetReplInfo](#).

SE (DRS_EXT_STRONG_ENCRYPTION, 0x00008000): If present, signifies that the DC supports additional 128-bit encryption for passwords over the wire. DCs MUST NOT replicate passwords to other DCs that do not support this extension.

DCF (DRS_EXT_DCINFO_VFFFFFFFF, 0x00010000): If present, signifies that the DC supports [DRS_MSG_DCINFOREPLY_VFFFFFFFF](#).

TM (DRS_EXT_TRANSITIVE_MEMBERSHIP, 0x00020000): If present, signifies that the DC supports [IDL DRSGetMemberships](#).

SH (DRS_EXT_ADD_SID_HISTORY, 0x00040000): If present, signifies that the DC supports [IDL DRSAddSidHistory](#).

PB3 (DRS_EXT_POST_BETA3, 0x00080000): Unused. SHOULD be 1 and MUST be ignored.

GC5 (DRS_EXT_GETCHGREQ_V5, 0x00100000): If present, signifies that the DC supports [DRS_MSG_GETCHGREQ_V5](#).

GM2 (DRS_EXT_GETMEMBERSHIPS2, 0x00200000): If present, signifies that the DC supports [IDL DRSGetMemberships2](#).

GC6 (DRS_EXT_GETCHGREQ_V6, 0x00400000): Unused. This bit was used for a pre-release version of Windows. No released version of Windows references it. This bit can be set or unset with no change in behavior.

ANC (DRS_EXT_NONDOMAIN_NCS, 0x00800000): If present, signifies that the DC supports application NCs.

GC8 (DRS_EXT_GETCHGREQ_V8, 0x01000000): If present, signifies that the DC supports [DRS_MSG_GETCHGREQ_V8](#).

GR5 (DRS_EXT_GETCHGREPLY_V5, 0x02000000): If present, signifies that the DC supports [DRS_MSG_GETCHGREPLY_V5](#).

GR6 (DRS_EXT_GETCHGREPLY_V6, 0x04000000): If present, signifies that the DC supports [DRS_MSG_GETCHGREPLY_V6](#).

WB3 (DRS_EXT_WHISTLER_BETA3, 0x08000000): If present, signifies that the DC supports [DRS_MSG_ADDENTRYREPLY_V3](#), [DRS_MSG_REPVERIFYOBJ](#), and [DRS_MSG_GETCHGREPLY_V7](#).

DF2 (DRS_EXT_W2K3_DEFLATE, 0x10000000): If present, signifies that the DC supports the W2K3 AD deflation library.

GC10 (DRS_EXT_GETCHGREQ_V10, 0x20000000): If present, signifies that the DC supports [DRS_MSG_GETCHGREQ_V10](#).

R2 (DRS_EXT_RESERVED_FOR_WIN2K_OR_DOTNET_PART2, 0x40000000): Unused. MUST be 0 and ignored.

R3 (DRS_EXT_RESERVED_FOR_WIN2K_OR_DOTNET_PART3, 0x80000000): Unused. MUST be 0 and ignored.

SiteObjGuid (16 bytes): 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 (4 bytes): 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. [<39>](#)

dwReplEpoch (4 bytes): A 32-bit, unsigned integer value that specifies the replication epoch. This value is set to zero by all client callers. The server sets this value by assigning the value of [msDS-ReplicationEpoch](#) from its nTDSDSA object. If **dwReplEpoch** is not included in DRS_EXTENSIONS_INT, the value is considered to be zero. [<40>](#)

dwFlagsExt (4 bytes): An 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. If **dwFlagsExt** is not included in DRS_EXTENSIONS_INT, all bit flags are considered unset. The following table lists the bit flags, which are presented in little-endian byte order. [<41>](#)

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	R	L	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
					B	H	A																								

DA (DRS_EXT_ADAM, 0x00000001): If present, signifies that the DC supports [DRS_MSG_REPSYNC_V1](#), [DRS_MSG_UPDREFS_V1](#), [DRS_MSG_INIT_DEMOTIONREQ_V1](#), [DRS_MSG_REPLICA_DEMOTIONREQ_V1](#), and [DRS_MSG_FINISH_DEMOTIONREQ_V1](#).

LH (DRS_EXT_LH_BETA2, 0x00000002): 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](#).

RB (DRS_EXT_RECYCLE_BIN, 0x00000004): If present, signifies that the DC has enabled the Recycle Bin optional feature.

ConfigObjGUID (16 bytes): A GUID. This field is set to zero by all client callers. The server sets this field by assigning it the value of the objectGUID of the config NC object. If

ConfigObjGUID is not included in DRS_EXTENSIONS_INT, the value is considered to be the NULL GUID value.<42>

5.39 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.

5.40 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;
```

It is a bit field, presented in little-endian byte order, that contains the following values.

Seven 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	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
M	P	I	W	A	A	G	A	F	S	N	G	G	C	T	A	I	S	N	R	S	S	F	F	G	S	N	U	D	D	S	P
R	S	S	R	L	R	C	S	S	N	R	S	A	O	S	S	S	S	S	G	U	Q	S	S	P	P	N	C	P	A	F	E
				L		/		/	/	R	/				R	N		Y	/			P	I				S	S			
				/		U		N	R		L				/				N												
				D		N		S	F		O				I				D												
				R											E																

X: Unused. MUST be zero and ignored.

AS (DRS_ASYNC_OP, 0x00000001): Perform the operation asynchronously.

GC (DRS_GETCHG_CHECK, 0x00000002): Treat ERROR_DS_DRA_REF_NOT_FOUND and ERROR_DS_DRA_REF_ALREADY_EXISTS as success for calls to [IDL_DRSUpdateRefs \(section 4.1.26\)](#).

UN (DRS_UPDATE_NOTIFICATION, 0x00000002): Identifies a call to [IDL_DRSReplicaSync](#) that was generated due to a replication notification. See [\[MS-ADTS\]](#) section 3.1.1.5.1.5 for more information on replication notifications. This flag is ignored by the server.

AR (DRS_ADD_REF, 0x00000004): Register a client DC for notifications of updates to the NC replica.

ALL (DRS_SYNC_ALL, 0x00000008): Replicate from all server DCs.

DR (DRS_DEL_REF, 0x00000008): Deregister a client DC from notifications of updates to the NC replica.

WR (DRS_WRIT_REP, 0x00000010): Replicate a writable replica, not a read-only partial replica or read-only full replica.

IS (DRS_INIT_SYNC, 0x00000020): Perform replication at startup.

PS (DRS_PER_SYNC, 0x00000040): Perform replication periodically.

MR (DRS_MAIL_REP, 0x00000080): Perform replication using SMTP as a transport.

ASR (DRS_ASYNC_REP, 0x00000100): Populate the NC replica asynchronously.

IE (DRS_IGNORE_ERROR, 0x00000100): Ignore errors.

TS (DRS_TWOWAY_SYNC, 0x00000200): Inform the server DC to replicate from the client DC.

CO (DRS_CRITICAL_ONLY, 0x00000400): Replicate only **system-critical objects**.

GA (DRS_GET_ANC, 0x00000800): Include updates to ancestor objects before updates to their descendants.

GS (DRS_GET_NC_SIZE, 0x00001000): Get the approximate size of the server NC replica.

LO (DRS_LOCAL_ONLY, 0x00001000): Perform the operation locally without contacting any other DC.

NRR (DRS_NONGC_RO_REP, 0x00002000): Replicate a read-only full replica. Not a writable or partial replica.

SN (DRS_SYNC_BYNAME, 0x00004000): Choose the source server by network name.

RF (DRS_REF_OK, 0x00004000): Allow the NC replica to be removed even if other DCs use this DC as a replication server DC.

FS (DRS_FULL_SYNC_NOW, 0x00008000): Replicate all updates in the replication cycle, even those that would normally be filtered.

NS (DRS_NO_SOURCE, 0x00008000): The NC replica has no server DCs.

FSI (DRS_FULL_SYNC_IN_PROGRESS, 0x00010000): When the flag DRS_FULL_SYNC_NOW is received in a call to [IDL_DRSReplicaSync](#), the flag DRS_FULL_SYNC_IN_PROGRESS is sent in the associated calls to [IDL_DRSGetNCChanges](#) until the replication cycle completes. This flag is ignored by the server.

FSP (DRS_FULL_SYNC_PACKET, 0x00020000): Replicate all updates in the replication request, even those that would normally be filtered.

SQ (DRS_SYNC_REQUEUE, 0x00040000): This flag is specific to the Microsoft client implementation of **IDL_DRSGetNCChanges**. It is used to identify whether the call was placed in the replicationQueue more than once due to implementation-specific errors. This flag is ignored by the server.

SU (DRS_SYNC_URGENT, 0x00080000): Perform the requested replication immediately; do not wait for any timeouts or delays. For information about urgent replication, see [\[MS-ADTS\]](#) section 3.1.1.5.1.6.

RG (DRS_REF_GCSPN, 0x00100000): Requests that the server add 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 notifying client DC contacts the server DC using the service principal name that begins with "GC" (section [2.2.3.2](#)).

ND (DRS_NO_DISCARD, 0x00100000): This flag is specific to the Microsoft implementation. It identifies when the client DC should call the requested **IDL_DRSReplicaSync** method individually, without overlapping other outstanding calls to **IDL_DRSReplicaSync**. This flag is ignored by the server.

NSY (DRS_NEVER_SYNCED, 0x00200000): There is no successfully completed replication from this source server.

SS (DRS_SPECIAL_SECRET_PROCESSING, 0x00400000): Do not replicate attribute values of attributes that contain secret data.

ISN (DRS_INIT_SYNC_NOW, 0x00800000): Perform initial replication now.

PE (DRS_PREEMPTED, 0x01000000): The replication attempt is preempted by a higher priority replication request.

SF (DRS_SYNC_FORCED, 0x02000000): Force replication, even if the replication system is otherwise disabled.

DAS (DRS_DISABLE_AUTO_SYNC, 0x04000000): Disable replication induced by update notifications.

DPS (DRS_DISABLE_PERIODIC_SYNC, 0x08000000): Disable periodic replication.

UC (DRS_USE_COMPRESSION, 0x10000000): Compress response messages.

NN (DRS_NEVER_NOTIFY, 0x20000000): Do not send update notifications.

SP (DRS_SYNC_PAS, 0x20000000): Expand the partial attribute set of the partial replica.

GP (DRS_GET_ALL_GROUP_MEMBERSHIP, 0x80000000): Replicate all kinds of group membership. If this flag is not present nonuniversal group membership will not be replicated.

For information about the Windows versions in which these flags were introduced and supported, see the following behavior note. [<43>](#)

5.41 DRS_MORE_GETCHGREQ_OPTIONS

DRS_MORE_GETCHGREQ_OPTIONS is a concrete type for a set of extra options sent to the [IDL_DRSGetNCChanges](#) method.

This type is declared as follows:

```
typedef unsigned long DRS_MORE_GETCHGREQ_OPTIONS;
```

It is a bit field, presented in little-endian byte order, which 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	
X	X	X	X	X	X	X	T G 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: Unused. MUST be zero and ignored.

TGT (DRS_GET_TGT, 0x00000001): Include updates to the target object of a link value before updates to the link value.

5.42 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, presented in little-endian byte order, that contains the following values:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
X	X	X	X	X	T Y P	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	R O	X	X	X	X	X	X	X	X

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 DCE RPC 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.
SECBUFFER_STREAM_HEADER 0x00000007	Indicates a protocol-specific header for a particular record.

RO (SECBUFFER_READONLY, 0x80000000): 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.

5.43 DRS_SecBufferDesc

DRS_SecBufferDesc is a Generic Security Service (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 number of items in the Buffers array.

Buffers: Buffers that contain authentication data.

5.44 DRS_SPN_CLASS

A *unicodestring* constant (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".

5.45 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: Sync NC replica from server DC.

DS_REPL_OP_TYPE_ADD: Add NC replica server DC.

DS_REPL_OP_TYPE_DELETE: Remove NC replica server DC.

DS_REPL_OP_TYPE_MODIFY: Modify NC replica server DC.

DS_REPL_OP_TYPE_UPDATE_REFS: Update NC replica client DC.

5.46 DSAObj

```
procedure DSAObj(): DSName
```

The DSAObj procedure returns the dsname of the DC's [nTDSDSA](#) object.

```
return select one o from children ConfigNC()
where o!objectGUID = dc.serverGUID
```

5.47 DSA_RPC_INST

The **DSA_RPC_INST** structure is a concrete type that represents a DC.

```
typedef struct _DSA_RPC_INST {
    DWORD cb;
    DWORD cbpszServerOffset;
    DWORD cbpszAnnotationOffset;
    DWORD cbpszInstanceOffset;
    DWORD cbpguidInstanceOffset;
} DSA_RPC_INST,
*PDSA_RPC_INST;
```

cb: The total number of bytes in the **DSA_RPC_INST** structure.

cbpszServerOffset: The offset from the start of the **DSA_RPC_INST** structure to a location that specifies the start of the server name of this instance.

cbpszAnnotationOffset: The offset from the start of the **DSA_RPC_INST** structure to a location that specifies the start of the annotation of this instance.

cbpszInstanceOffset: The offset from the start of the **DSA_RPC_INST** structure to a location that specifies the start of the [NetworkAddress \(section 5.130\)](#) of this instance.

cbpguidInstanceOffset: The offset from the start of the **DSA_RPC_INST** structure to a location that specifies the start of the GUID for the instance.

5.48 DSName

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 DSName = [dn: StringName , guid: GUID, sid: Sid]
```

The *dn* field corresponds to the **StringName** field of the [DSNAME](#) structure and contains the DN of the object.

The *guid* field corresponds to the **Guid** field of the [DSNAME](#) structure and contains the value of the object's [objectGUID](#) attribute.

The *sid* field corresponds to the **Sid** field of the [DSNAME](#) structure. 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.

5.49 DSNAME

DSNAME is a concrete type for representing a [DSName](#), 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;
    [range(0, 10485761), 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. If the values for all fields in the **GUID** structure are zero,

this indicates 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 [\[MS-WSQ\]](#) section 3.1.2.1.3), 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. 28 bytes is sufficient for a Windows SID.

NameLen: The number of characters in the **StringName** field, not including the terminating null character, used to represent the object's [distinguishedName](#) attribute value. Zero indicates that the DSNAME does not identify the [distinguishedName](#) value of the directory object.

StringName: A 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 terminating null character. Each Unicode value is encoded as 2 bytes. The byte ordering is little-endian.<44>

The following table shows an alternative representation of this structure.

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
structLen																															
SidLen																															
Guid.Data1																															
Guid.Data2																Guid.Data3															
Guid.Data4...																															
...Guid.Data4																															
Sid...																															
...Sid...																															
...Sid...																															
...Sid...																															
...Sid...																															
...Sid...																															
...Sid																															

NameLen
StringName (Variable Length) ...

Note All fields have little-endian byte ordering.

5.49.1 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 and *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.

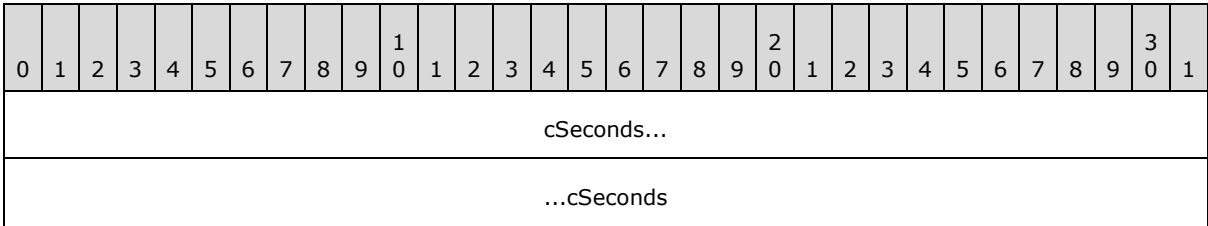
5.50 DSTIME

DSTIME is a concrete type for time expressed as the number of seconds since January 1, 1601, 12:00:00 A.M.

This type is declared as follows:

```
typedef LONGLONG DSTIME;
```

The following diagram shows an alternative representation of this type.



Note Byte ordering is little-endian.

5.51 DWORD

A concrete type for a 32-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.9.

