

[MS-DRDM]: Directory Replication and Data Management (DRDM) 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
04/03/2007	0.01		MCPD Milestone Longhorn Initial Availability
07/03/2007	1.0	Major	MLonghorn+90
07/20/2007	1.0.1	Editorial	Revised and edited the technical content.
08/10/2007	1.0.2	Editorial	Revised and edited the technical content.
09/28/2007	1.1	Minor	Updated the technical content.
10/23/2007	1.2	Minor	Updated the technical content.
11/30/2007	1.2.1	Editorial	Revised and edited the technical content.
01/25/2008	2.0	Major	Added two sections.
03/14/2008	2.1	Minor	Updated the technical content.
05/16/2008	2.1.1	Editorial	Revised and edited 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	25.0	Major	Significantly changed the technical content.
03/25/2011	26.0	Major	Significantly changed the technical content.
05/06/2011	26.0	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	26.1	Minor	Clarified the meaning of the technical content.

Contents

1	Introduction	15
1.1	Glossary	16
1.2	References.....	27
1.2.1	Normative References.....	27
1.2.2	Informative References	28
1.3	Overview	28
1.3.1	Methods Categorized by Function	28
1.3.2	Sequencing Issues.....	29
1.3.3	Most Frequently Used Types	30
1.4	Relationship to Other Protocols.....	31
1.5	Prerequisites/Preconditions	31
1.6	Applicability Statement.....	31
1.7	Versioning and Capability Negotiation.....	31
1.8	Vendor-Extensible Fields.....	32
1.9	Standards Assignments	32
2	Message Transport	33
2.1	RPC Transport	33
2.2	Protocol Security	33
2.2.1	General Background	33
2.2.2	Service Principal Names for Domain Controllers	33
2.2.3	DC-to-DC Operations	33
2.2.4	Client-to-DC Operations	34
2.2.4.1	Security Provider	34
2.2.4.2	SPN for a Target DC in AD DS	34
2.2.4.3	SPN for a Target DC in AD LDS.....	35
2.3	Directory Service Schema Elements	36
3	Background to Behavior Specifications	37
3.1	Document Organization	37
3.2	Typographical Conventions	37
3.3	State Model	38
3.3.1	Preliminaries.....	38
3.3.2	Transactions	38
3.3.3	Concrete and Abstract Types	38
3.4	Pseudocode Language	39
3.4.1	Naming Conventions.....	39
3.4.2	Language Constructs for Concrete Types.....	40
3.4.3	Language Constructs for Abstract Types.....	40
3.4.4	Common Language Constructs.....	42
3.4.5	Access to Objects and Their Attributes	43
3.4.6	Asynchronous Processing	46
3.5	Conventions for Protocol Examples.....	47
3.5.1	Common Configuration	47
3.5.2	Data Display Conventions.....	48
3.6	Server and Client Initialization.....	49
3.6.1	AD LDS Specifics	49
4	RPC Methods and Their Behavior.....	50
4.1	drsuapi RPC Interface.....	50

4.1.1	IDL_DRSAddSidHistory (Opnum 20)	52
4.1.1.1	Method-Specific Concrete Types	53
4.1.1.1.1	DRS_MSG_ADDSIDREQ	53
4.1.1.1.2	DRS_MSG_ADDSIDREQ_V1	53
4.1.1.1.3	DRS_MSG_ADDSIDREPLY	54
4.1.1.1.4	DRS_MSG_ADDSIDREPLY_V1	54
4.1.1.1.5	DRS_ADDSID_FLAGS	55
4.1.1.2	Method-Specific Abstract Types and Procedures	55
4.1.1.2.1	ConnectionCtx	55
4.1.1.2.2	ConnectToDC	55
4.1.1.2.3	ConnectToDCWithCreds	56
4.1.1.2.4	GenerateFailureAudit	56
4.1.1.2.5	GenerateSuccessAudit	56
4.1.1.2.6	GenerateSuccessAuditRemotely	56
4.1.1.2.7	GetKeyLength	56
4.1.1.2.8	GetPDC	57
4.1.1.2.9	HasAdminRights	57
4.1.1.2.10	IsAuditingEnabled	57
4.1.1.2.11	IsLocalRpcCall	57
4.1.1.2.12	IsNT4SP4OrBetter	57
4.1.1.2.13	IsAuditingGroupPresent	57
4.1.1.2.14	IsWellKnownDomainRelativeSid	58
4.1.1.2.15	LastRID	58
4.1.1.2.16	RemoteQuery	58
4.1.1.3	Server Behavior of the IDL_DRSAddSidHistory Method	58
4.1.1.4	Examples of the IDL_DRSAddSidHistory Method	66
4.1.1.4.1	Calling IDL_DRSAddSidHistory with DS_ADDSID_FLAG_PRIVATE_CHK_SECURE Flags	66
4.1.1.4.1.1	Client Request	66
4.1.1.4.1.2	Server Response	66
4.1.1.4.1.3	Final State	66
4.1.1.4.2	Calling IDL_DRSAddSidHistory with DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ Flags	66
4.1.1.4.2.1	Initial State	66
4.1.1.4.2.2	Client Request	67
4.1.1.4.2.3	Server Response	67
4.1.1.4.2.4	Final State	68
4.1.1.4.3	Calling IDL_DRSAddSidHistory with 0 in Flags	68
4.1.1.4.3.1	Initial State	68
4.1.1.4.3.2	Client Request	69
4.1.1.4.3.3	Server Response	70
4.1.1.4.3.4	Final State	70
4.1.2	IDL_DRSSBind (Opnum 0)	71
4.1.2.1	Client Behavior When Sending the IDL_DRSSBind Request	71
4.1.2.2	Server Behavior of the IDL_DRSSBind Method	74
4.1.2.3	Client Behavior When Receiving the IDL_DRSSBind Response	77
4.1.2.4	Examples of the IDL_DRSSBind Method	78
4.1.2.4.1	Initial State	78
4.1.2.4.2	Client Request	78
4.1.2.4.3	Server Response	79
4.1.2.4.4	Final State	80
4.1.3	IDL_DRSCrackNames (Opnum 12)	80
4.1.3.1	Method-Specific Concrete Types	81

4.1.3.1.1	DRS_MSG_CRACKREQ	81
4.1.3.1.2	DRS_MSG_CRACKREQ_V1	81
4.1.3.1.3	DS_NAME_FORMAT	83
4.1.3.1.4	DS_NAME_RESULT_ITEMW	84
4.1.3.1.5	DS_NAME_RESULTW	84
4.1.3.1.6	DRS_MSG_CRACKREPLY	84
4.1.3.1.7	DRS_MSG_CRACKREPLY_V1	85
4.1.3.1.8	DS_NAME_ERROR	85
4.1.3.2	Method-Specific Abstract Types and Procedures	87
4.1.3.2.1	CanonicalNameFromCanonicalNameEx	87
4.1.3.2.2	DomainDNSNameFromDomain	87
4.1.3.2.3	DomainFromDomainDNSName	87
4.1.3.2.4	DomainNameFromCanonicalName	87
4.1.3.2.5	DomainNameFromSid	87
4.1.3.2.6	DomainNameFromUPN	88
4.1.3.2.7	DomainNetBIOSNameFromDomain	88
4.1.3.2.8	DomainSidFromSid	88
4.1.3.2.9	CrackNames	88
4.1.3.2.10	LookupName	92
4.1.3.2.11	LookupAttr	95
4.1.3.2.12	LookupCanonicalName	96
4.1.3.2.13	GetCanonicalName	96
4.1.3.2.14	LookupSPN	97
4.1.3.2.15	LookupSID	97
4.1.3.2.16	LookupUnknownName	98
4.1.3.2.17	LookupUPNAndAltSecID	98
4.1.3.2.18	LookupFPO	99
4.1.3.2.19	MapSPN	100
4.1.3.2.20	ParseCanonicalName	100
4.1.3.2.21	RetrieveDCSuffixFromDn	101
4.1.3.2.22	UserNameFromUPN	101
4.1.3.2.23	TranslateFPOToName	101
4.1.3.2.24	ConstructOutput	102
4.1.3.3	Server Behavior of the IDL_DRSCrackNames Method	103
4.1.3.4	Examples of the IDL_DRSCrackNames Method	103
4.1.3.4.1	Initial State	104
4.1.3.4.2	Client Request	104
4.1.3.4.3	Server Response	105
4.1.3.4.4	Final State	105
4.1.4	IDL_DRSDomainControllerInfo (Opnum 16)	105
4.1.4.1	Method-Specific Concrete Types	106
4.1.4.1.1	DRS_MSG_DCINFOREQ	106
4.1.4.1.2	DRS_MSG_DCINFOREQ_V1	106
4.1.4.1.3	DRS_MSG_DCINFOREPLY	106
4.1.4.1.4	DRS_MSG_DCINFOREPLY_V1	107
4.1.4.1.5	DRS_MSG_DCINFOREPLY_V2	107
4.1.4.1.6	DRS_MSG_DCINFOREPLY_V3	107
4.1.4.1.7	DRS_MSG_DCINFOREPLY_VFFFFFFF	108
4.1.4.1.8	DS_DOMAIN_CONTROLLER_INFO_1W	108
4.1.4.1.9	DS_DOMAIN_CONTROLLER_INFO_2W	109
4.1.4.1.10	DS_DOMAIN_CONTROLLER_INFO_3W	110
4.1.4.1.11	DS_DOMAIN_CONTROLLER_INFO_FFFFFFFW	111
4.1.4.2	Server Behavior of the IDL_DRSDomainControllerInfo Method	111