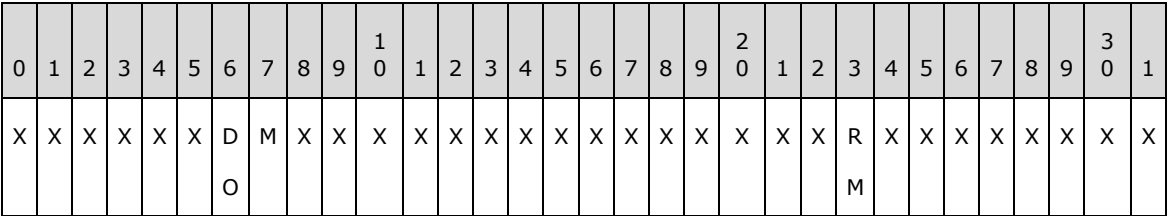
5.52 ENTINF

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, which are presented in little-endian byte order.



X: Unused. MUST be zero and ignored.

M (ENTINF_FROM_MASTER, 0x00000001): Retrieved from a full replica.

DO (ENTINF_DYNAMIC_OBJECT, 0x00000002): A **dynamic object**.

RM (ENTINF_REMOTE_MODIFY, 0x00010000): A remote modify request to IDL_DRSSAddEntry (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**.

5.53 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
```

5.54 ENTINF_SetValue

```
procedure ENTINF_SetValue (  
    var entInf: ENTINF,  
    attribute: ATTRTYP,  
    attrVal: ATTRVAL,  
    prefixTable: PrefixTable)
```

The ENTINF_SetValue procedure 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.

5.55 ENTINF_EnumerateAttributes

```
procedure ENTINF_EnumerateAttributes(  
    e: ENTINF,  
    prefixTable: PrefixTable): set of ATTRTYP
```

The ENTINF_EnumerateAttributes procedure 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.

5.56 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.

5.57 Expunge

```

procedure Expunge(obj: DSName)

```

The Expunge procedure 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.

5.58 FILETIME

FILETIME is a concrete type for a time, as specified in [\[MS-DTYP\]](#) section 2.3.1.

5.59 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
endif

```

```

endfor
pPartialAttrVector^.cattrs := j

return pPartialAttrVector^

```

5.60 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^

```

5.61 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 $s[start] \neq c, \dots, s[i-1] \neq c$. If such an i exists, this procedure returns i . Otherwise, this procedure returns -1.

5.62 FindCharRev

```
procedure FindCharRev(  
    s: unichodestring,  
    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 $s = \text{null}$, $\text{start} < 0$ or $\text{start} > s.\text{length}-1$, this procedure returns -1. Otherwise, let s be represented as the sequence of characters $\{s[0], \dots s[s.\text{length} - 1]\}$. Let i be such that $i \geq \text{start}$, $i \leq s.\text{length} - 1$, $s[i] = c$, and $s[i+1] \neq c, \dots, s[s.\text{length} - 1] \neq c$. If such an i exists, this procedure returns i . Otherwise, this procedure returns -1.

5.63 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	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Version																															
RecordCount																															
Records (variable)																															
...																															

Version (4 bytes): The version of the data structure. The only supported version of the data structure is 1.

RecordCount (4 bytes): The number of records present in the data structure.

Records (variable): Variable-length records that each contain a specific 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 [5.63.1](#).

Note All fields have little-endian byte ordering.

5.63.1 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	1	2	3	4	5	6	7	8	9	30	1
RecordLen																															
Flags																															
Timestamp																															
...																															
RecordType										ForestTrustData (variable)																					
...																															

RecordLen (4 bytes): The length, in bytes, of the entire record.

Flags (4 bytes): 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, which are presented in little-endian byte order.

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

X: Unused. Must be zero and ignored.

TDN (LSA_TLN_DISABLED_NEW, 0x00000001): The entry is not yet enabled.

TDA (LSA_TLN_DISABLED_ADMIN, 0x00000002): The entry is disabled by the administrator.

TDC (LSA_TLN_DISABLED_CONFLICT, 0x00000004): 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, which are presented in little-endian byte order.

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

X: Unused. MUST be zero and ignored.

SDA (LSA_SID_DISABLED_ADMIN, 0x00000001): The entry is disabled for SID-based matches by the administrator.

SDC (LSA_SID_DISABLED_CONFLICT, 0x00000002): The entry is disabled due to a SID conflict with another trusted domain.

NDA (LSA_NB_DISABLED_ADMIN, 0x00000004): The entry is disabled for NetBIOS name-based matches by the administrator.

NDC (LSA_NB_DISABLED_CONFLICT, 0x00000008): 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 (8 bytes): A [FILETIME \(section 5.58\)](#) that contains the time when this entry was created.

RecordType (1 byte): An 8-bit value that specifies the type of record contained in this specific entry. The allowed values are specified in section [5.64](#).

ForestTrustData (variable): A variable length, type-specific record, depending on the RecordType value, that contains the specific type of data about the forest trust relationship.

Important The type-specific ForestTrustData record is not necessarily aligned to a 32-bit boundary. Each record starts at the byte following the **RecordType** field.

There are three different type-specific records. Depending on the value of the **RecordType** field, the structure of the type-specific record differs as described below.

- If RecordType = 0 or RecordType = 1, then the type-specific record is represented in the following manner.

0	1	2	3	4	5	6	7	8	9	10	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 data shown below 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 2.4.2.

DnsNameLen: The length, in bytes, of the **DnsName** field.

DnsName: The FQDN 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 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: Trusted forest data.

5.63.2 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 section 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 UnmarshalForestTrustInfo 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 buffer is interpreted as a UTF-8 string, and a corresponding *unicodestring* (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 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 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
pdwRecordLength: 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 */
    pdwRecordLength := 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 procedures are used to determine if a given domain name, SID or UPN is in a trusted forest. Since they make use of forest trust information data stored in objects in the NC replica of the forest root domain (see FOREST_TRUST_INFORMATION), these functions only work on GC servers or DCs in the forest root domain.

```

procedure IsDomainNameInTrustedForest(name: unicodestring,
    referredDomain: 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, referredDomain) then
        return true
    endIf

    if IsDomainNetbiosNameInTrustedForest(name, referredDomain) then
        return true
    endIf

    return false

procedure IsDomainSidInTrustedForest(sid: SID): boolean

```

Informative summary of behavior: The IsDomainSidInTrustedForest procedure determines if the domain with the SID given by sid is in a trusted forest.

```

tdos: set of DSName
f: X_FOREST_TRUST_INFORMATION
b: boolean

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.Sid = sid and
          LSA_FTRECORD_DISABLED_REASONS not in e.Flags) then

        b := true
        foreach g in f.Entries
          if (g.ForestTrustType = ForestTrustTopLevelNameEx and
              LSA_FTRECORD_DISABLED_REASONS not in g.Flags and
              (g.TopLevelName = e.DomainInfo.DnsName or
               IsSubdomainOf(e.DomainInfo.DnsName, g.TopLevelName))) then

            b := false
            break
          endif
        endfor

        if b then
          return true
        endif
      endif
    endfor
  endif
endfor
return false

procedure IsUPNInTrustedForest(upn: uncodestring): boolean

```

Informative summary of behavior: The IsUPNInTrustedForest procedure determines if the domain containing the account with the UPN given by upn is in a trusted forest.

```

interpret upn as being in the format "username@domainName"
return IsNamespaceInTrustedDomain(domainName, trustedForestName)

```

The IsDomainNameInTrustedForest procedure uses the following helper procedures to determine if a domain is in a trusted forest.

```

procedure IsSubdomainOf(subdomainName: uncodestring,
    superiordomainName: uncodestring): boolean

```

The IsSubdomainOf procedure takes a pair of domain names and returns true if subdomainName is a subdomain of superiordomainName as described in [RFC1034](#) section 3.1, and false otherwise.

```

procedure ForestTrustOwnsName(f: X_FOREST_TRUST_INFORMATION, name: uncodestring): boolean

```

```

/* if the name matches or is a subdomain of one in the exclusion list, the
 * forest does not own this name */
foreach e in f.Entries
    if (e.ForestTrustType = ForestTrustTopLevelNameEx and
        (e.TopLevelName = name or

            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 = name or
                IsSubdomainOf(name, e.TopLevelName))) then
        return true
    endif
endfor

return false

procedure IsDomainDnsNameInTrustedForest(name: unicodestring,
    referredDomain: 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(f, e.DomainInfo.DnsName) then
                referredDomain := o!trustPartner
                return true
            endif
        endfor
    endif
endfor

return false

procedure IsDomainNetbiosNameInTrustedForest
    (name: unicodestring, referredDomain: 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(f, e.DomainInfo.DnsName) then
                    referredDomain := o!trustPartner
                    return true
                endif
            endfor
        endif
    endfor

return false

```

The IsUPNInTrustedForest procedure uses the following helper procedure to determine if a UPN is in a trusted forest.

```

procedure IsNamespaceInTrustedDomain
    (name: unicodestring, trustedForestName: unicodestring): boolean

    tdos: set of DSName
    f: X_FOREST_TRUST_INFORMATION
    b: boolean
    dnsParent: unicodestring
    parents: set of unicodestring

    /* if name is A.B.C, parents has the values A.B.C, B.C, and C */
    parents := DNS parents of name

    foreach dnsParent in parents
        /* 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 = ForestTrustTopLevelName and
        e.TopLevelName = dnsParent and
        LSA_FTRECORD_DISABLED_REASONS not in e.Flags) then

        b := true
        foreach g in f.Entries
            if (g.ForestTrustType = ForestTrustTopLevelNameEx and
                LSA_FTRECORD_DISABLED_REASONS not in g.Flags and
                (g.TopLevelName = dnsParent or
                 IsSubdomainOf(dnsParent, g.TopLevelName))) then
                b := false
                break
            endif
        endfor

        if (b) then
            trustedForestName := o!trustPartner
            return true
        endif
    endif
endfor
endif
endfor
endfor

return false

```

5.64 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
} FOREST_TRUST_RECORD_TYPE;

```

5.65 ForestRootDomainNC

```

procedure ForestRootDomainNC(): DSName

```

The ForestRootDomainNC procedure returns the [DSName](#) of the **forest root domain NC**.

5.66 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 (DSABObj()!msDS-hasMasterNCs +
              DSABObj()!msDS-hasFullReplicaNCs)

```

5.67 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

```

5.68 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

```

5.69 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
    end if
end if

```

```

    else
        ncType := ncType + {NCT_FILTERED_ATTRIBUTE_SET,
                           NCT_GC_PARTIAL}
    endif
endif
return ncType

```

5.70 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.

5.71 GetCallerAuthorizationInfo

```

procedure GetCallerAuthorizationInfo(): ClientAuthorizationInfo

```

The GetCallerAuthorizationInfo procedure returns the [ClientAuthorizationInfo](#) (a security token) of the current caller. For more information, see [\[MS-DTYP\]](#) section 2.5.3.

5.72 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

```

5.73 GetDSNameFromAttrVal

```

procedure GetDSNameFromAttrVal(attrTyp: ATTRTYP, attrVal: ATTRVAL): DSName

```

The GetDSNameFromAttrVal procedure extracts the DSName from the [ATTRVAL](#) *attrVal* based on its syntax, which is determined from the [ATTRTYP](#) *attrTyp*. If the syntax is not one of Object(DS-DN), Object(DN-String), or Object(DN-Binary), this procedure returns a null DSName.

5.74 GetDSNameFromDN

```
procedure GetDSNameFromDN(dn: uncodestring): DSName
```

The GetDSNameFromDN procedure produces a [DSName](#) from dn. Let d represent the returned [DSName](#). It is the case that d.dn= dn. If there is an object o in an NC replica hosted by the server such that o!distinguishedName = dn, then d.guid =o!objectGUID; otherwise, all fields of d.guid are zero. Furthermore, if o!objectSid ≠ null, then d.sid = o!objectSid; otherwise d.sid = null.

5.75 GetDSNameFromNetworkAddress

```
procedure GetDSNameFromNetworkAddress(n: NetworkAddress): DSName
```

The GetDSNameFromDN procedure produces a [DSName](#) from the [NetworkAddress](#) n. If n is an FQDN such that, for a DSA object d, there is a server object s such that d!parent = s and s!dnsHostName = n, then return the DSName of d. If n is in the format <DC-name>:<DC-identifier> as described in section [5.130](#), return the DSName of the DC's DSA object. Otherwise return NULL.

5.76 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
```

5.77 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

Symbolic constant	Value
FSMO_INFRASTRUCTURE	5

5.78 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".

5.79 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, or returns NULL if the NC is not found or it is not of the DN form specified in [\[RFC2253\]](#).

5.80 GetProxyEpoch

```
procedure GetProxyEpoch(dnbinValue: DNBinary): DWORD
```

The GetProxyEpoch procedure returns the decoded proxy epoch field from the dnbinValue, which is a [proxiedObjectName](#) value.

5.81 GetProxyType

```
procedure GetProxyType(dnbinValue: DNBinary): DWORD
```

The GetProxyType procedure returns the decoded proxy type field from the dnbinValue, which is a [proxiedObjectName](#) value.

5.82 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".

5.83 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".

5.84 groupType Bit Flags

The groupType bit flags may appear in values of the [groupType](#) attribute that define a group type. The bit flags are presented below in little-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	Q	B	U	R	A	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	S	X	X	X	X	X	X	X
		G	G	G	G	G																	E								

X: Unused. MUST be zero and ignored.

AG (GROUP_TYPE_ACCOUNT_GROUP, 0x00000002): The account group type.

RG (GROUP_TYPE_RESOURCE_GROUP, 0x00000004): The resource group type.

UG (GROUP_TYPE_UNIVERSAL_GROUP, 0x00000008): The universal group type.

BG (GROUP_TYPE_APP_BASIC_GROUP, 0x00000010): The application basic group type.

QG (GROUP_TYPE_APP_QUERY_GROUP, 0x00000020): The application query group type.

SE (GROUP_TYPE_SECURITY_ENABLED, 0x80000000): The group is security-enabled.

5.85 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.

5.86 GuidFromString

```
procedure GuidFromString(BracedFormat: boolean,
    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. When *BracedFormat* is true, to be a valid string representation of a GUID, *strGuid* MUST be in the curly braced GUID string format as defined in [\[MS-DTYP\]](#) section 2.3.2.3. When *BraceFormat* is false, to be a valid string representation of a GUID, *strGuid* MUST be in the string GUID format as defined in [\[RFC4122\]](#). If *strGuid* is not a valid string representation of a GUID, null is returned.

5.87 GuidToString

```
procedure GuidToString(guid: GUID): unicodestring
```

The `GuidToString` procedure converts `guid` to the concatenation of "{", the string representation defined in [\[RFC4122\]](#) section 3, and "}; for example, {12aa5f43-c776-4d63-b347-1175df806200}.

5.88 handle_t

`handle_t` is a concrete type for an RPC binding handle, as specified in [\[C706\]](#) section 4.2.9.7 and [\[MS-DTYP\]](#) section 2.1.3.

5.89 instanceType Bit Flags

The `instanceType` bit flags are bits that may appear in values of the `instanceType` attribute. The bit flags are presented below in little-endian byte order

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	N	N	N	W	U	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
		G	C	A	R	I	H																								

X: Unused. MUST be zero and ignored.

NH (IT_NC_HEAD, 0x00000001): The object is the root of an NC.

UI (IT_UNINSTANT, 0x00000002): The NC replica has not yet been instantiated.

WR (IT_WRITE, 0x00000004): The object is writable.

NA (IT_NC_ABOVE, 0x00000008): The DC hosts the NC above this one. The `IT_NC_HEAD` bit must also be set.

NC (IT_NC_COMING, 0x00000010): 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, 0x00000020): The NC replica is in the process of being removed from the DC. `IT_NC_HEAD` must also be set.

5.90 Is2PartSPN

```
procedure Is2PartSPN(spn: unicodestring): boolean
```

The Is2PartSPN procedure returns true if spn is an SPN consisting of two parts, and false otherwise.

5.91 Is3PartSPN

```
procedure Is3PartSPN(spn: uncodestring): boolean
```

The Is3PartSPN procedure returns true if spn is an SPN consisting of three parts, and false otherwise.

5.92 IsAdlds

```
procedure IsAdlds() : boolean
```

If the local DC is running Active Directory as AD LDS, this procedure returns TRUE. It returns FALSE otherwise.

5.93 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.

5.94 IsDomainNameInTrustedForest

```
procedure IsDomainNameInTrustedForest(name: uncodestring,  
var referredDomain: uncodestring) : 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 an FQDN or a NetBIOS name of a domain. If the IsDomainNameInTrustedForest procedure returns true, the *referredDomain* parameter value will be set to the FQDN of the root domain of the forest of the domain specified by the *name* parameter. If the IsDomainNameInTrustedForest procedure returns false, the value of the *referredDomain* parameter remains unchanged.

See section [5.63](#) for the specification of this procedure.

5.95 IsDomainSidInTrustedForest

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

The IsDomainSidInTrustedForest procedure returns true if the domain identified by *sid* 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 *sid* parameter is the SID of a domain.

See section [5.63.2](#) for the specification of this procedure.

5.96 IsDCAccount

```
procedure IsDCAccount(o: DSName): boolean
```

The IsDCAccount procedure returns true if the object represents the computer account of a DC.

5.97 IsForwardLinkAttribute

```
procedure IsForwardLinkAttribute (att:ATTRTYP): Boolean
```

The IsForwardLinkAttribute procedure returns true if the given [ATTRTYP](#) represents a **forward link attribute**. Forward link attribute is defined in [\[MS-ADTS\]](#) section 3.1.1.1.6.

5.98 IsGC

```
procedure IsGC(): boolean
```

The IsGC procedure returns true if the DC on which it is called is a global catalog server as defined in [\[MS-ADTS\]](#) section 3.1.1.1.10 or is in a forest that contains only one domain. Otherwise, the procedure returns false.

5.99 IsGetNCChangesPermissionGranted

```
procedure IsGetNCChangesPermissionGranted(  
    msgIn: DRS_MSG_GETCHGREQ_V10) : 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

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
endif

if (fRevealSecret = false) or (fRevealFilteredSet = false)
    return false
else
    return true
endif
endif

```

5.100 IsGUIDBasedDNSName

```

procedure IsGUIDBasedDNSName(o: DSName, instanceName: unicodestring):
    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>".

5.101 IsMemberOfBuiltinAdminGroup

```

procedure IsMemberOfBuiltinAdminGroup(): boolean

```

The IsMemberOfBuiltinAdminGroup procedure returns true if the client security context, which MUST be retrieved using the method described in [\[MS-RPCE\]](#) section 3.3.3.4.3, 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. [\[MS-SAMR\]](#) section 3.1.4.2 describes the accounts included in the built-in Administrators group by default.

5.102 IsRecycleBinEnabled

```

procedure IsRecycleBinEnabled(): boolean

```

Informative summary of behavior: The IsRecycleBinEnabled procedure returns true if the Recycle Bin optional feature is enabled. Otherwise, it returns false. For more information, see [\[MS-ADTS\]](#) sections [3.1.1.9](#), [3.1.1.9.1](#), and [7.1.1.2.1](#)

```

recycleBinFeatureGuid: GUID

```

```

scope: DSNAME
recycleBinFeatureGuid := {766ddcd8-acd0-445e-f3b9-a7f9b6744f2a}
scope := DSName of the nTDSDSA object
return IsOptionalFeatureEnabled (scope, recycleBinFeatureGuid)

```

5.103 IsRevealFilteredAttribute

```

procedure IsRevealFilteredAttribute(
    DRS_MSG_GETCHGREQ_V10 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;

```

5.104 IsRevealSecretRequest

```

procedure IsRevealSecretRequest (DRS_MSG_GETCHGREQ_V10 msgIn)
    : boolean

```

Informative summary of behavior: The IsRevealSecretRequest procedure returns true if the source DC is requesting secret attributes.

```

if AmILHServer() = false then
    if (DRS_WRITE_REP in msgIn.ulFlags) then
        return true
    else
        return false
    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_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;

```

5.105 IsServerExtensionsChanged

```

procedure IsServerExtensionsChanged(
    ServerExtensions: DRS_EXTENSIONS_INT): boolean;

```

The IsServerExtensionsChanged procedure returns true if the supplied extensions are different from the current server extensions. Otherwise, it returns false.

5.106 IsUPNInTrustedForest

```

prodecure IsUPNInTrustedForest(upn: unicodestring): boolean

```

The IsUPNInTrustedForest procedure returns true if the domain containing the account identified by *upn* 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 *upn* parameter is the UPN of an account in a domain.

See section [5.63](#) for the specification of this procedure.

5.107 IsValidServiceName

```
procedure IsValidServiceName(o: DSName, serviceName: uncodestring):  
    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:

1. For GC SPNs, the service name must be the DNS forest name.
2. 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.

5.108 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* (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 KCC noticed the first failure while contacting the DC.
- **FailureCount:** An integer that contains the total number of failures the KCC encountered while contacting the DC.
- **LastResult:** A [DWORD](#) that contains a Windows error code, as specified in [\[MS-ERREF\]](#) section 2.2, 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.

5.109 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.

5.110 LARGE_INTEGER

A concrete type for a 64-bit signed integer, as specified in [\[MS-DTYP\]](#) section 2.3.3.

5.111 LDAP_CONN_PROPERTIES

LDAP_CONN_PROPERTIES is a concrete type that contains bit flags that identify properties of an LDAP connection. The bit flags are presented below in little-endian byte order.

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

X: Unused. MUST be zero and ignored.

BND (0x00000001): A bind has been performed on this LDAP connection.

SSL (0x00000002): The LDAP connection corresponds to a Secure Sockets Layer (SSL) connection.

UDP (0x00000004): The LDAP connection corresponds to a User Datagram Protocol (UDP) connection.

GC (0x00000008): The LDAP connection was made through the GC port.

GSS (0x00000010): The Generic Security Services Application Programming Interface (GSS-API) security package was used for authentication.

NGO (0x00000020): The Simple and Protected GSS-API Negotiation (SPNEGO) security package was used for authentication.

SPL (0x00000040): The LDAP bind corresponds to LDAP simple bind.

MD5 (0x00000080): The Digest-MD5 security package was used for authentication.

SGN (0x00000100): Signing is enabled on the LDAP connection.

SL (0x00000200): Sealing is enabled on the LDAP connection.

5.112 LDAP_SERVER_DIRSYNC_OID LDAP Search Control

This section describes LDAP_SERVER_DIRSYNC_OID control processing. See [\[MS-ADTS\]](#) section 3.1.1.3.4.1.3 for more information.

5.112.1 Abstract Types

5.112.1.1 AttributeList

```
Type AttributeList = [next: AttributeList,
                      value: AttributeListElement]
```

AttributeList is an abstract type that contains information about the attributes of an object. It is a tuple of the following:

next: The next tuple.

value: An attribute type and its value(s).

5.112.1.2 AttributeListElement

```
Type AttributeListElement = [type: LDAPString,  
                             vals: AttributeVals]
```

AttributeListElement is an abstract type that contains information about an attribute of an object. It is a tuple of the following:

type: Attribute type.

vals: Attribute values.

5.112.1.3 AttributeVals

```
Type AttributeVals = [next: AttributeVals,  
                      value: LDAPString]
```

AttributeVals is an abstract type that contains information about the value(s) of an attribute. It is a tuple of the following:

next: The next tuple.

value: An attribute value.

5.112.1.4 Control

```
Type Control = [controlType: LDAPString, criticality: BOOL,  
                controlValue: LDAPString]
```

Control is an abstract type that contains information about an LDAP control. It is a tuple of the following:

controlType: LDAP control OID.

criticality: LDAP control criticality.

controlValue: LDAP control value.

5.112.1.5 DirSyncControlValue

```
Type DirSyncControlValue = [flags: ULONG, size: ULONG, cookie: LDAPString]
```

DirSyncControlValue is an abstract type that contains information about the LDAP_SERVER_DIRSYNC_OID control value. It is a tuple of the following:

flags: Flags, as described in [\[MS-ADTS\]](#) section 3.1.1.3.4.1.3.

size: In the request control value, this field indicates the maximum number of bytes expected in the reply. In the reply control value, it is set to 0.

cookie: A cookie value.

5.112.1.6 DirSyncSearchArg

```
Type DirSyncSearchArg = [pObject: LDAPString, pFilter: LDAPString,  
                          pSelection: ATTRBLOCK, sizeLimit: ULONG]
```

DirSyncSearchArg is an abstract type that contains information about the LDAP search request. It is a tuple of the following:

pObject: The LDAP search base object DN.

pFilter: The string representation, as defined in [\[RFC2254\]](#), of the LDAP search filter.

pSelection: The attribute selection info in the LDAP search request.

sizeLimit: The size limit in the LDAP search request.

5.112.1.7 LDAPString

```
Type LDAPString = [length: ULONG, value: * UCHAR]
```

LDAPString is an abstract type that contains information about a string of unsigned characters. It is a tuple of the following:

length: The number of unsigned characters in **value**.

value: The string of unsigned characters that may contain null characters.

5.112.1.8 SearchResultEntry

```
Type SearchResultEntry = [objectName: LDAPString,  
                          attributes: AttributeList]
```

SearchResultEntry is an abstract type that contains information about a search result entry. It is a tuple of the following:

objectName: The object DN.

attributes: The list of the attributes of **object**.

5.112.1.9 SearchResultEntryList

```
Type SearchResultEntryList = [next: SearchResultEntryList,  
                              entry: SearchResultEntry]
```

SearchResultEntryList is an abstract type that contains information about search result entries. It is a tuple of the following:

next: The next tuple.

entry: An entry in the search result.

5.112.2 Concrete Types

5.112.2.1 Cookie

The **Cookie** structure is a concrete type that contains information about the cookie in the LDAP_SERVER_DIRSYNC_OID control value.

```
typedef struct {
    UCHAR signature[4];
    ULONG version;
    FILETIME creationTime;
    ULONGLONG reserved;
    ULONG utdVectorSize;
    USN_VECTOR usnVector;
    UUID uuidSourceDsaInvocationID;
    UCHAR utdVector[variable];
} Cookie;
```

signature: Cookie signature.

version: The version number.

creationTime: The creation time.

reserved: Unused.

utdVectorSize: The up-to-date vector size.

usnVector: The USN vector.

uuidSourceDsaInvocationID: The invocation ID (a UUID) of the source DSA.

utdVector: The up-to-date vector.

5.112.3 ProcessDirSyncSearchRequest

```
ProcessDirSyncSearchRequest (
    [in] searchArg: DirSyncSearchArg,
    [in] dirSyncControlValue: DirSyncControlValue,
    [out] searchResultEntryList: SearchResultEntryList,
    [out] dirSyncResponseControl: Control
) : ULONG
```

Informative summary of behavior: The ProcessDirSyncSearchRequest procedure processes an LDAP search request with an LDAP_SERVER_DIRSYNC_OID control. It creates a list of search result entries and the LDAP_SERVER_DIRSYNC_OID response control.

```
err: ULONG
msgIn: DRS_MSG_GETCHGREQ_V10
msgOut: DRS_MSG_GETCHGREPLY_V6
filter: LDAPString
replFlags: ULONG

/* Transform the LDAP search request with LDAP_SERVER_DIRSYNC_OID control to
   a replication GetChanges request. */
```

```

err := DirSyncReqToGetChgReq(searchArg, dirSyncControlValue, msgIn)
if err ≠ 0 then
    return err
endif

replFlags := dirSyncControlValue.flags

/* Perform access checks unless client has specified object-level security */

if not (LDAP_DIRSYNC_OBJECT_SECURITY in replFlags) then
    err := SecurityCheckForChanges(msgIn)
    if err ≠ 0 then
        return err
    endif
endif

filter := searchArg.pfilter
/* Perform normal replication (Get replication changes). */

err := GetReplChanges(null, filter, replFlags, msgIn, msgOut)
if err ≠ 0 then
    return err
endif

/* Transform the replication GetChanges reply to reply for the LDAP search
   request with LDAP_SERVER_DIRSYNC_OID control. */

err := GetChgReplyToSearchResult(msgOut, searchResultEntryList,
                                dirSyncResponseControl)
if err ≠ 0 then
    return err
endif

return 0 /* success */

```

5.112.4 DirSyncReqToGetChgReq

```

procedure DirSyncReqToGetChgReq(
    [in] searchArg: DirSyncSearchArg,
    [in] dirSyncControlValue: DirSyncControlValue,
    [out] nativeRequest: DRS_MSG_GETCHGREQ_V10
): ULONG

```

Informative summary of behavior: The DirSyncReqToGetChgReq procedure transforms the received LDAP search request with an LDAP_SERVER_DIRSYNC_OID control message into a [DRS_MSG_GETCHGREQ_V10](#) request.

```

baseObjectDsName: DSNAME
replFlags: ULONG
ulSizeLimit: ULONG
ulMaxBytes: ULONG
errCode: ULONG

/* Validate input */
/* If the base of the search is not the root of an NC,

```

```

    return unwillingToPerform */

baseObjectDsName := GetDSNameFromDN (searchArg.pObject)
if baseObjectDsName ≠ GetObjectNC(baseObjectDsName) then
    return unwillingToPerform
endif

nativeRequest.pNC := baseObjectDsName
nativeRequest.ulFlags := 0

nativeRequest.ulMoreFlags := 0

replFlags := dirSyncControlValue.flags
if (LDAP_DIRSYNC_ANCESTORS_FIRST_ORDER in replFlags) then
    nativeRequest.ulFlags := nativeRequest.ulFlags + {DRS_GET_ANC}
endif

ulSizeLimit := 1000
if (searchArg.sizeLimit) then
    ulSizeLimit := min(searchArg.sizeLimit, ulSizeLimit)
endif

nativeRequest.cMaxObjects = max (ulSizeLimit, 100)

ulMaxBytes := dirSyncControlValue.size
if (ulMaxBytes = 0) then
    ulMaxBytes := 1024*1024
endif
nativeRequest.cMaxBytes := ulMaxBytes

cookie := dirSyncControlValue.cookie
errCode := GetUsnUtdVectorFromCookie (cookie, nativeRequest)
if (errCode ≠ 0) then
    return errCode
endif

/* Handle attribute selection */
copy the list of required attributes, if present, from searchArg.pSelection
to nativeRequest.pPartialAttrSet

return 0 /* success */

```

5.112.5 GetChgReplyToSearchResult

```

procedure GetChgReplyToSearchResult (
    [in] chgReply: DRS_MSG_GETCHGREPLY_V6,
    [out] searchResultEntryList: SearchResultEntryList,
    [out] dirSyncResponseControl: Control)

```

Informative summary of behavior: The GetChgReplyToSearchResult procedure generates a list of search result entries (*searchResultEntryList*) and an LDAP_SERVER_DIRSYNC_OID response control (*dirSyncResponseControl*) from a DRS_MSG_GETCHGREPLY_V6 structure.

The arguments to this procedure are as follows:

- *chgReply*: A DRS_MSG_GETCHGREPLY_V6 message generated by the GetReplChanges function.

- *searchResultEntryList*: A list of search result entries.
- *dirSyncResponseControl*: The control that is sent back to the client.