4.1.4.3	Examples of the IDL_DRSDomainControllerInfo Method	116
4.1.4.3.1	Initial State	116
4.1.4.3.2	Client Request	120
4.1.4.3.3	Server Response	120
4.1.4.3.4	Final State	121
4.1.5	IDL_DRSExecuteKCC (Opnum 18)	121
4.1.5.1	Method-Specific Concrete Types	121
4.1.5.1.1	DRS_MSG_KCC_EXECUTE	121
4.1.5.1.2	DRS_MSG_KCC_EXECUTE_V1	122
4.1.5.2	Method-Specific Abstract Types and Procedures	122
4.1.5.2.1	ExecuteKCCTasks	122
4.1.5.3	Server Behavior of the IDL_DRSExecuteKCC Method	123
4.1.6	IDL_DRSFinishDemotion (Opnum 27)	123
4.1.6.1	Method-Specific Concrete Types	124
4.1.6.1.1	DRS_MSG_FINISH_DEMOTIONREQ	124
4.1.6.1.2	DRS_MSG_FINISH_DEMOTIONREQ_V1	124
4.1.6.1.3	DRS_MSG_FINISH_DEMOTIONREPLY	125
4.1.6.1.4	DRS_MSG_FINISH_DEMOTIONREPLY_V1	126
4.1.6.2	Method-Specific Abstract Types and Procedures	126
4.1.6.2.1	RemoveADLDSServer	126
4.1.6.2.2	RemoveADLDSSCP	126
4.1.6.2.3	RemoveADLDSSPNs	127
4.1.6.3	Server Behavior of the IDL_DRSFinishDemotion Method	127
4.1.7	IDL_DRSGetReplInfo (Opnum 19)	129
4.1.7.1	Method-Specific Concrete Types	130
4.1.7.1.1	DRS_MSG_GETREPLINFO_REQ	130
4.1.7.1.2	DRS_MSG_GETREPLINFO_REQ_V1	130
4.1.7.1.3	DRS_MSG_GETREPLINFO_REQ_V2	131
4.1.7.1.4	DS_REPL_INFO Codes	131
4.1.7.1.5	DRS_MSG_GETREPLINFO_REPLY	132
4.1.7.1.6	DS_REPL_NEIGHBORSW	134
4.1.7.1.7	DS_REPL_NEIGHBORW	134
4.1.7.1.8	DS_REPL_CURSORS	135
4.1.7.1.9	DS_REPL_CURSOR	136
4.1.7.1.10	DS_REPL_CURSORS_2	136
4.1.7.1.11	DS_REPL_CURSOR_2	136
4.1.7.1.12	DS_REPL_CURSORS_3W	137
4.1.7.1.13	DS_REPL_CURSOR_3W	137
4.1.7.1.14	DS_REPL_OBJ_META_DATA	138
4.1.7.1.15	DS_REPL_ATTR_META_DATA	138
4.1.7.1.16	DS_REPL_OBJ_META_DATA_2	139
4.1.7.1.17	DS_REPL_ATTR_META_DATA_2	139
4.1.7.1.18	DS_REPL_KCC_DSA_FAILURESW	140
4.1.7.1.19	DS_REPL_KCC_DSA_FAILUREW	140
4.1.7.1.20	DS_REPL_PENDING_OPSW	141
4.1.7.1.21	DS_REPL_OPW	141
4.1.7.1.22	DS_REPL_ATTR_VALUE_META_DATA	142
4.1.7.1.23	DS_REPL_VALUE_META_DATA	142
4.1.7.1.24	DS_REPL_ATTR_VALUE_META_DATA_2	143
4.1.7.1.25	DS_REPL_VALUE_META_DATA_2	144
4.1.7.1.26	DS_REPL_CLIENT_CONTEXTS	145
4.1.7.1.27	DS_REPL_CLIENT_CONTEXT	145
4.1.7.1.28	DS_REPL_SERVER_OUTGOING_CALLS	146

4.1.7.1.29 DS_REPL_SERVER_OUTGOING_CALL	146
4.1.7.2 Method-Specific Abstract Types and Procedures	147
4.1.7.2.1 GetDNFromInvocationID	147
4.1.7.2.2 GetDNFromObjectGuid	147
4.1.7.2.3 GetNCs	148
4.1.7.2.4 GetUpToDatenessVector	148
4.1.7.3 Server Behavior of the IDL_DRSGetReplInfo Method	148
4.1.7.4 Examples of the IDL_DRSGetReplInfo Method	160
4.1.7.4.1 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_NEIGHBORS to find replication neighbors for a specified NC	160
4.1.7.4.1.1 Initial State	160
4.1.7.4.1.2 Client Request	161
4.1.7.4.1.3 Server Response	161
4.1.7.4.1.4 Final State	162
4.1.7.4.2 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_NEIGHBORS to find which naming contexts a DC receives updates for from a replication neighbor	162
4.1.7.4.2.1 Initial State	162
4.1.7.4.2.2 Client Request	164
4.1.7.4.2.3 Server Response	164
4.1.7.4.2.4 Final State	166
4.1.7.4.3 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_REPSTO to find replication neighbors for a specified NC	167
4.1.7.4.3.1 Initial State	167
4.1.7.4.3.2 Client Request	167
4.1.7.4.3.3 Server Response	167
4.1.7.4.3.4 Final State	168
4.1.7.4.4 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_CURSORS_3_FOR_NC	169
4.1.7.4.4.1 Initial State	169
4.1.7.4.4.2 Client Request	169
4.1.7.4.4.3 Server Response	169
4.1.7.4.4.4 Final State	170
4.1.7.4.5 Calling IDL_DRSGetReplInfo with infoType DS_REPL_INFO_METADATA_2_FOR_OBJ	170
4.1.7.4.5.1 Initial State	170
4.1.7.4.5.2 Client Request	171
4.1.7.4.5.3 Server Response	171
4.1.7.4.5.4 Final State	174
4.1.7.4.6 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	174
4.1.7.4.6.1 Initial State	174
4.1.7.4.6.2 Client Request	174
4.1.7.4.6.3 Server Response	175
4.1.7.4.6.4 Final State	177
4.1.7.4.7 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	177
4.1.7.4.7.1 Initial State	177
4.1.7.4.7.2 Client Request	177
4.1.7.4.7.3 Server Response	178
4.1.7.4.7.4 Final State	178