```

objCount: ULONG
valCount: ULONG
replEntinfList: REPLENTINFLIST
responseControlValue: DirSyncControlValue
objectGuid: GUID
attrType: ATTRTYP
minCookieLength: ULONG
utdVectorSize: ULONG

objCount := chgReply.cNumObjects
valCount := chgReply.cNumValues

/* Process object updates. */

replEntinfList := chgReply.pObjects
while (not replEntinfList = null)
    TransformEntinfToSearchEntry(replEntinfList.Entinf,
                                searchResultEntryList.entry)

    Add objectGUID attribute with value =
        replEntinfList.Entinf.pName.Guid to
        searchResultEntryList.entry

    Add parentGUID attribute with value =
        replEntinfList.pParentGuid to
        searchResultEntryList.entry

    replEntinfList := replEntinfList.pNextEntInf
    searchResultEntryList := searchResultEntryList.next
endwhile

/* Process value updates. */

foreach distinct object GUID objectGuid, referred by REPLVALINF,
in chgReply.rgvalues

    TransformReplValInfListToSearchEntry(
        REPLVALINF.pObject, chgReply, searchResultEntryList.entry)

    Add objectGUID attribute with value =
        REPLVALINF.pObject.Guid to
        searchResultEntryList.entry

```

```

        searchResultEntryList := searchResultEntryList.next
    endfor

    /* Construct LDAP_SERVER_DIRSYNC_OID response control */

    dirSyncResponseControl.value.controlType := LDAP_SERVER_DIRSYNC_OID
    dirSyncResponseControl.value.criticality := false

    responseControlValue := GetResponseDirSyncControlValue (chgReply)
    dirSyncResponseControl.value.controlValue :=
        BER encoding of the responseControlValue

    return

```

5.112.6 TransformEntinfToSearchEntry

```

procedure TransformEntinfToSearchEntry (
    [in] entinf: ENTINF
    [out] searchResultEntry: SearchResultEntry
)

```

Informative summary of behavior: The TransformEntinfToSearchEntry procedure transforms an ENTINF structure (*entinf*) into a SearchResultEntry structure (*searchResultEntry*).

```

attrList: AttributeList
attrVals: AttributeVals

TransformDSNameToLdapDN (entinf.pName, searchResultEntry.objectName)

attrList := searchResultEntry.attributes

foreach i in [0 .. entInf.AttrBlock.attrCount -1] do

    attrList.value.type := LDAPDisplayNameFromAttrTyp (
        entInf.AttrBlock.pAttr[i].attrTyp)

    attrVals := attrList.value.vals

    foreach j in [0 .. entInf.AttrBlock.pAttr[i].AttrVal.valCount -1] do

        attrVals.value := ValueFromATTRVAL(
            entInf.AttrBlock.pAttr[i].AttrVal.pAVal[j],
            Syntax(entInf.AttrBlock.pAttr[i].attrTyp),
            dc.prefixTable)
    endfor
endfor

```

```

        attrVals := attrVals.next
    endfor
    attrList := attrList.next
endfor

return

```

5.112.7 TransformReplValInfListToSearchEntry

```

procedure TransformReplValInfListToSearchEntry (
    [in] o: DSNAME,
    [in] chgReply: DRS_MSG_GETCHGREPLY_V6,
    [out] searchResultEntry: SEARCH_RESULT_ENRty
)

```

Informative summary of behavior: The TransformReplValInfListToSearchEntry procedure transforms, for object *o*, the attribute values in the REPLVALINF list (*chgReply.rgValues*) to an AttributeList structure in *searchResultEntry*.

```

attrList: AttributeList
attrVals: AttributeVals
attributeType: ATTRTYP

TransformDSNameToLdapDN (o, searchResultEntry.objectName)

attrList := searchResultEntry.attributes

foreach distinct attribute attrType of the object o in
    chgReply.rgvalues

    attrList.value.type := LDAPDisplayNameFromAttrTyp (attrType)
    attrVals := attrList.value.vals

    foreach attribute value replAttrVal of the attribute attrType
        of the object o in chgReply.rgvalues

        attrVals.value := ValueFromATTRVAL(replAttrVal,
                                            attrType,
                                            dc.prefixTable)

        attrVals := attrVals.next
    endfor

    attrList := attrList.next
endfor

```



```
return
```

5.112.8 TransformDSNameToLdapDN

```
procedure TransformDSNameToLdapDN (  
    [in] dsName: DSNAME  
    [out] dn: LDAPString  
)
```

Informative summary of behavior: The TransformDSNameToLdapDN procedure transforms a DSNAME to an LDAPString DN.

```
dn.length := dsName.NameLen  
dn.value := dsName.StringName
```

```
return
```

5.112.9 LDAPDisplayNameFromAttrTyp

```
Procedure LDAPDisplayNameFromAttrTyp (  
    [in] attrTyp: ATTRTYP  
) : LDAPString
```

Informative summary of behavior: The LDAPDisplayNameFromAttrTyp procedure transforms an attribute type (ATTRTYP) to an attribute name that is used by the LDAP clients.

```
oid : OID  
attributeDisplayName: LDAPString  
attrObj: DSName  
  
oid := OidFromAttid(dc.prefixTable, attrTyp)  
attrObj := select o from subtree SchemaNC() where  
    (attributeSchema in o!objectClass) and  
    (o!attributeID = oid)  
attrDisplayname := attrObj!LDAPDisplayName  
  
return attrDisplayName
```

5.112.10 GetResponseDirSyncControlValue

```
procedure GetResponseDirSyncControlValue (  
    [in] chgReply: DRS_MSG_GETCHGREPLY_V6  
) : DirSyncControlValue
```

Informative summary of behavior: The `GetResponseDirSyncControlValue` procedure creates an `LDAP_SERVER_DIRSYNC_OID` control value, to be returned in the response `LDAP_SERVER_DIRSYNC_OID` control, from a `DRS_MSG_GETCHGREPLY_V6` structure (*chgReply*).

The arguments to this procedure are as follows:

- *chgReply*: A `DRS_MSG_GETCHGREPLY_V6` message generated by the `GetReplChanges` function.

```
replControlValue: DirSyncControlValue
minCookieLength: ULONG
utdVectorSize: ULONG
pUpToDateVecSrcV1: UPTODATE_VECTOR_V1_EXT

/* Construct LDAP_SERVER_DIRSYNC_OID response control value */

replControlValue.flag := chgReply.fMoreData
replControlValue.size := 0 /* must be ignored by the client */

/* minimum possible cookie size in bytes; that is, the size of a cookie
when a UTD vector is not present in the cookie. */

minCookieLength := 17*4

pUpToDateVecSrcV1 := If necessary, convert chgReply.pUpToDateVecSrc (of
    type UPTODATE_VECTOR_V2_EXT) to UPTODATE_VECTOR_V1_EXT by
    creating a new UPTODATE_VECTOR_V1_EXT with a UPTODATE_CURSOR_V1
    cursor for each UPTODATE_CURSOR_V2 cursor, ignoring the
    timeLastSyncSuccess field in UPTODATE_CURSOR_V2.

utdVectorSize := 16 /* offsetof(UPTODATE_VECTOR,V1.rgCursors[0]) */
    + (pUpToDateVecSrcV1.V1.cNumCursors *
        sizeof(UPTODATE_CURSOR_V1))

replControlValue.cookie.length := minCookieLength + utdVectorSize
replControlValue.cookie.value.signature := "SDSM"
replControlValue.cookie.value.version := 3
replControlValue.cookie.value.creationTime := current system time
replControlValue.cookie.value.utdVectorSize := utdVectorSize
replControlValue.cookie.value.usnVector := chgReply.usnvecTo
replControlValue.cookie.value.uuidSourceDsaInvocationID :=
    chgReply.uuidInvocIdSrc
copy utdVectorSize bytes from pUpToDateVecSrcV1 to
    replControlValue.cookie.value.utdVector

return replControlValue
```

5.112.11 GetUsnUtdVectorFromCookie

```
procedure GetUsnUtdVectorFromCookie(  
    [in] replCookie: LDAPString,  
    [in/out] nativeRequest: DRS_MSG_GETCHGREQ_V10  
): ULONG
```

Informative summary of behavior: The GetUsnUtdVectorFromCookie procedure extracts the USN vector and the UTD vector from an LDAP_SERVER_DIRSYNC_OID control value (*replCookie*) and sets the USN vector **from** (*nativeRequest.usnvecFrom*) and UTD vector **destination** (*nativeRequest.pUpToDateVecDest*).

```
utdVectorSize: ULONG  
minCookieLength: ULONG  
  
/* minimum possible cookie size in bytes; that is, the size of a cookie  
when a UTD vector is not present in the cookie. */  
  
minCookieLength := 17*4  
  
/* Validate cookie, and extract USN and UTD vectors. */  
If (replCookie.length ≠ 0) then  
    If (replCookie.length < minCookieLength)  
        or  
        (replCookie.value.signature ≠ "SDSM")  
        or  
        (replCookie.value.version ≠ 3)  
  
        return protocolError  
    endif  
  
    utdVectorSize := replCookie.value.utdVectorSize  
    if (utdVectorSize < sizeof(UPTODATE_VECTOR_V1_EXT)  
        or  
        replCookie.length < minCookieLength +  
            utdVectorSize) then  
        utdVectorSize := 0  
    endif  
  
    if (replCookie.value.uuidSourceDsaInvocationId =  
        DSAObj().!invocationId) then  
        nativeRequest.usnvecFrom := replCookie.value.usnVector  
    endif  
  
    if (utdVectorSize > 0) then  
        Copy utdVectorSize bytes from replCookie.value.utdVector  
        to nativeRequest.pUpToDateVecDest
```

```

/* some more validation */
if (nativeRequest.pUpToDateVecDest.dwVersion != 1
    or
    16 /* (offsetof(UPTODATE_VECTOR,V1.rgCursors[0]) */
    + (nativeRequest.pUpToDateVecDest.V1.cNumCursors
        * sizeof(UPTODATE_CURSOR_V1))) != utdVectorSize
    or
    replCookie.length != minCookieLength
        + utdVectorSize) then
    return protocolError
endif
Endif
endif

return 0 /* success */

```

5.112.12 SecurityCheckForChanges

```

procedure SecurityCheckForChanges(
    [in] msgIn: DRS_MSG_GETCHGREQ_V10
): ULONG

```

Informative summary of behavior: The SecurityCheckForChanges procedure checks whether an LDAP_SERVER_DIRSYNC_OID control client has access rights to read the changes in an NC (*msgIn.pNC*).

```

if AccessCheckCAR(msgIn.pNC, Ds-Replication-Get-Changes) = false then
    return insufficientAccessRights
endif

if msgIn.pPartialAttrSet.cAttrs != 0 and
    IsFilteredAttributePresent(msgIn.pPartialAttrSet) = true and
    AccessCheckCAR(msgIn.pNC,
        Ds-Replication-Get-Changes-In-Filtered-Set) = false and
    AccessCheckCAR(msgIn.pNC,
        Ds-Replication-Get-Changes-All) = false
then
    return insufficientAccessRights
endif

return 0 /* success */

```

5.112.13 IsFilteredAttributePresent

```

procedure IsFilteredAttributePresent(
    [in] attrVec: PARTIAL_ATTR_VECTOR_V1_EXT
): boolean

```

Informative summary of behavior: The IsFilteredAttributePresent procedure returns true if an attribute from the filtered set is present in *attrVec*; otherwise, it returns false.

```

filteredAttrSet: sequence of ATTRTYP
i: int

filteredAttrSet := GetFilteredAttributeSet()

for i := 0 to attrVec.cAttrs - 1
    if attrVec.rgPartialAttr[i] in filteredAttrSet then
        return true
    endif
endfor

return false

```

5.113 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 properties of the connection.
- **totalRequests:** An integer that contains the total number of LDAP requests processed on the connection.
- **userName:** A *unicodestring* (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.lapConnections](#), which maintains the DC's LDAPConnections state.

5.114 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.

5.115 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](#), $z.flIsPresent$ is TRUE if $x.timeDeleted$ is 0 and FALSE if $x.timeDeleted$ is nonzero.

5.116 LocalAttidFromRemoteAttid

```
procedure LocalAttidFromRemoteAttid(  
    remotePT: PrefixTable,  
    remoteAttid : ATTRTYP) : ATTRTYP
```

Informative summary of behavior: The LocalAttidFromRemoteAttid procedure 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)
```

5.117 LONG

A concrete type for a 32-bit, signed integer, as specified in [\[MS-DTYP\]](#) section 2.2.27.

5.118 LONGLONG

A concrete type for a 64-bit, signed integer, as specified in [\[MS-DTYP\]](#) section 2.2.28.

5.119 LPWSTR

A concrete type for a pointer to a string of double-byte Unicode characters, as specified in [\[MS-DTYP\]](#) section 2.2.35.

5.120 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 [5.16.4](#) for the specification of this procedure.

5.121 MakeProxyValue

```
procedure MakeProxyValue(  
  dnPart: DSName,  
  proxyType: DWORD,  
  proxyEpoch: DWORD): DNBinary
```

The MakeProxyValue procedure constructs and returns a value in the [proxiedObjectName](#) format (section [5.153](#)) from the provided parts. Let d be the returned [DNBinary](#). d.dn equals dnPart and d.binary is constructed from proxyType and proxyEpoch.

5.122 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
```

5.123 MD5_CTX

MD5_CTX is an abstract type defined in [\[RFC1321\]](#).

5.124 MD5Final

```
procedure MD5Final(var context: MD5_CTX)
```

The MD5Final procedure performs the MD5Final algorithm, as specified in [\[RFC1321\]](#).

5.125 MD5Init

```
procedure MD5Init(var context: MD5_CTX)
```

The MD5Init procedure performs the MD5Init algorithm, as specified in [\[RFC1321\]](#).

5.126 MD5Update

```
procedure MD5Update(  
  var context: MD5_CTX,  
  input: sequence of BYTE,  
  inputLen: integer)
```

The MD5Update procedure performs the MD5Update algorithm, as specified in [\[RFC1321\]](#).

5.127 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 the uuidDsa to the returned [UPTODATE VECTOR V1 EXT](#) with a corresponding USN value such that the USN is the smaller of the USNs corresponding to the uuidDsa in utd1 and utd2.

5.128 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 terminating null 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	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
mtx_namelen																															
mtx_name (variable length) ...																															

5.129 NCType Bits

Bit flags, presented in little-endian byte order, describing NCType.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	F	G	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
					P	P	P																								

X: Unused. MUST be zero and ignored.

SP (NCT_SPECIAL_SECRET_PROCESSING, 0x00000001): The NC replica requests special secret processing while replicating objects.

GP (NCT_GC_PARTIAL, 0x00000002): Objects in the NC replica can only have attributes that are specified in the **GC partial attribute set**.

FP (NCT_FILTERED_ATTRIBUTE_SET, 0x00000004): Objects in the NC replica do not have attributes defined in the **filtered attribute set**.

5.130 NetworkAddress

NetworkAddress is an abstract type for the transport-specific address of a DC represented as a string. For the SMTP transport, the address is an SMTP address (as specified in [\[RFC2821\]](#) and [\[MS-SRPL\]](#)). For the RPC transport in AD DS, the NetworkAddress is a fully qualified DNS name corresponding to the DC.

For the RPC transport in AD LDS, the NetworkAddress is a UTF-8 string in the following format:

- <DC-name>:<DC-identifier>

In the preceding format:

- <DC-name> is an IP address in the UTF-8 format, a fully qualified DNS name, or a NetBIOS name.
- <DC-identifier> is either a GUID or an integer. The GUID corresponds to the objectGUID attribute of the DC's [nTDSDSA](#) object. The integer is the **IdapPort** attribute of the DC's [nTDSDSA](#) object.
- The colon (":") is the literal separator between the DC-name and the DC-identifier.

A `NetworkAddress` is stored as an `mtx_name` within an [MTX_ADDR](#) structure or in a location that is pointed to by the `cbpszInstanceOffset` member of the [DSA_RPC_INST](#) structure, which in turn is stored within a [REPS_TO](#) structure. The concrete representation of `NetworkAddress` in these concrete structures is the same as the abstract representation described above.

5.131 NewPrefixTable

```
procedure NewPrefixTable() : PrefixTable
```

The `NewPrefixTable` procedure creates a new [PrefixTable](#) that contains a set of default prefixes. See section [5.16.4](#) for the specification of this procedure.

5.132 Nt4ReplicationState

`Nt4ReplicationState` is an abstract type for the replication state for Microsoft Windows NT® 4.0 operating system 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.

5.133 NT4SID

The **NT4SID** structure defines a concrete type for a 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.

5.134 NTSAPI_CLIENT_GUID

`NTDSAPI_CLIENT_GUID` is a value of type GUID that is defined as {e24d201a-4fd6-11d1-a3da-0000f875ae0d}.

5.135 NTDSTRANSPORT_OPT Values

The valid system flags used on directory objects are specified in [\[MS-ADTS\]](#) section 7.1.1.2.2.3.1.

5.136 NULLGUID

NULLGUID is a value of type [GUID](#) that is entirely zero, that is, {00000000-0000-0000-0000-000000000000}.

5.137 ObjExists

```
procedure ObjExists(dsName: DSName): boolean
```

The ObjExists procedure returns true if *dsName* identifies an object in some NC replica hosted by the server.

```
rt: DSName
rt:= select one v from all where v = dsName
if (rt = null) then
    return false
else
    return true
endif
```

5.138 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".

5.139 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: An array of bytes that constitute an OID or a prefix of an OID.

5.140 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 [5.16.4](#) for the specification of this procedure.

5.141 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.

5.142 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: Unused. MUST be 0 and ignored.

cAttrs: The number of attributes in the rgPartialAttr array.

rgPartialAttr: The attributes in the set.

5.143 partialAttributeSet

The abstract, nonreplicated, 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.

5.144 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
```

5.145 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	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
version																															
size																															
flag																															
pas (variable)																															
...																															

version (4 bytes): The version of the structure; MUST be 1.

size (4 bytes): The size of the entire structure.

flag (4 bytes): Unused. MUST be 0 and ignored.

pas (variable): A [PARTIAL ATTR VECTOR V1 EXT](#) structure, which specifies the additional attributes being requested as part of a PAS cycle.

5.146 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 Microsoft Windows NT® 4.0 operating system 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.

5.147 PerformAddOperation

```
procedure PerformAddOperation(
    data: ENTINF,
    var newObjectName: DSName,
    prefixTable: PrefixTable,
    boolean: fAsOriginating): 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 [LocalAttrFromRemoteAttrId](#)(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 *fAsOriginating* is true, then add the object as an originating update. See [\[MS-ADTS\]](#) section 3.1.1.1.9.

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.

5.148 PerformAddOperationAsSystem

```
procedure PerformAddOperationAsSystem(  
    data: ENTINF,  
    prefixTable: PrefixTable,  
    var newObjectName: DSNAME): integer
```

The *PerformAddOperationAsSystem* procedure 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.

5.149 PrefixTable

PrefixTable is an abstract type for a prefix table. See section [5.16.4](#) for the specification of this type.

5.150 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.

5.151 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.

5.152 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 number of items in the **rgMetaData** array.

rgMetaData: An array of attribute stamps.

5.153 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 for the first DWORD.

Symbolic name	Value	Meaning
PROXY_TYPE_MOVED_OBJECT	0x0	An object that was cross-NC moved.
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.

5.154 RDN

RDN is an abstract type for representing the relative distinguished name (RDN) (as specified in [\[RFC2253\]](#)).

5.155 rdnType

rdnType 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.

5.156 RecycleObj

```
procedure RecycleObj(o: DSName)
```

The RecycleObj procedure transforms, as described in [\[MS-ADTS\]](#) section 3.1.1.5.5, the object whose DSName is *o* into a **recycled-object**. All appropriate attributes (possibly including [distinguishedName](#)) are changed or removed from the object to conform to the invariants of [\[MS-ADTS\]](#) section 3.1.1.5.5. Any changes that need to be made to the object are performed as an originating update, except for changes required to remove linked attribute values, which are simply removed from the directory. Attributes and values that already conform to the invariants are not changed. See [\[MS-ADTS\]](#) section 3.1.1.1.9 for more information on originating updates.

5.157 RemoveObj

```
procedure RemoveObj(o: DSName,treeDeletion: boolean): ULONG
```

The RemoveObj procedure performs a delete operation on the object whose [DSName](#) is *o*. If the value of parameter *treeDeletion* is true, then the tree-delete variation of the operation is performed. As described in [\[MS-ADTS\]](#) section 3.1.1.5.5, the delete operation transforms the targeted object into a **deleted-object** or a tombstone, depending on the state of the Recycle Bin optional feature. The tree-delete operation performs, as described in [\[MS-ADTS\]](#) section 3.1.1.5.5.7.3, a delete operation on all objects in the subtree rooted at the target object. All appropriate attributes (possibly including [distinguishedName](#)) are changed or removed from the deleted objects to conform to the invariants of [\[MS-ADTS\]](#) section 3.1.1.5.5. Any changes that need to be made to the objects are performed as an originating update, except for changes required to remove linked attribute values, which are simply removed from the directory. Attributes and values that already conform to the invariants are not changed. See [\[MS-ADTS\]](#) section 3.1.1.1.9 for more information on originating updates. If the delete operation succeeds, 0 is returned. Otherwise, this procedure returns an error code, as specified in [\[MS-ADTS\]](#) section 3.1.1.5.5, that indicates the reason for the failure.

5.158 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 $[0 \dots \text{Entinf.AttrBlock.attrCount}]$, the stamp for the attribute described by `Entinf.AttrBlock.pAttri` is `pMetaDataExti.rgMetaData[i]`.

5.159 ReplicatedAttributes

```
procedure ReplicatedAttributes(): Set of ATTRTYP
```

The `ReplicatedAttributes` procedure returns the set of [ATTRTYP](#) of all attributes in the schema excluding nonreplicated attributes. See [\[MS-ADTS\]](#) section 3.1.1.2.3 for more information on nonreplicated attributes.

5.160 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.

5.161 REPLTIMES

The **REPLTIMES** structure defines a concrete type for times at which periodic replication occurs.

```
typedef struct {
```

```

    UCHAR rgTimes[84];
} REPLTIMES;

```

rgTimes: A byte array of length 84 that is used to set periodic replication times. Each bit in this byte array represents a 15-minute period for which replication can be scheduled within a one-week period. The replication schedule begins on Sunday 12:00:00 AM UTC. Each byte in the array represents a two-hour period of a week in ascending order, starting Sunday 12:00:00 AM UTC. The most significant bit of a byte represents the earliest 15-minute period in the two-hour period, and the rest of the bits in the byte represent their respective 15-minute periods in this order.

The following diagram shows an alternative representation of this structure.



5.162 replUpToDateVector, ReplUpToDateVector

The nonreplicated 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 [replUpToDateVector](#) attribute. Reading the [replUpToDateVector](#) attribute produces one or more ReplUpToDateVector values.

The ReplUpToDateVector type 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*.

5.163 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 if and only if the link value has been removed from the attribute.

MetaData: The stamp associated with the link value.

5.164 REPS_FROM

The nonreplicated, multivalued attribute [repsFrom](#) is an optional attribute on the root object of every NC replica. It is stored with the structure of the REPS_FROM concrete type, which is represented by the following 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
dwVersion																															
cb																															
cConsecutiveFailures																															
timeLastSuccess																															
...																															
timeLastAttempt																															
...																															
ulResultLastAttempt																															
cbOtherDraOffset																															
cbOtherDra																															
ulReplicaFlags																															
rtSchedule																															

...
...
...
...
...
...
...
...
(rtSchedule cont'd for 13 rows)
usnVec
...
...
...
...
...
...
...
uuidDsaObj
...
...
...
uuidInvocId
...
...
...
uuidTransportObj

...
...
...
dwReserved
cbPasDataOffset
data (variable)
...

dwVersion (4 bytes): The version of this structure. The value must be 1 or 2. [<45>](#)

cb (4 bytes): The total number of bytes in the REPS_FROM structure.

cConsecutiveFailures (4 bytes): An unsigned long that contains the number of consecutive failures that have occurred while replicating from the source DC.

timeLastSuccess (8 bytes): A [DSTIME](#) that contains the time of the last successful replication cycle with the source DC.

timeLastAttempt (8 bytes): A [DSTIME](#) that contains the time of the last replication attempt with the source DC.

ulResultLastAttempt (4 bytes): A Win32 error code, as specified in [\[MS-ERREF\]](#) section 2.2, that represents the result of the last replication attempt with the source DC.

cbOtherDraOffset (4 bytes): The offset from the start of the structure to a location in the data field, specifying the start of a structure that contains a [NetworkAddress](#) for the source DC. If **dwVersion** is 1, it is an [MTX_ADDR](#) structure. If **dwVersion** is 2, it is a [DSA_RPC_INST](#) structure.

cbOtherDra (4 bytes): The size of the structure pointed to by **cbOtherDraOffset**.

ulReplicaFlags (4 bytes): A [ULONG](#). This field contains a set of [DRS_OPTIONS](#) that are applicable when replicating from the source DC.

rtSchedule (84 bytes): 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 (24 bytes): 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 (16 bytes): A [GUID](#) that is the DSA GUID of the source DC.

uuidInvocId (16 bytes): A [GUID](#) that contains the invocation ID of the source DC.

uuidTransportObj (16 bytes): A [GUID](#) that contains the [objectGUID](#) of the [interSiteTransport](#) object that corresponds to the transport used for communication with the source DC.

dwReserved (4 bytes): Unused. MUST be 0 and ignored.

cbPasDataOffset (4 bytes): The offset from the start of the structure to a location in the data field, specifying the start of a PAS_DATA structure.

data (variable): The storage for the rest of the structure. The structures pointed to by **cbOtherDraOffset** and **cbPasDataOffset** are packed into this field and can be referenced using the offsets.

5.165 REPS_TO

The nonreplicated, multivalued attribute [repsTo](#) is an optional attribute on the root object of every NC replica. It is stored with the structure of the REPS_TO concrete type, which is represented by the following 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
dwVersion																															
cb																															
cConsecutiveFailures																															
timeLastSuccess																															
...																															
timeLastAttempt																															
...																															
ulResultLastAttempt																															
cbOtherDraOffset																															
cbOtherDra																															
ulReplicaFlags																															
rtSchedule																															
...																															
...																															

...
...
...
...
...
(rtSchedule cont'd for 13 rows)
usnVec
...
...
...
...
...
...
uuidDsaObj
...
...
...
uuidInvocId
...
...
...
uuidTransportObj
...
...

...
dwReserved
cbPasDataOffset
data (variable)
...

dwVersion (4 bytes): The version of this structure. The value must be 1 or 2. [<46>](#)

cb (4 bytes): The total number of bytes in the REPS_TO structure.

cConsecutiveFailures (4 bytes): An unsigned long that contains the number of unsuccessful consecutive attempts to send a replication notification to the DC identified by uuidDsaObj.

timeLastSuccess (8 bytes): 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 (8 bytes): 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 (4 bytes): 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 of 0 if the last notification was sent successfully or a Windows error code (as specified in [\[MS-ERREF\]](#) section 2.2) otherwise.

cbOtherDraOffset (4 bytes): The offset from the start of the structure to a location in the data field, specifying the start of a structure that contains a [NetworkAddress](#). If **dwVersion** is 1, it is an [MTX_ADDR](#) structure. If **dwVersion** is 2, it is a [DSA_RPC_INST](#) structure.

cbOtherDra (4 bytes): The size of the structure pointed to by **cbOtherDraOffset**.

ulReplicaFlags (4 bytes): A [ULONG](#). This set contains DRS_WRIT_REP (section [5.40](#)) if this replica is writable. This set never contains any other elements.

rtSchedule (84 bytes): A [REPLTIMES](#) structure. Not used.

usnVec (24 bytes): A [USN_VECTOR](#) structure. Not used.

uuidDsaObj (16 bytes): A GUID. A DSA GUID that identifies a DC.

uuidInvocId (16 bytes): A GUID. Not used.

uuidTransportObj (16 bytes): A GUID. Not used.

dwReserved (4 bytes): Unused. MUST be 0 and ignored.

cbPasDataOffset (4 bytes): Not used.

data (variable): The storage for the rest of the structure. The structure pointed to by **cbOtherDraOffset** is packed into this field and can be referenced using the offset.

5.166 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](#) that corresponds to cbOtherDraOffset and cbOtherDra in [REPS_FROM](#). This is a [NetworkAddress](#) of the DC.

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 replicate the NC replica from the DC identified by uuidDsa, as defined by the periodic replication schedule.
- DRS_MAIL_REP: Replicate the NC replica from the DC identified by uuidDsa via SMTP (see [\[MS-SRPL\]](#)).
- DRS_DISABLE_AUTO_SYNC: Disable notification-based replication of the NC replica from the DC identified by uuidDsa.
- DRS_DISABLE_PERIODIC_SYNC: Disable 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](#) that 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 cConsecutiveFailures 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.pas.cAttrs is 0, set cbPasDataOffset to 0.

5.167 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 conversions from [REPS_TO](#) specified below. 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](#) that corresponds to cbOtherDraOffset and cbOtherDra in [REPS_TO](#). This is the [NetworkAddress](#) of a DC.
- **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 when an attempt was made to send a replication notification to the DC identified by uuidDsa, or 0 if no attempt has been made.
- **timeLastSuccess:** A [DSTIME](#) that corresponds to timeLastSuccess in [REPS_TO](#). Contains the time when the last successful replication notification to the DC identified by uuidDsa was sent, or 0 if no replication notification has been 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.

5.168 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 2.4.2 and [\[MS-WSO\]](#) section 3.1.2.1.3.

5.169 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-Manage-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 non-overlapping sets, there is no ambiguity in having one type represent rights of both kinds.

5.170 RIGHT Values

The valid access rights used in **ACLs** in security descriptors are defined in [\[MS-ADTS\]](#) section 5.1.3.2.

5.171 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.

5.172 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.

5.173 sAMAccountType Values

sAMAccountType values describe information about various account type objects. See [\[MS-SAMR\]](#) section 2.2.1.9 for descriptions of these values

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_MACHINE_ACCOUNT	0x30000001
SAM_TRUST_ACCOUNT	0x30000002

Only the values used by this protocol are contained in this table.

5.174 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 number of items in the **pPrefixEntry** array.

pPrefixEntry: An array of [PrefixTableEntry](#) items in the table.

5.175 SchemaNC

```
procedure SchemaNC(): DSName
```

The SchemaNC procedure returns the [DSName](#) of [dc.schemaNC](#).

5.176 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
```

5.177 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.

5.178 SID

A concrete type for the Microsoft Windows NT® operating system **SID** structure, as specified in [\[MS-DTYP\]](#) section 2.4.2.

5.179 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 [\[MS-WSO\]](#) section 3.1.2.1.3 for the string representation of a SID.

5.180 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

```

5.181 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.

5.182 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 [\[MS-WSO\]](#) section 3.1.2.1.3 for the string representation of a SID.

5.183 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.

5.184 Syntax

```

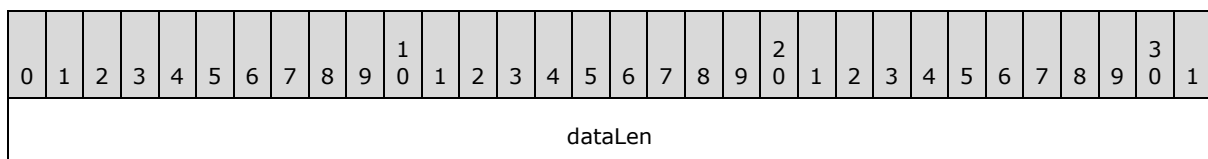
procedure Syntax(attr: ATTRTYP): AttributeSyntax

```

The Syntax procedure returns the syntax of the attribute attr.

5.185 SYNTAX_ADDRESS

The SYNTAX_ADDRESS packet is the concrete type for a sequence of bytes or Unicode characters.



byteVal (variable)
...

dataLen (4 bytes): The size of the entire structure (including this field), in bytes.

byteVal (variable): 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;
```

5.186 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 (4 bytes): The length of the structure, in bytes, up to and including the field StringName.

SidLen (4 bytes): 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 (16 bytes): The value of the object's [objectGUID](#) attribute specified as a GUID structure, which is defined in [\[MS-DTYP\]](#) section 2.3.2. If the values for all fields in the GUID structure are zero, this indicates that the SYNTAX_DISTNAME_BINARY does not identify the [objectGUID](#) value of the directory object.

Sid (28 bytes): The value of the object's [objectSid](#) attribute, its security identifier (see [\[MS-WSO\]](#) section 3.1.2.1.3), 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 (4 bytes): The number of characters in the StringName field, not including the terminating null character, 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 (variable): 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 terminating null character. Each Unicode value is encoded as 2 bytes. The byte ordering is little-endian.

Padding (variable): The padding (bytes with value zero) to align the field dataLen at a double word boundary.

dataLen (4 bytes): The length of the remaining structure, including this field, in bytes.

byteVal (variable): An 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;
```

5.187 systemFlags Values

The valid system flags used on directory objects are defined in [\[MS-ADTS\]](#) section 2.2.10.

5.188 UCHAR

A concrete type, as defined in [\[MS-DTYP\]](#) section 2.2.44. A UCHAR is an 8-bit, unsigned quantity.

5.189 ULARGE_INTEGER

ULARGE_INTEGER is a concrete type for a 64-bit, unsigned integer.