4.1.7.4.8	Calling IDL_DRSGetReplInfo with infoType	
	DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES	179
4.1.7.4.8.1	Initial State	179
4.1.7.4.8.2	Client Request.....	179
4.1.7.4.8.3	Server Response	179
4.1.7.4.8.4	Final State	180
4.1.7.4.9	Calling IDL_DRSGetReplInfo with infoType	
	DS_REPL_INFO_PENDING_OPS	180
4.1.7.4.9.1	Initial State	180
4.1.7.4.9.2	Client Request.....	180
4.1.7.4.9.3	Server Response	180
4.1.7.4.9.4	Final State	181
4.1.8	IDL_DRSInitDemotion (Opnum 25).....	181
4.1.8.1	Method-Specific Concrete Types	182
4.1.8.1.1	DRS_MSG_INIT_DEMOTIONREQ	182
4.1.8.1.2	DRS_MSG_INIT_DEMOTIONREQ_V1	182
4.1.8.1.3	DRS_MSG_INIT_DEMOTIONREPLY.....	182
4.1.8.1.4	DRS_MSG_INIT_DEMOTIONREPLY_V1	183
4.1.8.2	Server Behavior of the IDL_DRSInitDemotion Method	183
4.1.9	IDL_DRSQuerySitesByCost (Opnum 24)	184
4.1.9.1	Method-Specific Concrete Types	184
4.1.9.1.1	DRS_MSG_QUERYsitesREQ.....	184
4.1.9.1.2	DRS_MSG_QUERYsitesREQ_V1	185
4.1.9.1.3	DRS_MSG_QUERYsitesREPLY	185
4.1.9.1.4	DRS_MSG_QUERYsitesREPLY_V1.....	185
4.1.9.1.5	DRS_MSG_QUERYsitesREPLYELEMENT_V1.....	186
4.1.9.2	Method-Specific Abstract Types and Procedures	186
4.1.9.2.1	ValidateSiteRDN.....	186
4.1.9.2.2	WeightedArc and WeightedArcSet	187
4.1.9.2.3	MinWeightPath.....	187
4.1.9.3	Server Behavior of the IDL_DRSQuerySitesByCost Method.....	187
4.1.9.4	Examples of IDL_DRSQuerySitesByCost Method	190
4.1.9.4.1	Nontransitive Communication Using siteLinkBridge	190
4.1.9.4.1.1	Initial State	191
4.1.9.4.1.2	Client Request.....	197
4.1.9.4.1.3	Server Response	197
4.1.9.4.1.4	Final State	197
4.1.9.4.2	Transitive Communication	198
4.1.9.4.2.1	Initial State	198
4.1.9.4.2.2	Client Request.....	203
4.1.9.4.2.3	Server Response	204
4.1.9.4.2.4	Final State.....	204
4.1.10	IDL_DRSRemoveDsDomain (Opnum 15).....	204
4.1.10.1	Method-Specific Concrete Types	205
4.1.10.1.1	DRS_MSG_RMDMNREQ	205
4.1.10.1.2	DRS_MSG_RMDMNREQ_V1.....	205
4.1.10.1.3	DRS_MSG_RMDMNREPLY	205
4.1.10.1.4	DRS_MSG_RMDMNREPLY_V1	206
4.1.10.2	Method-Specific Abstract Types and Procedures.....	206
4.1.10.2.1	HasNCReplicated	206
4.1.10.3	Server Behavior of the IDL_DRSRemoveDsDomain Method	206
4.1.11	IDL_DRSRemoveDsServer (Opnum 14)	208
4.1.11.1	Method-Specific Concrete Types	208

4.1.11.1.1	DRS_MSG_RMSVRREQ	208
4.1.11.1.2	DRS_MSG_RMSVRREQ_V1	209
4.1.11.1.3	DRS_MSG_RMSVRREPLY	209
4.1.11.1.4	DRS_MSG_RMSVRREPLY_V1	209
4.1.11.2	Server Behavior of the IDL_DRSRemoveDsServer Method	210
4.1.12	IDL_DRSReplicaAdd (Opnum 5)	212
4.1.12.1	Method-Specific Concrete Types	212
4.1.12.1.1	DRS_MSG_REPADD	212
4.1.12.1.2	DRS_MSG_REPADD_V1	213
4.1.12.1.3	DRS_MSG_REPADD_V2	213
4.1.12.2	Server Behavior of the IDL_DRSReplicaAdd Method	214
4.1.13	IDL_DRSReplicaDel (Opnum 6)	217
4.1.13.1	Method-Specific Concrete Types	217
4.1.13.1.1	DRS_MSG_REPDEL	217
4.1.13.1.2	DRS_MSG_REPDEL_V1	217
4.1.13.2	Server Behavior of the IDL_DRSReplicaDel Method	218
4.1.14	IDL_DRSReplicaDemotion (Opnum 26)	221
4.1.14.1	Method-Specific Concrete Types	221
4.1.14.1.1	DRS_MSG_REPLICA_DEMOTIONREQ	221
4.1.14.1.2	DRS_MSG_REPLICA_DEMOTIONREQ_V1	222
4.1.14.1.3	DRS_MSG_REPLICA_DEMOTIONREPLY	222
4.1.14.1.4	DRS_MSG_REPLICA_DEMOTIONREPLY_V1	223
4.1.14.2	Method-Specific Abstract Types and Procedures	223
4.1.14.2.1	ReplicationPartners()	223
4.1.14.2.2	AbandonAllFSMORoles()	223
4.1.14.2.3	ReplicateOffChanges()	224
4.1.14.3	Server Behavior of the IDL_DRSReplicaDemotion Method	224
4.1.15	IDL_DRSReplicaModify (Opnum 7)	225
4.1.15.1	Method-Specific Concrete Types	226
4.1.15.1.1	DRS_MSG_REPMOD	226
4.1.15.1.2	DRS_MSG_REPMOD_V1	226
4.1.15.2	Server Behavior of the IDL_DRSReplicaModify Method	227
4.1.16	IDL_DRSReplicaSync (Opnum 2)	229
4.1.16.1	Method-Specific Concrete Types	229
4.1.16.1.1	DRS_MSG_REPSYNC	229
4.1.16.1.2	DRS_MSG_REPSYNC_V1	229
4.1.16.2	Server Behavior of the IDL_DRSReplicaSync Method	230
4.1.17	IDL_DRSReplicaVerifyObjects (Opnum 22)	231
4.1.17.1	Method-Specific Concrete Types	232
4.1.17.1.1	DRS_MSG_REPVERIFYOBJ	232
4.1.17.1.2	DRS_MSG_REPVERIFYOBJ_V1	232
4.1.17.2	Method-Specific Abstract Types and Procedures	233
4.1.17.2.1	GetRemoteUTD	233
4.1.17.2.2	ObjectExistsAtDC	233
4.1.17.3	Server Behavior of the IDL_DRSReplicaVerifyObjects Method	233
4.1.17.4	Examples of the IDL_DRSReplicaVerifyObjects Method	235
4.1.17.4.1	Initial State	235
4.1.17.4.2	Client Request	238
4.1.17.4.3	Server Response	238
4.1.17.4.4	Final State	238
4.1.18	IDL_DRSUnbind (Opnum 1)	239
4.1.18.1	Server Behavior of the IDL_DRSUnbind Method	240
4.1.19	IDL_DRSUpdateRefs (Opnum 4)	240