```
typedef struct {
    ULONGLONG QuadPart;
} ULARGE_INTEGER;
```

5.190 ULONG

A concrete type for a 32-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.50.

5.191 ULONGLONG

A concrete type for a 64-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.54.

5.192 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.

5.193 UnbindFromDSA()

```
procedure UnbindFromDSA(hDRS: DRS_HANDLE)
```

The UnbindFromDSA procedure closes the RPC connection that was established by the [BindToDSA](#) procedure.

5.194 UpdateRefs

```
procedure UpdateRefs(DRS_MSG_UPDREFS_V1 msgIn): ULONG
```

The UpdateRefs method implements the core functionality of [IDL DRSUpdateRefs](#), that is, adds or deletes a value from the repsTo of a specified NC replica.

```
err: DWORD
nc: DSName
rt: RepsTo
nc := msgIn.pNC^

/* If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously.
if DRS_ASYNC_OP in msgIn.ulOptions then
    Asynchronous Processing: Initiate a logical thread of control
    to process the remainder of this request asynchronously
    return ERROR_SUCCESS
endif

/* If DRS_DEL_REF is specified, the return value is that associated with the DRS_DEL_REF
if DRS_DEL_REF in msgIn.ulOptions 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 := ERROR_SUCCESS
    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 msgIn.ulOptions 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 := msgIn.ulOptions ∩ {DRS_WRIT_REP}
        rt.timeLastAttempt := 0
        rt.timeLastSuccess := current time
        rt.consecutiveFailures := 0
        rt.resultLastAttempt := 0
        nc!repsTo := nc!repsTo + {rt}
        err := ERROR_SUCCESS
    else
        err := ERROR_DS_DRA_REF_ALREADY_EXISTS
    endif
endif
return err
```

5.195 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*.

5.196 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*.

5.197 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 1.
- dwReserved1:** Unused. MUST be 0 and ignored.
- cNumCursors:** The number of items in the **rgCursors** array.
- dwReserved2:** Unused. MUST be 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.

5.198 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 2.
- dwReserved1:** Unused. MUST be 0 and ignored.
- cNumCursors:** The number of items in the **rgCursors** array.
- dwReserved2:** Unused. MUST be 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.

5.199 userAccountControl Bits

The userAccountControl bits are bit flags that describe various qualities of a security account. The bit flags are presented below in little-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	L O	X	X	A D	X	X	X	S T	W T	I D	X	N A	D A	X	X	X	X	X	X	X	X	X	X	X	X	X	P S	X	X

- X:** Unused. MUST be zero and ignored.
- AD (ADS_UF_ACCOUNTDISABLE, 0x00000002):** The account is disabled.
- LO (ADS_UF_LOCKOUT, 0x00000010):** The account is temporarily locked out.

DA (ADS_UF_TEMP_DUPLICATE_ACCOUNT, 0x00000100): This is an account for a user whose primary account is in another domain.

NA (ADS_UF_NORMAL_ACCOUNT, 0x00000200): The default account type that represents a typical user.

ID (ADS_UF_INTERDOMAIN_TRUST_ACCOUNT, 0x00000800): The account for a domain-to-domain trust.

WT (ADS_UF_WORKSTATION_ACCOUNT, 0x00001000): The computer account for a computer that is a member of this domain.

ST (ADS_UF_SERVER_TRUST_ACCOUNT, 0x00002000): The computer account for a DC.

PS (ADS_UF_PARTIAL_SECRETS_ACCOUNT, 0x04000000): The computer account for an RODC.

5.200 UserNameFromNT4AccountName

```
procedure UserNameFromNT4AccountName(  
    nt4AccountName: unicodestring): unicodestring
```

If nt4AccountName is a name in Microsoft Windows NT® 4.0 operating system account name format, that is, two components separated by a backslash (for example, "DOMAIN\username"), 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.

5.201 USHORT

A concrete type for a 16-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.57.

5.202 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;
```

5.203 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.

5.204 UUID

UUID is a type that is equivalent to the [GUID](#) type.

5.205 ValidateDRSDemotionInput

```
procedure ValidatedRSDemotionInput(hDrs: DRS_HANDLE, opnum: integer)
```

Informative summary of behavior: The ValidateDRSDemotionInput procedure performs certain checks based on the input and throws an exception, if needed.

The server MUST raise an ERROR_INVALID_PARAMETER exception when opnum = 25 and IsAdlds() == false.

The server SHOULD raise an ERROR_INVALID_PARAMETER exception when opnum = 26 and IsAdlds() == false. [<47>](#)

The server MUST raise an ERROR_INVALID_PARAMETER exception when opnum = 27 and IsAdlds() == false.

5.206 ValidateDRSInput

```
procedure ValidatedRSInput(hDrs: DRS_HANDLE, opnum: integer)
```

Informative summary of behavior: The ValidateDRSInput procedure performs certain checks based on the input and throws an exception, if needed.

```
if opnum = 0 then
    return
endif

if (hDrs = null) then
    raise ERROR_INVALID_HANDLE exception
endif

if (ClientUUID(hDrs) ≠ NTDSAPI_CLIENT_GUID) and
    (IsServerExtensionsChanged(ServerExtensions(hDrs)) and
    opnum ≠ 1
    then
        raise ERROR_DS_DRS_EXTENSIONS_CHANGED exception
    endif

if (ClientUUID(hDrs) ≠ NTDSAPI_CLIENT_GUID) and
    (ClientExtensions(hDrs).dwReplEpoch ≠ DSAObj()!msDS-ReplicationEpoch) and
    opnum ≠ 1
```

```

then
    raise ERROR_DS_DIFFERENT_REPL_EPOCHS exception
endif

if IsAdlds() and
    (opnum = 9 or /*IDL_DRSGetMemberships*/
     opnum = 10 or /*IDL_DRSInterDomainMove*/
     opnum = 11 or /*IDL_DRSGetNT4ChangeLog*/
     opnum = 13 or /*IDL_DRSWriteSPN*/
     opnum = 15 or /*IDL_DRSRemoveDsDomain*/
     opnum = 16 or /*IDL_DRSDomainControllerInfo*/
     opnum = 20 or /*IDL_DRSAddSidHistory*/
     opnum = 21 or /*IDL_DRSGetMemberships2*/
     opnum = 24 /*IDL_DRSQuerySitesByCost*/)
then
    raise ERROR_INVALID_PARAMETER exception
endif

if AmIRODC() and
    (opnum = 3 or /*IDL_DRSGetNCChanges*/
     opnum = 10 or /*IDL_DRSInterDomainMove*/
     opnum = 11 or /*IDL_DRSGetNT4ChangeLog*/
     opnum = 14 or /*IDL_DRSRemoveDsServer*/
     opnum = 15 or /*IDL_DRSRemoveDsDomain*/
     opnum = 17 or /*IDL_DRSAddEntry*/
     opnum = 20 /*IDL_DRSAddSidHistory*/)
then
    raise ERROR_INVALID_PARAMETER exception
endif

ValidateDRSDemotionInput(hDrs, opnum)

```

5.207 Value

Value is an abstract type for attribute values used for abstract value representation (see section [5.16.2](#)).

5.208 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.

5.209 ValueFromATTRVAL

```
procedure ValueFromATTRVAL(  
  a: ATTRVAL, s: Syntax, t: PrefixTable) : Value
```

The ValueFromATTRVAL procedure converts a value of syntax *s* expressed as a concrete [ATTRVAL](#) *a* into the abstract [Value](#) encoding, using the prefix table represented by *t*.

See section [5.16.3](#) for the specification of this procedure.

5.210 WCHAR

A concrete type, as specified in [\[MS-DTYP\]](#) section 2.2.59. A WCHAR is a 16-bit, unsigned integer in little-endian byte order that is used to store a double-byte Unicode character. A WCHAR * is a pointer to a null-terminated Unicode string.

6 Security

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
{

    typedef LONGLONG DSTIME;

    typedef [context_handle] void * DRS_HANDLE;

    typedef struct {
        char Data[28];
    } NT4SID;

    typedef struct {
        unsigned long structLen;
        unsigned long SidLen;
        GUID Guid;
        NT4SID Sid;
        unsigned long NameLen;
        [range(0, 1048576)] [size_is(NameLen + 1)] WCHAR StringName[];
    } DSNAME;

    typedef LONGLONG USN;

    typedef struct {
        USN usnHighObjUpdate;
        USN usnReserved;
        USN usnHighPropUpdate;
    } USN_VECTOR;

    typedef struct {
        UUID uuidDsa;
        USN usnHighPropUpdate;
    } UPTODATE_CURSOR_V1;

    typedef struct {
        DWORD dwVersion;
        DWORD dwReserved1;
        [range(0,1048576)] DWORD cNumCursors;
        DWORD dwReserved2;
        [size_is(cNumCursors)] UPTODATE_CURSOR_V1 rgCursors[];
    } UPTODATE_VECTOR_V1_EXT;

    typedef 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(0,26214400)] 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 fIsRdc;
    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_FFFFFFFFW;

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;
    } INTFORMPROB_DRS_WIRE_V1;

typedef struct _PROBLEMLIST_DRS_WIRE_V1 {
    struct _PROBLEMLIST_DRS_WIRE_V1 *pNextProblem;
    INTFORMPROB_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;
    RPC_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;
    [unique] 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;
    [unique] 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 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;
    ULONG ulMoreFlags;
} DRS_MSG_GETCHGREQ_V10;

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;
    [case(10)] DRS_MSG_GETCHGREQ_V10 V10;
} DRS_MSG_GETCHGREQ;

typedef struct {
    DRS_COMPRESSED_BLOB CompressedV1;
} DRS_MSG_GETCHGREPLY_V2;

typedef enum {
    DRS_COMP_ALG_NONE = 0,
    DRS_COMP_ALG_UNUSED = 1,
    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_RMSVREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_RMSVREQ_V1 V1;
} DRS_MSG_RMSVREQ;

typedef struct {
    BOOL fLastDcInDomain;
} DRS_MSG_RMSVREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_RMSVREPLY_V1 V1;
} DRS_MSG_RMSVREPLY;

typedef struct {
    [string] LPWSTR DomainDN;
} DRS_MSG_RMDMREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_RMDMREQ_V1 V1;
} DRS_MSG_RMDMREQ;

typedef struct {
    DWORD Reserved;
} DRS_MSG_RMDMREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_RMDMREPLY_V1 V1;
} DRS_MSG_RMDMREPLY;

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;

```

```

        DSTIME 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;
    [ref] 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;
    [string] 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_MOVE_REPLY *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_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);

// 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_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);

// 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_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);

// opnum 20
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);

// 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 pwErrorMessage;
} 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 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;