4.1.19.1	Method-Specific Concrete Types	241
4.1.19.1.1	DRS_MSG_UPDREFS	241
4.1.19.1.2	DRS_MSG_UPDREFS_V1	241
4.1.19.2	Server Behavior of the IDL_DRSUpdateRefs Method	241
4.1.19.3	Examples of the IDL_DRSUpdateRefs Method	243
4.1.19.3.1	Adding a repsTo Entry	243
4.1.19.3.1.1	Initial State	243
4.1.19.3.1.2	Client Request	243
4.1.19.3.1.3	Server Response	244
4.1.19.3.1.4	Final State	244
4.1.19.3.2	Replacing a repsTo Entry	245
4.1.19.3.2.1	Initial State	245
4.1.19.3.2.2	Client Request	245
4.1.19.3.2.3	Server Response	246
4.1.19.3.2.4	Final State	246
4.1.20	IDL_DRSWriteSPN (Opnum 13)	246
4.1.20.1	Method-Specific Concrete Types	247
4.1.20.1.1	DRS_MSG_SPNREQ	247
4.1.20.1.2	DRS_MSG_SPNREQ_V1	247
4.1.20.1.3	DRS_MSG_SPNREPLY	248
4.1.20.1.4	DRS_MSG_SPNREPLY_V1	248
4.1.20.1.5	DS_SPN_OPERATION	248
4.1.20.2	Method-Specific Abstract Types and Procedures	249
4.1.20.2.1	ExecuteWriteSPNRemotely	249
4.1.20.3	Server Behavior of the IDL_DRSWriteSPN Method	249
4.2	dsaop RPC Interface	251
5	Common Data Types, Variables, and Procedures	253
5.1	AbstractPTFromConcretePT	253
5.2	AccessCheckAttr	253
5.3	AccessCheckCAR	254
5.4	AccessCheckObject	254
5.5	AccessCheckWriteToSpnAttribute	254
5.6	AmIRODC	255
5.7	AttributeStamp	256
5.8	AttributeSyntax	257
5.9	AttrStamp	257
5.10	ATTRTYP	257
5.11	AttrtypFromSchemaObj	258
5.12	ATTRVAL	258
5.12.1	Abstract Value Representations	258
5.12.1.1	Object(DS-DN)	259
5.12.1.2	Object(DN-String)	260
5.12.1.3	Object(DN-Binary)	260
5.12.1.4	Object(Access-Point)	260
5.12.1.5	Object(OR-Name)	261
5.12.1.6	String(NT-Sec-Desc)	261
5.12.1.7	String(Sid)	261
5.12.1.8	String(Teletex)	261
5.12.2	ATTRTYP-to-OID Conversion	261
5.13	BOOL	267
5.14	BYTE	267
5.15	ClientExtensions	267

5.16	ClientUUID	267
5.17	ConcretePTFromAbstractPT	267
5.18	ConfigNC	268
5.19	dc, DC	268
5.20	DefaultNC	269
5.21	DelSubRef	269
5.22	DescendantObject	269
5.23	DN	270
5.24	DNBinary	270
5.25	DomainNameFromNT4AccountName	270
5.26	DRS_EXTENSIONS	270
5.27	DRS_EXTENSIONS_INT	270
5.28	DRS_HANDLE	275
5.29	DRS_OPTIONS	275
5.30	DRS_SPN_CLASS	277
5.31	DS_REPL_OP_TYPE	277
5.32	DSABObj	278
5.33	DSA_RPC_INST	278
5.34	DSName	278
5.35	DSNAME	279
5.35.1	DSNAME Equality	280
5.36	DSTIME	281
5.37	DWORD	281
5.38	Expunge	281
5.39	FILETIME	282
5.40	FindCharRev	282
5.41	FOREST_TRUST_INFORMATION	282
5.41.1	Record	283
5.41.2	Determining If a Name Is in a Trusted Forest	285
5.42	FOREST_TRUST_RECORD_TYPE	293
5.43	ForestRootDomainNC	293
5.44	FullReplicaExists	293
5.45	GetAttrVals	294
5.46	GetDefaultObjectCategory	294
5.47	GetDSNameFromDN	294
5.48	GetFSMORoleOwner	294
5.49	GetInstanceNameFromSPN	295
5.50	GetObjectNC	295
5.51	GetServiceClassFromSPN	295
5.52	GetServiceNameFromSPN	295
5.53	groupType Bit Flags	295
5.54	GUID	296
5.55	GuidFromString	296
5.56	GuidToString	297
5.57	handle_t	297
5.58	instanceType Bit Flags	297
5.59	Is2PartSPN	297
5.60	Is3PartSPN	297
5.61	IsAdIds	298
5.62	IsBuiltinPrincipal	298
5.63	IsDomainNameInTrustedForest	298
5.64	IsDomainSidInTrustedForest	298
5.65	IsDCAccount	298

5.66	IsForwardLinkAttribute	298
5.67	IsGC.....	299
5.68	IsGUIDBasedDNSName.....	299
5.69	IsMemberOfBuiltinAdminGroup	299
5.70	IsServerExtensionsChanged	299
5.71	IsUPNInTrustedForest.....	299
5.72	IsValidServiceName	300
5.73	KCCFailedConnections	300
5.74	KCCFailedLinks	300
5.75	LARGE_INTEGER.....	300
5.76	LDAP_CONN_PROPERTIES	300
5.77	LDAPConnections	301
5.78	LinkStamp	302
5.79	LinkValueStamp.....	302
5.80	LinkValueStampCompare	302
5.81	LONG	303
5.82	LONGLONG	303
5.83	LPWSTR.....	304
5.84	MakeAttid	304
5.85	MergeUTD.....	304
5.86	MTX_ADDR	304
5.87	NetworkAddress.....	305
5.88	NewPrefixTable	305
5.89	NT4SID.....	305
5.90	NTSAPI_CLIENT_GUID.....	305
5.91	NTDSTRANSPORT_OPT Values	305
5.92	NULLGUID.....	306
5.93	ObjExists	306
5.94	OID.....	306
5.95	OID_t.....	306
5.96	OidFromAttid	306
5.97	parent	306
5.98	PARTIAL_ATTR_VECTOR_V1_EXT.....	307
5.99	partialAttributeSet	307
5.100	PartialGCReplicaExists.....	307
5.101	PAS_DATA.....	307
5.102	PerformReplication	308
5.103	PrefixTable	308
5.104	PrefixTableEntry.....	308
5.105	RDN	308
5.106	rdnType	309
5.107	RemoveObj	309
5.108	ReplicatedAttributes	309
5.109	ReplicationQueue	309
5.110	REPLTIMES.....	310
5.111	replUpToDateVector, ReplUpToDateVector	311
5.112	REPS_FROM	311
5.113	REPS_TO.....	314
5.114	repsFrom, RepsFrom	317
5.115	repsTo, RepsTo	318
5.116	Rid	319
5.117	Right	319
5.118	RIGHT Values	319

5.119	RPCClientContexts.....	319
5.120	RPCOutgoingContexts.....	320
5.121	sAMAccountType Values.....	320
5.122	SCHEMA_PREFIX_TABLE.....	321
5.123	SchemaNC.....	321
5.124	SchemaObj.....	321
5.125	ServerExtensions.....	321
5.126	SID.....	321
5.127	SidFromStringSid.....	322
5.128	StampLessThanOrEqualUTD.....	322
5.129	StartsWith.....	322
5.130	StringSidFromSid.....	322
5.131	SubString.....	322
5.132	Syntax.....	323
5.133	SYNTAX_ADDRESS.....	323
5.134	SYNTAX_DISTNAME_BINARY.....	323
5.135	systemFlags Values.....	325
5.136	UCHAR.....	325
5.137	ULONG.....	325
5.138	ULONGLONG.....	325
5.139	UPTODATE_CURSOR_V1.....	325
5.140	UPTODATE_CURSOR_V2.....	326
5.141	UPTODATE_VECTOR_V1_EXT.....	326
5.142	UPTODATE_VECTOR_V2_EXT.....	327
5.143	userAccountControl Bits.....	327
5.144	UserNameFromNT4AccountName.....	328
5.145	USHORT.....	328
5.146	USN.....	328
5.147	USN_VECTOR.....	328
5.148	UUID.....	329
5.149	ValidateDRSDemotionInput.....	329
5.150	ValidateDRSInput.....	329
5.151	Value.....	330
5.152	WCHAR.....	330
6	Security.....	331
6.1	Security Considerations for Implementers.....	331
6.2	Index of Security Parameters.....	331
7	Appendix A: Full IDL.....	332
8	Appendix B: Product Behavior.....	348
9	Change Tracking.....	355
10	Index.....	357

1 Introduction

The Directory Replication and Data Management (DRDM) Remote Protocol is an **RPC** protocol that is used for management of server-to-server replication in Active Directory and management of data in **Active Directory**. A portion of Microsoft's implementation of the DRDM Remote Protocol is used to communicate between servers, and the other portion deals with client-to-server communications. Those server-to-server communications are not used by Microsoft to communicate with Microsoft Windows® client operating systems and are not included in this specification. Licensees can implement server-to-server directory replication using any protocol they choose. This specification describes the client-to-server portions of the DRDM Remote Protocol that are used between Windows servers and Windows client operating systems to diagnose, monitor, and manage Windows directory replication and directory data. In some cases, the client-to-server communications include replication data that might be presented to administrators in order to assist in diagnosing or monitoring the directory replication. However, the specific content of this data is not understood by Windows client operating systems, and the structure of the data is not prescribed by this specification. Interoperation with Windows client operating systems does not require the use of specific structures within these data elements. Licensees can implement the DRDM Remote Protocol to provide and accept any data that is meaningful for diagnosing or monitoring their server-to-server directory replication, or no data at all, as they choose.