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_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);
}

```

8 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Active Directory® Application Mode (ADAM)
- Active Directory® Lightweight Directory Services (AD LDS) for Windows® Vista
- Active Directory® Lightweight Directory Services (AD LDS) for Windows® 7
- Microsoft Windows® 2000 operating system
- Microsoft Windows® 2000 Server operating system
- Windows® XP operating system
- Windows Server® 2003 operating system
- Windows Server® 2003 R2 operating system
- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product 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 using only 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 2.2.4.2:](#) SPN "ldap/<NetBIOS hostname>/<NetBIOS domain name>" is available in Windows 7 and Windows Server 2008 R2.

[<4> Section 4.1:](#) 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. All IDL methods and their associated concrete types continue to exist in this interface in subsequent versions of Windows according to the applicability list at the beginning of this section.

Data type or IDL method	Section	Windows version introduced
DRS_MSG_GETCHGREQ_V7 support	4.1.10.2.5	Windows Server 2003

Data type or IDL method	Section	Windows version introduced
DRS_MSG_DCINFOREPLY_V3	4.1.5.1.6	Windows Server 2008
DS_DOMAIN_CONTROLLER_INFO_3W	4.1.5.1.10	Windows Server 2008
DRS_MSG_GETCHGREQ_V8 support	4.1.10.2.6	Windows Server® 2003 operating system
DRS_MSG_GETCHGREQ_V10 support	4.1.10.2.7	Windows Server 2008 R2
DRS_MSG_GETCHGREPLY_V6 support	4.1.10.2.11	Windows Server® 2003 operating system
DRS_MSG_GETCHGREPLY_V7 support	4.1.10.2.12	Windows Server® 2003 operating system
DRS_MSG_ADDENTRYREQ_V3 support	4.1.1.1.4	Windows Server® 2003 operating system
DRS_MSG_ADDENTRYREPLY_V3 support	4.1.1.1.8	Windows Server® 2003 operating system
DRS_MSG_GETREPLINFO_REQ_V2	4.1.13.1.3	Windows Server® 2003 operating system
IDL_DRSGetObjectExistence	4.1.12	Windows Server® 2003 operating system
IDL_DRSReplicaVerifyObjects	4.1.24	Windows Server® 2003 operating system
IDL_DRSFinishDemotion	4.1.7	Windows Server 2008
IDL_DRSInitDemotion	4.1.14	Windows Server® 2008 operating system
IDL_DRSReplicaDemotion	4.1.21	Windows Server® 2008 operating system

<5> [Section 4.1](#): The following table identifies the methods for which the Windows client operating systems (Windows 2000, Windows XP, Windows Vista, and Windows 7) can implement a client role.

Method	Windows client operating systems can implement a client role
IDL_DRSBind Opnum: 0	Yes
IDL_DRSUnbind Opnum: 1	Yes
IDL_DRSReplicaSync Opnum: 2	Yes
IDL_DRSGetNCChanges Opnum: 3	No
IDL_DRSUpdateRefs Opnum: 4	Yes
IDL_DRSReplicaAdd Opnum: 5	Yes
IDL_DRSReplicaDel Opnum: 6	Yes
IDL_DRSReplicaModify	Yes

Method	Windows client operating systems can implement a client role
Opnum: 7	
IDL_DRSVerifyNames Opnum: 8	No
IDL_DRSGetMemberships Opnum: 9	No
IDL_DRSInterDomainMove Opnum: 10	No
IDL_DRSGetNT4ChangeLog Opnum: 11	No
IDL_DRSCrackNames Opnum: 12	Yes
IDL_DRSWriteSPN Opnum: 13	Yes
IDL_DRSRemoveDsServer Opnum: 14	Yes
IDL_DRSRemoveDsDomain Opnum: 15	Yes
IDL_DRSDomainControllerInfo Opnum: 16	Yes
IDL_DRSAddEntry Opnum: 17	No
IDL_DRSExecuteKCC Opnum: 18	Yes
IDL_DRSGetReplInfo Opnum: 19	Yes
IDL_DRSAddSidHistory Opnum: 20	Yes
IDL_DRSGetMemberships2 Opnum: 21	No
IDL_DRSReplicaVerifyObjects Opnum: 22	Yes
IDL_DRSGetObjectExistence Opnum: 23	No
IDL_DRSQuerySitesByCost Opnum: 24	Yes

Method	Windows client operating systems can implement a client role
IDL DRSInitDemotion Opnum: 25	Yes
IDL DRSReplicaDemotion Opnum: 26	Yes
IDL DRSFinishDemotion Opnum: 27	Yes

<6> [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.

<7> [Section 4.1.1.1.6:](#) Though this response version appears in the IDL, Windows DCs do not support it.

<8> [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 [3.1.4.9.4](#)). The **srcDomainController** variable in the [IDL DRSAddSidHistory](#) method is used as the *SystemName* parameter to LsarOpenPolicy2, 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 LsarQueryInformationPolicy 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](#) method, 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.

<9> [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.

<10> [Section 4.1.3.1](#): Windows non-DC client callers always pass NTDSAPI_CLIENT_GUID in puuidClientDsa. If a Windows DC client caller uses the returned DRS_HANDLE for subsequent calls to the [IDL DRSWriteSPN](#) method, then the client MUST pass NTDSAPI_CLIENT_GUID in puuidClientDsa. In any other cases, Windows DC client callers pass DC!serverGuid in puuidClientDsa.

<11> [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). Windows non-DC clients and Windows DC clients not running Windows Server 2008 or Windows Server 2008 R2 do not include the **ConfigObjGUID** and **dwFlagsExt** fields in the [DRS_EXTENSIONS_INT](#) structure.

<12> [Section 4.1.3.2](#): The **ConfigObjGUID** and **dwFlagsExt** fields in the [DRS_EXTENSIONS_INT](#) structure are included only by servers running Windows Server 2008 or Windows Server 2008 R2.

<13> [Section 4.1.5.2](#): All of the information levels listed in section [4.1.5.2](#) have existed in the [drsuapi RPC interface](#) since Windows 2000, except as noted in the following table.

Infolevel	Introduced in Windows version
3	Windows Server 2008

<14> [Section 4.1.6.3](#): The Windows implementation of this method puts the KCC execution requests in a local-machine work queue. If DS_KCC_FLAG_DAMPED is specified in the call to [IDL DRSExecuteKCC](#) and there is already a request pending, the execution request is not added to the queue in order to reduce redundant requests.

<15> [Section 4.1.7.3](#): The Windows implementation of the [IDL DRSFinishDemotion](#) method causes the underlying RPC protocol [\[MS-RPCE\]](#) to throw an RPC_S_INVALID_TAG exception when returning ERROR_INVALID_PARAMETER.

<16> [Section 4.1.10.1.1](#): The Windows implementation never declares msgIn.uuidInvocIdSrc and msgIn.usnvecFrom that are otherwise valid to be stale.

<17> [Section 4.1.10.1.1](#): The internal format of [USN_VECTOR](#) identifies the start of a cycle.

<18> [Section 4.1.10.1.2](#): The goal is advanced on each request of a cycle.

<19> [Section 4.1.10.1.2](#): c.usnHighPropUpdate is never set to 0.

<20> [Section 4.1.10.1.3](#): The Request Role extended operation is supported by Windows 2000 Server, Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2.

<21> [Section 4.1.10.1.3](#): The Abandon Role extended operation is supported by Windows 2000 Server, Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2.

<22> [Section 4.1.10.1.3](#): The Allocate RIDs extended operation is supported by Windows 2000 Server, Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2.

<23> [Section 4.1.10.1.3](#): The Replicate Single Object extended operation is supported by Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2.

<24> [Section 4.1.10.1.3](#): The Replicate Single Object including Secret Data extended operation is supported by Windows Server 2008 and Windows Server 2008 R2.

<25> [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 [\[MS-SRPL\]](#)).

<26> [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 ([\[MS-SRPL\]](#)).

<27> [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 ([\[MS-SRPL\]](#)).

<28> [Section 4.1.10.5.7](#): 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 Ancestors 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.

<29> [Section 4.1.10.6.5](#): 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.

<30> [Section 4.1.12.3](#): Windows uses count = 1000.

<31> [Section 4.1.14.2](#): The Windows implementation of the [IDL DRSInitDemotion](#) method causes the underlying RPC protocol (as specified in [\[MS-RPCE\]](#)) to throw an RPC_S_INVALID_TAG exception when returning ERROR_ACCESS_DENIED.

<32> [Section 4.1.15.1.2](#): Although this request version appears in the IDL, Windows DCs do not support it. It was never supported in a released version of Windows Server.

<33> [Section 4.1.15.1.5](#): Although this response version appears in the IDL, Windows DCs do not support it.

<34> [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.

<35> [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.

<36> [Section 5.38](#): In Windows 2000 Server, the **cb** field contains the count of bytes in the fields **dwFlags** through **Pid**, inclusive, which is the size of the structure in that version minus the 4 bytes of the **cb** field.

<37> [Section 5.38](#): In Windows Server 2003, the **cb** field contains the count of bytes in the fields **dwFlags** through **dwReplEpoch**, inclusive, which is the size of the structure in that version minus the 4 bytes of the **cb** field.

<38> [Section 5.38](#): Client callers set dwFlags to zero.

<39> [Section 5.38](#): This field contains the process ID of the client.

<40> [Section 5.38](#): The **dwReplEpoch** field in the [DRS_EXTENSIONS_INT](#) structure is included only by servers running Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, or Windows Server 2008 R2.

<41> [Section 5.38](#): The **ConfigObjGUID** and **dwFlagsExt** fields in the [DRS_EXTENSIONS_INT](#) structure are included only by servers running Windows Server 2008 or Windows Server 2008 R2.

<42> [Section 5.38](#): 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). Windows non-DC clients and Windows DC clients not running Windows Server 2008 or Windows Server 2008 R2 do not include the **ConfigObjGUID** and **dwFlagsExt** fields in the [DRS_EXTENSIONS_INT](#) structure.

<43> [Section 5.40](#): All the DRS_OPTIONS listed in section [5.40](#) have existed in the drsuapi RPC interface since Windows 2000 except those listed in the following table.

Flag	Windows version introduced
DRS_SYNC_PAS	Windows Server 2003

This flag is ignored inbound on the operating systems previous to its introduction.

All the DRS_OPTIONS listed in section [5.40](#) continue to exist in this interface in subsequent versions of Windows according to the applicability list at the beginning of this section, except those listed in the following table.

Flag	Windows version introduced
DRS_SYNC_ALL	Windows Server 2008 R2

This flag is ignored inbound on the operating systems in which it was removed and in subsequent versions.

The following pseudocode specifies how to unmask the flags that are unsupported for each operating system.

msgReq.ulFlags &= msgReq.ulFlags & allowedFlagsForSpecificOS

<44> [Section 5.49](#): No range is supported on any member of DSNAME in Windows 2000 Server. A range of 0 to 10485760 is supported on the **NameLen** member of DSNAME in Windows Server 2003 and Windows Server 2003 R2. A range of 0 to 10485761 is supported on the **StringName** member of DSNAME in Windows Server 2008 and Windows Server 2008 R2.

<45> [Section 5.164](#): Windows 2000 Server, Windows Server 2003, and Windows Server 2003 R2 AD DS DCs have a value of 1 in the dwVersion field. Windows Server 2008 and Windows Server 2008 R2 AD DS DCs have a value of 2 in the dwVersion field. Windows Server 2003 R2, Windows Server 2008 and Windows Server 2008 R2 AD LDS DCs have a value of 2 in the **dwVersion** field.

<46> [Section 5.165](#): Windows 2000 Server, Windows Server 2003, and Windows Server 2003 R2 AD DS DCs have a value of 1 in the dwVersion field. Windows Server 2008 and Windows Server 2008 R2 AD DS DCs have a value of 2 in the **dwVersion** field. Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2 AD LDS DCs have a value of 2 in the **dwVersion** field.

<47> [Section 5.205](#): Windows Server 2008 and Windows Server 2008 R2 do not raise ERROR_INVALID_PARAMETER exception when opnum==26 and IsAdIds() == false. Instead, the method [IDL DRSReplicaDemotion \(section 4.1.21\)](#) executes, and the effects vary depending on the NC specified in pmsgIn.V1.pNC.

If pmsgIn.V1.pNC contains the [DSNAME](#) of the default NC, then:

- The return code from [IDL DRSReplicaDemotion](#) is ERROR_SUCCESS.
- Only the FSMO roles contained within the domain NC, as described in [\[MS-ADTS\] section 3.1.1.1.11, FSMO Roles](#), are transferred to a replication partner.
- pmsgIn.V1.pNC!repsFrom values are not removed.

If pmsgIn.V1.pNC contains the [DSNAME](#) of the config NC, then:

- The return code from [IDL DRSReplicaDemotion](#) is ERROR_INVALID_DOMAINNAME.
- No FSMO roles are transferred.
- pmsgIn.V1.pNC!repsFrom values are not removed.

If pmsgIn.V1.pNC contains the [DSNAME](#) of the schema NC, then:

- The return code from [IDL DRSReplicaDemotion](#) is ERROR_INVALID_DOMAINNAME.
- No FSMO roles are transferred.
- pmsgIn.V1.pNC!repsFrom values are not removed.

If pmsgIn.V1.pNC contains the [DSNAME](#) of a domain NC and pmsgIn.V1.pNC!instanceType does not contain IT_WRITE, then:

- The return code from [IDL DRSReplicaDemotion](#) is ERROR_NO_SUCH_DOMAIN.
- No FSMO roles are transferred.
- pmsgIn.V1.pNC!repsFrom values are not removed.

If pmsgIn.V1.pNC contains the [DSNAME](#) of an application NC, then:

- The return code from [IDL_DRSReplicaDemotion](#) is ERROR_NO_SUCH_DOMAIN.
- No FSMO roles are transferred.
- pmsgIn.V1.pNC!repsFrom values are not removed.

9 Change Tracking

This section identifies changes that were made to the [MS-DRSR] protocol document between the May 2011 and June 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated**.

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.2 References	Added explanatory statement regarding the removal of the publishing year from Microsoft Open Specification document references.	N	Content updated.

10 Index

A

Abstract types ([section 3.3.3](#) 50, [section 3.4.3](#) 52)
[Abstract value representations](#) 450
[Abstract value representations - converting to concrete](#) 453
[AbstractPTFromConcretePT](#) 442
[AccessCheckAttr](#) 442
[AccessCheckCAR](#) 443
[AccessCheckObject](#) 443
[AccessCheckWriteToSpnAttribute](#) 443
[AD LDS specifics](#) 61
[ADDENTRY_REPLY_INFO structure](#) 68
[AmILHServer](#) 445
[AmIRODC](#) 445
[Applicability](#) 42
[Asynchronous processing](#) 58
[ATRERR_DRS_WIRE_V1 structure](#) 69
[ATTR structure](#) 446
[ATTRBLOCK structure](#) 446
[Attributes](#) 56
[AttributeStamp](#) 446
[AttributeSyntax](#) 448
[AttrStamp](#) 448
[ATTRTYP](#) 448
[AttrtypFromSchemaObj](#) 448
[ATTRTYP-to-OID conversion](#) 463
ATTRVAL
 [abstract value representations](#) 450
 [ATTRTYP-to-OID conversion](#) 463
 [concrete value representations](#) 449
 [converting between abstract and concrete value representations](#) 453
 [overview](#) 449
[ATTRVAL structure](#) 449
[ATTRVALBLOCK](#) 469
[ATTRVALBLOCK structure](#) 469
[ATTRVALFromValue](#) 469

B

[BOOL](#) 469
[BYTE](#) 470

C

[Capability negotiation](#) 42
[Change tracking](#) 603
[CHANGE_LOG_ENTRIES](#) 470
[CHANGELOG_ENTRY](#) 470
[CheckGroupMembership](#) 470
[Client initialization](#) 61
[ClientAuthorizationInfo](#) 470
[ClientExtensions](#) 470
Client-to-DC operations
 [security](#) 45
 [security provider](#) 46
 [SPN for target DC in AD DS](#) 46

[SPN for target DC in AD LDS](#) 47
 [Common configuration example](#) 59
 [COMPRESSED_DATA structure](#) 189
Concrete types ([section 3.3.3](#) 50, [section 3.4.2](#) 52)
[Concrete value representations](#) 449
[Concrete value representations - converting to abstract](#) 453
[ConcretePTFromAbstractPT](#) 471
[ConfigNC](#) 471
[Configuration example](#) 59
[CONTREF_DRS_WIRE_V1 structure](#) 71
[Cookie structure](#) 519

D

[Data display conventions](#) 60
[Data types](#) 442
[dc/DC](#) 471
[DC-to-DC operations](#) 44
[DefaultNC](#) 472
[DescendantObject](#) 473
[DN](#) 473
[DNBinary](#) 473
[DomainNameFromNT4AccountName](#) 473
[DRS_COMP_ALG_TYPE enumeration](#) 189
[DRS_COMPRESSED_BLOB structure](#) 190
[DRS_ERROR_DATA_V1 structure](#) 74
[DRS_EXTENSIONS](#) 474
[DRS_EXTENSIONS structure](#) 474
[DRS_EXTENSIONS_INT](#) 474
[DRS_EXTENSIONS_INT packet](#) 474
[DRS_HANDLE](#) 478
[DRS_MSG_ADDENTRYREPLY_V1 structure](#) 66
[DRS_MSG_ADDENTRYREPLY_V2 structure](#) 67
[DRS_MSG_ADDENTRYREPLY_V3 structure](#) 67
[DRS_MSG_ADDENTRYREQ_V1 structure](#) 65
[DRS_MSG_ADDENTRYREQ_V2 structure](#) 65
[DRS_MSG_ADDENTRYREQ_V3 structure](#) 66
[DRS_MSG_ADDSIDREPLY_V1 structure](#) 89
[DRS_MSG_ADDSIDREQ_V1 structure](#) 88
[DRS_MSG_CRACKREPLY_V1 structure](#) 123
[DRS_MSG_CRACKREQ_V1 structure](#) 119
[DRS_MSG_DCINFOREPLY_V1 structure](#) 146
[DRS_MSG_DCINFOREPLY_V2 structure](#) 146
[DRS_MSG_DCINFOREPLY_V3 structure](#) 146
[DRS_MSG_DCINFOREPLY_VFFFFFFFF structure](#) 147
[DRS_MSG_DCINFOREQ_V1 structure](#) 145
[DRS_MSG_EXISTREPLY_V1 structure](#) 290
[DRS_MSG_EXISTREQ_V1 structure](#) 289
[DRS_MSG_FINISH_DEMOTIONREPLY_V1 structure](#) 164
[DRS_MSG_FINISH_DEMOTIONREQ_V1 structure](#) 163
[DRS_MSG_GETCHGREPLY_V1 structure](#) 186
[DRS_MSG_GETCHGREPLY_V2 structure](#) 187
[DRS_MSG_GETCHGREPLY_V6 structure](#) 187
[DRS_MSG_GETCHGREPLY_V7 structure](#) 188
[DRS_MSG_GETCHGREQ_V10 structure](#) 185

[DRS MSG GETCHGREQ V3 structure](#) 182
[DRS MSG GETCHGREQ V4 structure](#) 182
[DRS MSG GETCHGREQ V5 structure](#) 183
[DRS MSG GETCHGREQ V7 structure](#) 183
[DRS MSG GETCHGREQ V8 structure](#) 184
[DRS MSG GETMEMBERSHIPS2 REPLY V1 structure](#) 177
[DRS MSG GETMEMBERSHIPS2 REQ V1 structure](#) 177
[DRS MSG GETREPLINFO REQ V1 structure](#) 294
[DRS MSG GETREPLINFO REQ V2 structure](#) 295
[DRS MSG INIT DEMOTIONREPLY V1 structure](#) 345
[DRS MSG INIT DEMOTIONREQ V1 structure](#) 345
[DRS MSG KCC EXECUTE V1 structure](#) 161
[DRS MSG MOVEREPLY V1 structure](#) 348
[DRS MSG MOVEREPLY V2 structure](#) 349
[DRS MSG MOVEREQ V1 structure](#) 347
[DRS MSG MOVEREQ V2 structure](#) 348
[DRS MSG NT4 CHGLOG REPLY V1 structure](#) 282
[DRS MSG NT4 CHGLOG REQ V1 structure](#) 281
[DRS MSG QUERYSITESREPLY V1 structure](#) 360
[DRS MSG QUERYSITESREPLYELEMENT V1 structure](#) 360
[DRS MSG QUERYSITESREQ V1 structure](#) 359
[DRS MSG REPADD V1 structure](#) 388
[DRS MSG REPADD V2 structure](#) 388
[DRS MSG REPDEL V1 structure](#) 393
[DRS MSG REPLICA DEMOTIONREPLY V1 structure](#) 398
[DRS MSG REPLICA DEMOTIONREQ V1 structure](#) 397
[DRS MSG REPMOD V1 structure](#) 402
[DRS MSG REPSYNC V1 structure](#) 405
[DRS MSG REPVERIFYOBJ V1 structure](#) 408
[DRS MSG REVMEMB REPLY V1 structure](#) 170
[DRS MSG REVMEMB REQ V1 structure](#) 169
[DRS MSG RMDMNREPLY V1 structure](#) 381
[DRS MSG RMDMNREQ V1 structure](#) 380
[DRS MSG RMSVRREPLY V1 structure](#) 384
[DRS MSG RMSVRREQ V1 structure](#) 384
[DRS MSG SPNREPLY V1 structure](#) 430
[DRS MSG SPNREQ V1 structure](#) 429
[DRS MSG UPDREFS V1 structure](#) 416
[DRS MSG VERIFYREPLY V1 structure](#) 423
[DRS MSG VERIFYREQ V1 structure](#) 422
[DRS OPTIONS](#) 478
[DRS SecBuffer](#) 481
[DRS SecBuffer structure](#) 481
[DRS SecBufferDesc](#) 482
[DRS SecBufferDesc structure](#) 482
[DRS SPN CLASS](#) 483
[DS DOMAIN CONTROLLER INFO 1W structure](#) 147
[DS DOMAIN CONTROLLER INFO 2W structure](#) 147
[DS DOMAIN CONTROLLER INFO 3W structure](#) 148
[DS DOMAIN CONTROLLER INFO FFFFFFFW structure](#) 149
[DS NAME FORMAT enumeration](#) 121
[DS NAME RESULT ITEMW structure](#) 122
[DS NAME RESULTW structure](#) 123
[DS REPL ATTR META DATA structure](#) 302
[DS REPL ATTR META DATA 2 structure](#) 303

[DS REPL ATTR VALUE META DATA structure](#) 305
[DS REPL ATTR VALUE META DATA 2 structure](#) 306
[DS REPL CLIENT CONTEXT structure](#) 308
[DS REPL CLIENT CONTEXTS structure](#) 308
[DS REPL CURSOR structure](#) 299
[DS REPL CURSOR 2 structure](#) 300
[DS REPL CURSOR 3W structure](#) 301
[DS REPL CURSORS structure](#) 299
[DS REPL CURSORS 2 structure](#) 300
[DS REPL CURSORS 3W structure](#) 300
[DS REPL KCC DSA FAILURESW structure](#) 303
[DS REPL KCC DSA FAILUREW structure](#) 304
[DS REPL NEIGHBORSW structure](#) 298
[DS REPL NEIGHBORW structure](#) 298
[DS REPL OBJ META DATA structure](#) 301
[DS REPL OBJ META DATA 2 structure](#) 302
[DS REPL OP TYPE](#) 483
[DS REPL OP TYPE enumeration](#) 483
[DS REPL OPW structure](#) 304
[DS REPL PENDING OPSW structure](#) 304
[DS REPL SERVER OUTGOING CALL structure](#) 309
[DS REPL SERVER OUTGOING CALLS structure](#) 309
[DS REPL VALUE META DATA structure](#) 305
[DS REPL VALUE META DATA 2 structure](#) 307
[DSA ADDRESS LIST DRS WIRE V1 structure](#) 71
[DSA MSG EXECUTE SCRIPT REPLY V1 structure](#) 440

[DSA MSG EXECUTE SCRIPT REQ V1 structure](#) 439
[DSA MSG PREPARE SCRIPT REPLY V1 structure](#) 435
[DSA MSG PREPARE SCRIPT REQ V1 structure](#) 434
[DSA RPC INST](#) 483
[DSA RPC INST structure](#) 483
[DSAOBJ](#) 483
[DSName \(\[section 5.48\]\(#\) 484, \[section 5.49\]\(#\) 484\)](#)
[DSNAME equality](#) 486
[DSNAME structure](#) 484
[DSTIME](#) 486
[DWORD](#) 487

E

[ENCRYPTED PAYLOAD packet](#) 190
[ENTINF](#) 487
[ENTINF structure](#) 487
[ENTINF EnumerateAttributes](#) 488
[ENTINF GetValue](#) 488
[ENTINF SetValue](#) 488
[ENTINFLIST](#) 488
[ENTINFLIST structure](#) 488
 Examples
 [common configuration example](#) 59
 [data display conventions](#) 60
 [IDL DRSBind method example](#) 115
[Expunge](#) 489

F

[Fields - vendor-extensible](#) 42
[FILETIME](#) 489
[FilteredGCPAS](#) 489
[FilteredPAS](#) 490
[FindChar](#) 490
[FindCharRev](#) 491
[FOREST_TRUST_INFORMATION](#) 491
[FOREST_TRUST_INFORMATION packet](#) 491
[FOREST_TRUST_RECORD_TYPE](#) 502
[FOREST_TRUST_RECORD_TYPE enumeration](#) 502
[ForestRootDomainNC](#) 502
[Full IDL](#) 566
[FullReplicaExists](#) 502

G

[GCPAS](#) 503
[GetAttrVals](#) 505
[GetCallerAuthzInfo](#) 505
[GetDefaultObjectCategory](#) 505
[GetDSNameFromDN](#) 506
[GetDSNameOfEnterpriseRODCsGroup](#) 172
[GetFilteredAttributeSet](#) 503
[GetForestFunctionalLevel](#) 506
[GetFSMORoleOwner](#) 506
[GetInstanceNameFromSPN](#) 507
[GetNCType](#) 504
[GetObjectNC](#) 507
[GetProxyEpoch](#) 507
[GetProxyType](#) 507
[GetServiceClassFromSPN](#) 507
[GetServiceNameFromSPN](#) 507
[Glossary](#) 23
[groupType bit flags](#) 508
[GUID](#) 508
[GuidFromString](#) 508
[GuidToString](#) 509

H

[handle_t](#) 509

I

[IDL](#) 566
[IDL_DRSAddEntry_method](#) 64
[IDL_DRSAddSidHistory_method](#) 87
[IDL_DRSBind_method](#) 105
[IDL_DRSBind_method example](#) 115
[IDL_DRSCrackNames_method](#) 118
[IDL_DRSDomainControllerInfo_method](#) 144
[IDL_DRSExecuteKCC_method](#) 160
[IDL_DRSFinishDemotion_method](#) 162
[IDL_DRSGetMemberships_method](#) 168
[IDL_DRSGetMemberships2_method](#) 176
[IDL_DRSGetNCChanges_method](#) 178
[IDL_DRSGetNT4ChangeLog_method](#) 280
[IDL_DRSGetObjectExistence_method](#) 288
[IDL_DRSGetReplInfo_method](#) 293
[IDL_DRSInitDemotion_method](#) 344
[IDL_DRSInterDomainMove_method](#) 346
[IDL_DRSQuerySitesByCost_method](#) 359

[IDL_DRSRemoveDsDomain_method](#) 379
[IDL_DRSRemoveDsServer_method](#) 383
[IDL_DRSReplicaAdd_method](#) 387
[IDL_DRSReplicaDel_method](#) 392
[IDL_DRSReplicaDemotion_method](#) 396
[IDL_DRSReplicaModify_method](#) 401
[IDL_DRSReplicaSync_method](#) 404
[IDL_DRSReplicaVerifyObjects_method](#) 407
[IDL_DRSUnbind_method](#) 415
[IDL_DRSUpdateRefs_method](#) 415
[IDL_DRSVerifyNames_method](#) 421
[IDL_DRSWriteSPN_method](#) 428
[IDL_DSASExecuteScript_method](#) 438
[IDL_DSAPrepareScript_method](#) 434
[Implementer - security considerations](#) 565
[Index of security parameters](#) 565
[Informative references](#) 37
[Initialization](#) 61
[instanceType bit flags](#) 509
[INT32 packet](#) 449
[INT64 packet](#) 449
[INTFORMPROB_DRS_WIRE_V1 structure](#) 69
[Introduction](#) 22
[Is2PartSPN](#) 509
[Is3PartSPN](#) 510
[IsBuiltinPrincipal](#) 510
[IsDCAccount](#) 511
[IsDomainNameInTrustedForest](#) 510
[IsGC](#) 511
[IsGetNCChangesPermissionGranted](#) 511
[IsGUIDBasedDNSName](#) 512
[IsMemberOfBuiltinAdminGroup](#) 512
[IsRecycleBinEnabled](#) 512
[IsRevealFilteredAttribute](#) 513
[IsRevealSecretRequest](#) 513
[IsValidServiceName](#) 515

K

[KCCFailedConnections](#) 515
[KCCFailedLinks](#) 515

L

Language constructs ([section 3.4.2](#) 52, [section 3.4.3](#) 52, [section 3.4.4](#) 54)
[LARGE_INTEGER](#) 515
[LDAP_CONN_PROPERTIES](#) 515
[LDAP_SERVER_DIRSYNC_OID](#) LDAP search control
[abstract types](#) 516
[concrete types](#) 519
[DirSyncReqToGetChgReq](#) 520
[GetChgReplyToSearchResult](#) 521
[GetResponseDirSyncControlValue](#) 525
[GetUsnUtdVectorFromCookie](#) 527
[IsFilteredAttributePresent](#) 528
[LDAPDisplayNameFromAttrTyp](#) 525
[overview](#) 516
[ProcessDirSyncSearchRequest](#) 519
[SecurityCheckForChanges](#) 528
[TransformDSNameToLdapDN](#) 525
[TransformEntinfToSearchEntry](#) 523

[TransformRepValInfListToSearchEntry](#) 524
[LDAPConnections](#) 529
[LinkStamp](#) 529
[LinkValueStamp](#) 530
[LocalAttidFromRemoteAttid](#) 530
[LONG](#) 531
[LONGLONG](#) 531
[LPWSTR](#) 531

M

[MakeAttid](#) 531
[MakeProxyValue](#) 531
[MasterReplicaExists](#) 531
[MD5_CTX](#) 531
[MD5Final](#) 532
[MD5Init](#) 532
[MD5Update](#) 532
[MergeUTD](#) 532
Messages
 [overview](#) 43
 [transport](#) 43
Methods ([section 1.3.1](#) 37, [section 4](#) 62)
[MTX_ADDR](#) 532
[MTX_ADDR structure](#) 532

N

[NAMERESOP_DRS_WIRE_V1 structure](#) 71
[NAMERR_DRS_WIRE_V1 structure](#) 70
[Naming conventions](#) 51
[NCType bits](#) 533
[NetworkAddress](#) 533
[NewPrefixTable](#) 534
[Normative references](#) 35
[NT4_REPLICATION_STATE structure](#) 283
[Nt4ReplicationState](#) 534
[NT4SID](#) 534
[NT4SID structure](#) 534
[NTDSTRANSPORT_OPT values](#) 535
[NULLGUID](#) 535

O

[Object attributes](#) 56
[Object\(Access-Point\)](#) 452
[Object\(DN-Binary\)](#) 452
[Object\(DN-String\)](#) 452
[Object\(DS-DN\)](#) 451
[Object\(OR-Name\)](#) 453
[ObjExists](#) 535
[OID](#) 535
[OID_t](#) 535
[OID_t structure](#) 535
[OidFromAttid](#) 535
[Organization](#) 49
[Overview](#) 49
[Overview \(synopsis\)](#) 37

P

[Parameters - security index](#) 565

[parent](#) 536
[PARTIAL_ATTR_VECTOR_V1_EXT](#) 536
[PARTIAL_ATTR_VECTOR_V1_EXT structure](#) 536
[partialAttributeSet](#) 536
[PartialGCReplicaExists](#) 536
[PAS_DATA](#) 536
[PAS_DATA packet](#) 536
[PdcChangeLog](#) 537
[PDS_NAME_RESULT_ITEMW](#) 122
[PDS_NAME_RESULTW](#) 123
[PDSA_RPC_INST](#) 483
[PerformAddOperation](#) 537
[PerformAddOperationAsSystem](#) 538
[PerformModifyOperation](#) 246
[Preconditions](#) 42
[PrefixTable](#) 538
[PrefixTableEntry](#) 538
[PrefixTableEntry structure](#) 538
Prerequisites ([section 1.5](#) 42, [section 3.3.1](#) 50)
[PROBLEMLIST_DRS_WIRE_V1 structure](#) 69
[Procedures](#) 442
[Processing - asynchronous](#) 58
[Product behavior](#) 594
[PROPERTY_META_DATA structure](#) 191
[PROPERTY_META_DATA_EXT](#) 538
[PROPERTY_META_DATA_EXT structure](#) 538
[PROPERTY_META_DATA_EXT_VECTOR](#) 539
[PROPERTY_META_DATA_EXT_VECTOR structure](#) 539
[proxiedObjectName value format](#) 539
[Pseudocode](#) 51

R

[RDN](#) 539
[rdnType](#) 539
[Record packet](#) 492
[RecycleObj](#) 540
References
 [informative](#) 37
 [normative](#) 35
[REFERR_DRS_WIRE_V1 structure](#) 70
[Relationship to other protocols](#) 41
[RemoveObj](#) 540
[REPLENTINFLIST](#) 540
[REPLENTINFLIST structure](#) 540
[ReplicationQueue](#) 541
[REPLTIMES](#) 541
[REPLTIMES structure](#) 541
[replUpToDateVector/ReplUpToDateVector](#) 542
[REPLVALINF](#) 542
[REPLVALINF structure](#) 542
[REPS_FROM](#) 543
[REPS_FROM packet](#) 543
[REPS_TO](#) 546
[REPS_TO packet](#) 546
[repsFrom/RepsFrom](#) 549
[repsTo/RepsTo](#) 550
[RevealSecretsPolicy enumeration](#) 228
[REVERSE_MEMBERSHIP_OPERATION_TYPE enumeration](#) 169
[Rid](#) 551

[Right](#) 551
[RIGHT values](#) 551
[RPCClientContexts](#) 551
[RPCOutgoingContexts](#) 551

S

[sAMAccountType values](#) 552
[SCHEMA_PREFIX_TABLE](#) 552
[SCHEMA_PREFIX_TABLE structure](#) 552
[SchemaNC](#) 553
[SchemaObj](#) 553
[SECERR_DRS_WIRE_V1 structure](#) 72
Security
 [background](#) 43
 [client-to-DC operations](#) 45
 [DC-to-DC operations](#) 44
 [implementer considerations](#) 565
 [overview](#) 43
 [parameter index](#) 565
 provider ([section 2.2.3.1](#) 44, [section 2.2.4.1](#) 46)
 [service authentication](#) 43
 [SPN for target DC in AD DS](#) 46
 [SPN for target DC in AD LDS](#) 47
 [Security provider](#) 44
 [Sequencing issues](#) 38
 [Server extensions](#) 553
 [Server initialization](#) 61
 [Service authentication](#) 43
 [SID](#) 553
 [SidFromStringSid](#) 553
 [SPN for target DC in AD DS](#) 46
 [SPN for target DC in AD LDS](#) 47
 [SPN target DC in AD DS](#) 44
 [SPN target DC in AD LDS](#) 45
 [StampLessThanOrEqualUTD](#) 553
 [Standards assignments](#) 42
 [StartsWith](#) 554
 [State model](#) 50
 [String\(NT-Sec-Desc\)](#) 453
 [String\(Sid\)](#) 453
 [String\(Teletex\)](#) 453
 [StringSidFromSid](#) 554
 [SubString](#) 554
 [SVCERR_DRS_WIRE_V1 structure](#) 73
 [Syntax](#) 554
 [SYNTAX_ADDRESS packet](#) 554
 [SYNTAX_DISTNAME_BINARY packet](#) 555
 [SYSERR_DRS_WIRE_V1 structure](#) 73
 [systemFlags values](#) 557

T

[Tracking changes](#) 603
[Transactions](#) 50
[Transport](#) 43
Types ([section 1.3.3](#) 40, [section 3.3.3](#) 50)
[Typographical conventions](#) 49

U

[UCHAR](#) 557

[ULARGE_INTEGER structure](#) 557
[ULONG](#) 557
[ULONGLONG](#) 557
[UPDERR_DRS_WIRE_V1 structure](#) 73
[UPTODATE_CURSOR_V1 structure](#) 559
[UPTODATE_CURSOR_V2 structure](#) 559
[UPTODATE_VECTOR_V1_EXT structure](#) 559
[UPTODATE_VECTOR_V2_EXT structure](#) 560
[userAccountControl bits](#) 560
[UserNameFromNT4AccountName](#) 561
[USHORT](#) 561
[USN](#) 561
[USN_VECTOR structure](#) 561
[UUID](#) 562

V

[Value](#) 563
[VALUE_META_DATA_EXT_V1 structure](#) 563
[ValueFromATTRVAL](#) 564
Values ([section 5.16.1](#) 449, [section 5.16.2](#) 450, [section 5.16.3](#) 453)
[Variables](#) 442
[Vendor-extensible fields](#) 42
[Versioning](#) 42

W

[WCHAR](#) 564