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 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.

The name of each drsuapi method begins with "IDL_DRS". There are eight methods that are not outlined in this document: methods with **opnums** 3, 8, 9, 10, 11, 17, 21, and 23. These undocumented methods perform special server-to-server operations, such as **object** addition, object verification and lookup, **replication**, and **flexible single master operation (FSMO)** tasks.

Some functionality exposed by this RPC protocol 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
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)
nonreplicated 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 section 1, Introduction.) 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.

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

class: See **object class**.

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.

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 the full domain 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 section [1](#), Introduction. 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 3.1.2.1.5.2, 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.

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**). Given a **DSName**, an **object** can be identified within a set of **NC replicas** according to the matching rules defined in section 5.34.

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.

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 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**. The server-to-server methods required to implement any aspect of a **FSMO role**, or to transfer a **FSMO role** from one **DC** to another **DC**, are not included in this document and are not required for interoperation with Windows client operating systems.

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 server-to-server **replication** and to translate those trees into settings of variables that implement the **replication** topology. The

implementation details of this component are not included in this document and are not required for interoperation with Windows client operating systems.

lingering object: An **object** that still exists in an **NC replica** even though it has been deleted and "**garbage collected**" from other **replicas**. These **objects** are a consequence of a characteristic of the server-to-server **replication** implementation. **Lingering objects** can occur in this implementation, for example, when a **DC** goes offline for longer than the **tombstone lifetime**. The specific details of the implementation that can create a **lingering object** are not included in this document and are not required for interoperation with Windows client operating systems.

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**.

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 section [1](#), Introduction.

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

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 section [1](#), Introduction. 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 3.1.1.8.

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 section [1](#), Introduction.

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 3.1.1.8.1.

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 section [1](#), Introduction.

replicated attribute: An **attribute** whose values are replicated. See **replication**.

replicated update: An **update** performed to an **NC replica** by the implemented **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**. Neither this stamp nor any other specific aspects of a replicated update are required for interoperation with Windows client operating systems.

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. The description and details of the methods used for this server-to-server implementation are not included in this document and are not required for interoperation with Windows client operating systems.

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.

schema naming context (schema NC): A specific type of **NC** that contains **schema objects** representing the **schema**. A **forest** has a single **schema NC**, which is replicated to each **DC** in the **forest**. Each **attribute** and **class** in the **forest's schema** is represented as a corresponding **object** in the **forest's schema NC**.

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

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

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

security identifier (SID): An identifier for a **security principal object**. The **SID** is composed of an account authority portion (typically corresponding to a **domain**) and an integer representing an identity relative to the account authority, termed the **RID**. The **SID** format is specified in [\[MS-DTYP\]](#) section 2.4.2; a string representation of a **SID** is specified in [\[MS-WSO\]](#) 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.

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

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

site of a DC: The **site object** that is an **ancestor** of a **DC's DSA object**. See **site object**.

stamp: Information describing an **originating update** by a **DC**. The **stamp** is not the new data value; the **stamp** is information about the **update** that created the new data value. A **stamp** is often called metadata, because it is additional information that "talks about" the actual data values. Neither this stamp nor any other specific aspects of a replicated update are required for interoperability with Windows client operating systems.

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: A structure in the Microsoft implementation of server-to-server **replication** that is a representation of an assertion about the presence of **originating updates** from different sources in an **NC replica**. This structure is used in the server-to-server replication implementation and is not required for interoperation with Windows client operating systems. See **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. This structure is used in the server-to-server replication implementation, and is not required for interoperation with Windows client operating systems. 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.

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.

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

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

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

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

[RFC1034] Mockapetris, P., "Domain Names - Concepts and Facilities", STD 13, RFC 1034, November 1987, <http://www.ietf.org/rfc/rfc1034.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>

[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>

[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>

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](#)".

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

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

1.3 Overview

This document specifies the Directory Replication and Data Management (DRDM) Remote Protocol, an RPC protocol for replication between **domain controllers** and management of Active Directory. Only the methods that are used to communicate between a Microsoft Windows® client and a Windows server are defined in this document. Some of the methods that are defined in this document are used to diagnose, monitor, or manage Windows directory replication services and are not required for interoperation with Windows clients. The protocol consists of two RPC interfaces, named drsuapi and dsaop. The name of each drsuapi method begins with "IDL_DRS". The names of the dsaop methods are not documented here and are not required for interoperation with Windows client operating systems.

1.3.1 Methods Categorized by Function

The DRDM Remote Protocol contains 30 methods, ten of which are not documented here and are not required for interoperation with Microsoft Windows® client operating systems. The documented 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.
- Replication methods: IDL_DRSReplicaSync, IDL_DRSReplicaVerifyObjects, IDL_DRSGetReplInfo. The IDL_DRSReplicaSync and IDL_DRSReplicaVerifyObjects methods cause the server to initiate server-to-server replication. The IDL_DRSGetReplInfo method is used to gather information about the details of the server-to-server replication implementation. These methods are exposed only to monitor, diagnose, and manage the replication implementation and are not required for interoperation with Windows clients.
- Lookup: IDL_DRSCrackNames. This method performs **directory** lookups.
- 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**.
- **Knowledge Consistency Checker (KCC)** support methods: IDL_DRSUpdateRefs, IDL_DRSReplicaAdd, IDL_DRSReplicaDel, IDL_DRSReplicaModify, IDL_DRSExecuteKCC. These methods can be used to create a replication topology, and by administrator tools to manage that replication topology. These methods are exposed only to monitor, diagnose, and manage the replication topology implementation and are not required for interoperation with Windows clients.
- Administrator-tool support methods: IDL_DRSAddSidHistory, IDL_DRSRemoveDsServer, IDL_DRSRemoveDsDomain, 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 in [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 \(section 4.1.18\)](#), 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_DRSUnbind**. 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_DRSUnbind**, 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.

- [IDL_DRSInitDemotion \(section 4.1.8\)](#) 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_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 a server-to-server replication 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](#) ([4.1.7.3](#)) for exact details.

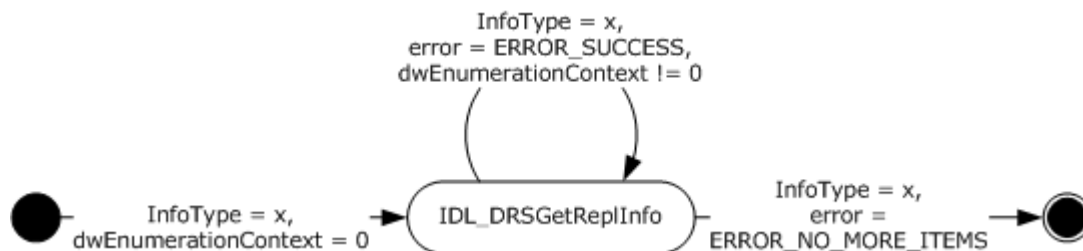


Figure 1: Using `dwEnumerationContext`

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 this protocol contains a [DSNAME](#) either in its request or its response.

1.4 Relationship to Other Protocols

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**: [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 the server-to-server replication implementation. These operations are not required for interoperability with Microsoft Windows® clients and are exposed only for diagnostics, monitoring, and management of the server-to-server replication implementation.
- Object constructed attributes: [canonicalName](#) (this exposes some functionality of IDL_DRSCrackNames), [tokenGroups](#), [tokenGroupsNoGCAcceptable](#), [tokenGroupsGlobalAndUniversal](#) (these expose some functionality of the implementation of **group** expansion).
- Controls: LDAP_SERVER_DIRSYNC_OID

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

- The function of IDL_DRSSetSPN can be performed as an LDAP Modify of the [servicePrincipalName](#) attribute.

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 the management of objects in a directory 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**: The drsuapi protocol interface described in this document has a single version number of 4.0.
- **Security and Authentication Methods**: See [\[MS-RPCE\]](#) section 1.7.

- **Capability Negotiation:** This protocol does explicit capability negotiation as described in [IDL DRSBind \(section 4.1.2\)](#) 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 4.1.20

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.

2.2 Protocol Security

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

2.2.1 General Background

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.

The specific SPNs produced by a client in each scenario are described in the following sections.

2.2.3 DC-to-DC Operations

The security and mutual authentication requirements for those DRDM Remote Protocol operations that involve DC-to-DC interactions are implementation-specific. [<2>](#)

2.2.4 Client-to-DC Operations

This section describes the security and mutual authentication requirements for those DRDM 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** (as specified in [\[MS-RPCE\]](#) section 2.2.1.1.8) 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 DRDM 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>"

GUID is defined in [\[MS-DTYP\]](#) section 2.3.2. 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.

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 DRDM 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.

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. 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 RPC interface 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 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 RPC interface.

- 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 5.10](#)) 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 := "\x55\x06\x02"`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Therefore:

- `S := {}`

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

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

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

is false.

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

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

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

- `stringSet := stringArrayPtr^`

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

3.4.4 Common Language Constructs

The syntax of standard control structures is as follows:

```
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

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 following assignment removes the attr attribute from object o:

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

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 following assignment assigns the value null to the integer variable i:

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

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, the following is a valid pseudocode sequence:

```
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

```

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 following statement is a specification error if the attr attribute is not present on object *o*:

```

i := o!attr * 2

```

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

in the LDP transcript is changed to:

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

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

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

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

3.6 Server and Client Initialization

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

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](#).

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. Not required for interoperation with Windows clients. Opnum: 2
Opnum3ServerToServerOnly	Opnum: 3
IDL_DRSUpdateRefs	Adds or deletes a value from the repsTo attribute of a specified NC replica. Not required for interoperation with Windows clients. Opnum: 4
IDL_DRSReplicaAdd	Adds a replication source reference for the specified NC. Not required for interoperation with Windows clients. Opnum: 5
IDL_DRSReplicaDel	Deletes a replication source reference for the specified NC. Not required for interoperation with Windows clients. Opnum: 6
IDL_DRSReplicaModify	Updates the value for repsFrom for the NC replica. Not required for interoperation with Windows clients. Opnum: 7
Opnum8ServerToServerOnly	Opnum: 8
Opnum9ServerToServerOnly	Opnum: 9
Opnum10ServerToServerOnly	Opnum: 10
Opnum11ServerToServerOnly	Opnum: 11
IDL_DRSCrackNames	Looks up each of a set of objects in the directory and returns it to the caller in the requested format. Opnum: 12

Method	Description
IDL_DRSWriteSPN	Updates the set of service principal names (SPNs) on an object. Opnum: 13
IDL_DRSRemoveDsServer	Removes the representation of a DC from the directory. Not required for interoperation with Windows clients. Opnum: 14
IDL_DRSRemoveDsDomain	Removes the representation of a domain from the directory. Not required for interoperation with Windows clients. Opnum: 15
IDL_DRSDomainControllerInfo	Retrieves information about DCs in a given domain. Opnum: 16
Opnum17ServerToServerOnly	Opnum: 17
IDL_DRSExecuteKCC	Validates the replication interconnections of DCs and updates them if necessary. Not required for interoperation with Windows clients. Opnum: 18
IDL_DRSGetReplInfo	Retrieves the implemented replication state of the server. Not required for interoperation with Windows clients. Opnum: 19
IDL_DRSAddSidHistory	Adds one or more SIDs to the sIDHistory attribute of a given object. Opnum: 20
Opnum21ServerToServerOnly	Opnum: 21
IDL_DRSReplicaVerifyObjects	Verifies the existence of objects in an NC replica. Not required for interoperation with Windows clients. Opnum: 22
Opnum23ServerToServerOnly	Opnum: 23
IDL_DRSQuerySitesByCost	Determines the communication cost from a "from" site to one or more "to" sites. Opnum: 24
IDL_DRSInitDemotion	Performs the first phase of the removal of a DC from an AD LDS forest. Not required for interoperation with Windows clients. Opnum: 25
IDL_DRSReplicaDemotion	Replicates off all changes to the specified NC and moves any FSMOs held to another server. Not required for interoperation with Windows clients. Opnum: 26
IDL_DRSFinishDemotion	Finishes or cancels the removal of a DC from an AD LDS forest. Not required for interoperation with Windows clients. 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>](#)

Note that gaps in the opnum numbering sequence represent opnums that are not required for interoperation with Windows client operating systems. [<6>](#)

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_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.

4.1.1.1 Method-Specific Concrete Types

4.1.1.1.1 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.

4.1.1.1.2 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. 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 SAM name. Otherwise, it is a DN.

4.1.1.1.3 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.

4.1.1.1.4 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.

4.1.1.1.5 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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	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 client connection 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;
```

4.1.1.2 Method-Specific Abstract Types and Procedures

4.1.1.2.1 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.

4.1.1.2.2 ConnectToDC

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

Creates a [ConnectionCtx](#) for the DC named by dcname, associating the credentials of the client's security context 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.

4.1.1.2.3 ConnectToDCWithCreds

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

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

4.1.1.2.4 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. It does nothing if auditing is not enabled. The content of the audit event is an implementation-specific behavior.

4.1.1.2.5 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. It does nothing if auditing is not enabled. The content of the audit event is an implementation-specific behavior.

4.1.1.2.6 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. [<7>](#)

4.1.1.2.7 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.

4.1.1.2.8 GetPDC

```
procedure GetPDC(domainName: uncodestring): uncodestring
```

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.

4.1.1.2.9 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.

4.1.1.2.10 IsAuditingEnabled

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

Returns true if auditing on the server on which it is called is enabled, and returns false otherwise.

4.1.1.2.11 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.

4.1.1.2.12 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. [<8>](#)

4.1.1.2.13 IsAuditingGroupPresent

```
procedure IsAuditingGroupPresent(dcname: uncodestring, nETBIOSName: uncodestring): 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.

4.1.1.2.14 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 3.1.2.1.3 whose value is less than 1000, and returns false otherwise.

4.1.1.2.15 LastRID

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

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

4.1.1.2.16 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.

4.1.1.3 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

ValidatedDRSInput(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
 * DNSs. */
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

/* 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^.Vl.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^.Vl.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^.Vl.dwWin32Error := ERROR_ACCESS_DENIED
        return 0
    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^.Vl.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^.Vl.SrcDomain = null) or
    (pmsgIn^.Vl.SrcDomain = "") or
    (pmsgIn^.Vl.DstDomain = null) or
    (pmsgIn^.Vl.DstDomain = "") or
    (pmsgIn^.Vl.SrcCredsUserLength > 0 and
        pmsgIn^.Vl.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
    srcCtx := 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 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

```

4.1.1.4 Examples of the IDL_DRSAddSidHistory Method

4.1.1.4.1 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.

4.1.1.4.1.1 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

4.1.1.4.1.2 Server Response

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

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

4.1.1.4.1.3 Final State

There are no changes in state.

4.1.1.4.2 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".

4.1.1.4.2.1 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;

4.1.1.4.2.2 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"

4.1.1.4.2.3 Server Response

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

- pdwOutVersion = 1

- pmsgOut = DRS_MSG_ADDSIDREPLY_V1
 - dwWin32Error: 0

4.1.1.4.2.4 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:

4.1.1.4.3 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.

4.1.1.4.3.1 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;

4.1.1.4.3.2 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"

4.1.1.4.3.3 Server Response

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

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

4.1.1.4.3.4 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;

4.1.2 IDL_DRSBind (Opnum 0)

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

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

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

puuidClientDsa: A pointer to 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.

4.1.2.1 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`.[<9>](#)

Windows non-DC clients use `pextClient` to pass a properly initialized [DRS_EXTENSIONS_INT](#) structure to the server. The Windows non-DC client sets the **dwReplEpoch** field of the [DRS_EXTENSIONS_INT](#) structure to zero. A nonzero value for this field is used only by server-to-server implemented methods. This field **MAY** have meaning to peer Windows servers, but its meaning is not significant to Windows client operating systems.

Windows non-DC client callers **SHOULD** set the **dwFlags** field of the [DRS_EXTENSIONS_INT](#) structure to zero and **SHOULD** also set all `dwFlagsExt` bits to zero. Additionally, Windows non-DC client callers **SHOULD** always set the **SiteObjGuid** field of the [DRS_EXTENSIONS_INT](#) structure to the **NULL GUID** value, set the **Pid** field of the [DRS_EXTENSIONS_INT](#) structure to an

implementation-specific, client-local process identifier (PID), set the **cb** field equal to the size as detailed in section 5.26, and SHOULD set the **ConfigObjGUID** field of the [DRS_EXTENSIONS_INT](#) structure to zero. The information in the [DRS_EXTENSIONS_INT](#) structure 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. <10>

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* = 2 to [IDL_DRSGetReplInfo](#), the client also passes a [DRS_MSG_GETREPLINFO_REQ_V2](#) 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* = 2 from [IDL_DRSDomainControllerInfo](#), the server also returns a [DRS_MSG_DCINFOREPLY_V2](#) response.

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

- **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	Response version
1	-

[IDL_DRSUpdateRefs](#)

Request version	Response version
1	-

[IDL_DRSReplicaAdd](#)

Request version	Response version
1	-
2	-

[IDL_DRSReplicaDel](#)

Request version	Response version
1	-

[IDL_DRSReplicaModify](#)

Request version	Response version
1	-

[IDL_DRSCrackNames](#)

Request version	Response version
1	1

[IDL_DRSSetSPN](#)

Request version	Response version
1	1

[IDL_DRSRemoveDsServer](#)

Request version	Response version
1	1

[IDL_DRSRemoveDsDomain](#)

Request version	Response version
1	1

[IDL_DRSDomainControllerInfo](#)

Request version	Response version
1	request.InfoLevel ¹

[IDL_DRSExecuteKCC](#)

Request version	Response version
1	1

[IDL_DRSGetReplInfo](#)

Request version	Response version
1	request.InfoType ²

Request version	Response version
2	request.InfoType ²

[IDL_DRSAddSidHistory](#)

Request version	Response version
1	1

[IDL_DRSReplicaVerifyObjects](#)

Request version	Response version
1	-

[IDL_DRSQuerySitesByCost](#)

Request version	Response version
1	1

[IDL_DRSInitDemotion](#)

Request version	Response version
1	1

[IDL_DRSReplicaDemotion](#)

Request version	Response version
1	1

[IDL_DRSFinishDemotion](#)

Request version	Response version
1	1

¹ Possible values are 0x1, 0x2, and 0xffffffff (see section [4.1.4](#)).

² Possible values are detailed in section [4.1.7](#).

4.1.2.2 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** field is initialized by reading the value of [msDS-ReplicationEpoch](#) from its [nTDSDSA](#) object and assigning this value to it, and whose other fields describe the server. [<11>](#) The server then returns a [DRS_HANDLE](#) as the referent of phDrs. The **dwReplEpoch** and **ConfigObjectGUID** fields are for

server-to-server replication implementation only, the client does not interpret them. The fields MAY have meaning to Microsoft Windows® server implementations, but their meaning is not significant to Windows clients.

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.

A server supports all requests via the RPC transport.

[IDL_DRSReplicaSync](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[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_DRSCrackNames](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

[IDL_DRSWriteSPN](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

IDL_DRSRemoveDsServer

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_REMOVEAPI

IDL_DRSRemoveDsDomain

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_REMOVEAPI

IDL_DRSDomainControllerInfo

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

IDL_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	DRS_EXT_GETCHGREPLY_V6

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

[IDL_DRSAddSidHistory](#)

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_ADD_SID_HISTORY

[IDL_DRSReplicaVerifyObjects](#)

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

4.1.2.3 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 \(section 4.1.2.2\)](#).

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. 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.

4.1.2.4 Examples of the IDL_DRSBind Method

A client is binding to the directory server `DC1.CONTOSO.COM`.

4.1.2.4.1 Initial State

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, 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);
```

##### 4.1.2.4.2 Client Request

A client invokes the [IDL\\_DRSBind](#) method against `DC1`, with the following parameters ([DRS\\_HANDLE](#) to `DC1` omitted):

- `puuidClientDsa = GUID {e24d201a-4fd6-11d1-a3da-0000f875ae0d}`
- `pextClient`:
  - `cb`: 0x1c
  - `dwFlags`: 0
  - `SiteObjGuid`: NULL GUID
  - `Pid`: 0
  - `dwReplEpoch`: 0
  - `dwFlagsExt`: 0

- ConfigObjGUID: NULL GUID

#### 4.1.2.4.3 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

#### 4.1.2.4.4 Final State

No change in state.

### 4.1.3 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`.

### 4.1.3.1 Method-Specific Concrete Types

#### 4.1.3.1.1 DRS\_MSG\_CRACKREQ

The **DRS\_MSG\_CRACKREQ** union defines the request messages sent to the [IDL\\_DRS CrackNames](#) 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.

#### 4.1.3.1.2 DRS\_MSG\_CRACKREQ\_V1

The **DRS\_MSG\_CRACKREQ\_V1** structure defines the request message sent to the [IDL\\_DRS CrackNames](#) 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 | 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 | T | G | X | X | X | X | X  | X | X | X | X | X | X | X | X | X | X  | X | X | X | F | X | X | X | X | X | X  | X | X |
|   |   |   |   | R | C |   |   |   |   |    |   |   |   |   |   |   |   |   |   |    |   |   | P |   |   |   |   |   |   |    |   |   |
|   |   |   |   |   |   |   |   |   |   |    |   |   |   |   |   |   |   |   |   |    |   |   | O |   |   |   |   |   |   |    |   |   |

**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 \(section 4.1.3.1.3\)](#) or one of the following.

| Value                                             | Meaning                                                                                                                                                                           |
|---------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DS_LIST_SITES<br>0xFFFFFFFF                       | Get all sites in the forest.                                                                                                                                                      |
| DS_LIST_SERVERS_IN_SITE<br>0xFFFFFFFFE            | Get all servers in a given site.                                                                                                                                                  |
| DS_LIST_DOMAINS_IN_SITE<br>0xFFFFFFFFD            | Get all domains in a given site.                                                                                                                                                  |
| DS_LIST_SERVERS_FOR_DOMAIN_IN_SITE<br>0xFFFFFFFFC | Get all DCs of a specified domain in a given site.                                                                                                                                |
| DS_LIST_INFO_FOR_SERVER<br>0xFFFFFFFFB            | Get DNS host name and server reference for a given DC.                                                                                                                            |
| DS_LIST_ROLES<br>0xFFFFFFFFA                      | Get <b>FSMO role owners</b> .                                                                                                                                                     |
| DS_NT4_ACCOUNT_NAME_SANS_DOMAIN<br>0xFFFFFFFF9    | Get value of <a href="#">sAMAccountName</a> attribute.                                                                                                                            |
| DS_MAP_SCHEMA_GUID<br>0xFFFFFFFF8                 | Get LDAP display name from <b>schema GUID</b> . The given schema GUID should be in the curly braced GUID string format as specified in <a href="#">[MS-DTYP]</a> section 2.3.2.3. |
| DS_LIST_DOMAINS<br>0xFFFFFFFF7                    | Get all domains in the forest.                                                                                                                                                    |
| DS_LIST_NCS<br>0xFFFFFFFF6                        | Get all NCs in the forest.                                                                                                                                                        |
| DS_ALT_SECURITY_IDENTITIES_NAME<br>0xFFFFFFFF5    | Compares input names against the values of the <b>altSecurityIdentities</b> attribute.                                                                                            |
| DS_STRING_SID_NAME<br>0xFFFFFFFF4                 | String form of SID.                                                                                                                                                               |
| DS_LIST_SERVERS_WITH_DCS_IN_SITE<br>0xFFFFFFFF3   | Get all DCs in a given site.                                                                                                                                                      |
| DS_LIST_GLOBAL_CATALOG_SERVERS<br>0xFFFFFFFF1     | Get all <b>GCs</b> in the forest.                                                                                                                                                 |
| DS_NT4_ACCOUNT_NAME_SANS_DOMAIN_EX<br>0xFFFFFFFF0 | Get value of <a href="#">sAMAccountName</a> attribute; return status DS_NAME_ERROR_NOT_FOUND if account is invalid.                                                               |

| Value                                              | Meaning                                                                                                            |
|----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| DS_USER_PRINCIPAL_NAME_AND_ALTSECID<br>0xFFFFFFFFF | Compares input names against the user principal name and the values of the <b>altSecurityIdentities</b> 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<br>0xFFFFFFFF4               | String form of a SID. |
| DS_USER_PRINCIPAL_NAME_FOR_LOGON<br>0xFFFFFFFF2 | User principal name.  |

**cNames:** Count of items in the **rpNames** array.

**rpNames:** Input names to translate.

#### 4.1.3.1.3 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.

#### 4.1.3.1.4 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.

#### 4.1.3.1.5 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.

#### 4.1.3.1.6 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.

#### 4.1.3.1.7 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.

#### 4.1.3.1.8 DS\_NAME\_ERROR

This section enumerates the possible statuses of a translation operation.

| Symbolic name                                       | Description                                                                                                                                                                  |
|-----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0<br>DS_NAME_NO_ERROR                               | No error occurred during the name translation.                                                                                                                               |
| 1<br>DS_NAME_ERROR_RESOLVING                        | Generic processing error during the name translation.                                                                                                                        |
| 2<br>DS_NAME_ERROR_NOT_FOUND                        | The object with the specified name cannot be found.                                                                                                                          |
| 3<br>DS_NAME_ERROR_NOT_UNIQUE                       | More than one object is located with the specified name.                                                                                                                     |
| 4<br>DS_NAME_ERROR_NO_MAPPING                       | The desired output format cannot be applied to the object with the specified name.                                                                                           |
| 5<br>DS_NAME_ERROR_DOMAIN_ONLY                      | Only the domain part of the name was translated.                                                                                                                             |
| 7<br>DS_NAME_ERROR_TRUST_REFERRAL                   | The specified name belongs to a trusted forest, a referral is returned.                                                                                                      |
| 0xFFFFFFFF2<br>DS_NAME_ERROR_IS_SID_HISTORY_UNKNOWN | The specified name matches a value in the <b>sidHistory</b> attribute of an object, but the type of the object is unknown.                                                   |
| 0xFFFFFFFF3<br>DS_NAME_ERROR_IS_SID_HISTORY_ALIAS   | Translation was successful. The specified name matches a value in the <b>sidHistory</b> attribute of an object. The object's <b>sAMAccountType</b> attribute value is either |

| Symbolic name                                           | Description                                                                                                                                                                                                                                                                                                     |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                         | SAM_NON_SECURITY_ALIAS_OBJECT or SAM_ALIAS_OBJECT as defined in <a href="#">[MS-SAMR]</a> section 2.2.1.9, ACCOUNT_TYPE Values.                                                                                                                                                                                 |
| 0xFFFFFFFF4<br>DS_NAME_ERROR_IS_SID_HISTORY_GROUP       | Translation was successful. The specified name matches a value in the <b>sidHistory</b> attribute of an object. The object's <b>sAMAccountType</b> attribute value is either SAM_GROUP_OBJECT or SAM_NON_SECURITY_GROUP_OBJECT as defined in <a href="#">[MS-SAMR]</a> section 2.2.1.9, ACCOUNT_TYPE Values.    |
| 0xFFFFFFFF5<br>DS_NAME_ERROR_IS_SID_HISTORY_USER        | Translation was successful. The specified name matches a value in the <b>sidHistory</b> attribute of an object. The object's <b>sAMAccountType</b> attribute value is SAM_USER_OBJECT or SAM_MACHINE_ACCOUNT or SAM_TRUST_ACCOUNT as defined in <a href="#">[MS-SAMR]</a> section 2.2.1.9, ACCOUNT_TYPE Values. |
| 0xFFFFFFFF6<br>DS_NAME_ERROR_IS_SID_UNKNOWN             | The specified name matches the <b>objectSid</b> attribute of an object, but the type of the object is unknown.                                                                                                                                                                                                  |
| 0xFFFFFFFF7<br>DS_NAME_ERROR_IS_SID_ALIAS               | Translation was successful. The specified name matches the objectSid attribute of an object. The object's <b>sAMAccountType</b> attribute value is either SAM_NON_SECURITY_ALIAS_OBJECT or SAM_ALIAS_OBJECT as defined in <a href="#">[MS-SAMR]</a> section 2.2.1.9, ACCOUNT_TYPE Values.                       |
| 0xFFFFFFFF8<br>DS_NAME_ERROR_IS_SID_GROUP               | Translation was successful. The specified name matches the objectSid attribute of an object. The object's <b>sAMAccountType</b> attribute value is either SAM_GROUP_OBJECT or SAM_NON_SECURITY_GROUP_OBJECT as defined in <a href="#">[MS-SAMR]</a> section 2.2.1.9, ACCOUNT_TYPE Values.                       |
| 0xFFFFFFFF9<br>DS_NAME_ERROR_IS_SID_USER                | Translation was successful. The specified name matches the objectSid attribute of an object. The object's <b>sAMAccountType</b> attribute value is SAM_USER_OBJECT or SAM_MACHINE_ACCOUNT or SAM_TRUST_ACCOUNT as defined in <a href="#">[MS-SAMR]</a> section 2.2.1.9, ACCOUNT_TYPE Values.                    |
| 0xFFFFFFFFFA<br>DS_NAME_ERROR_SCHEMA_GUID_CONTROL_RIGHT | Translation was successful. The GUID identifies a control access right.                                                                                                                                                                                                                                         |
| 0xFFFFFFFFFB<br>DS_NAME_ERROR_SCHEMA_GUID_CLASS         | Translation was successful. The GUID identifies a classSchema object.                                                                                                                                                                                                                                           |
| 0xFFFFFFFFFC<br>DS_NAME_ERROR_SCHEMA_GUID_ATTR_SET      | Translation was successful. The GUID identifies a property set.                                                                                                                                                                                                                                                 |
| 0xFFFFFFFFFD<br>DS_NAME_ERROR_SCHEMA_GUID_ATTR          | Translation was successful. The GUID identifies an attributeSchema object.                                                                                                                                                                                                                                      |

| Symbolic name                                     | Description                                                       |
|---------------------------------------------------|-------------------------------------------------------------------|
| 0xFFFFFFFF<br>DS_NAME_ERROR_SCHEMA_GUID_NOT_FOUND | The GUID cannot be resolved.                                      |
| 0xFFFFFFFF<br>DS_NAME_ERROR_IS_FPO                | The object with the specified name is a Foreign Principal Object. |

### 4.1.3.2 Method-Specific Abstract Types and Procedures

#### 4.1.3.2.1 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\ndleaf" (where \n designates a newline character), this procedure returns null.

#### 4.1.3.2.2 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.

#### 4.1.3.2.3 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.

#### 4.1.3.2.4 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".

#### 4.1.3.2.5 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.

#### 4.1.3.2.6 DomainNameFromUPN

```
procedure DomainNameFromUPN(upn: uncodestring): uncodestring
```

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.

#### 4.1.3.2.7 DomainNetBIOSNameFromDomain

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

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.

#### 4.1.3.2.8 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.

#### 4.1.3.2.9 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, DNs).

```
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

```

```

 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
endif
endif
endfor
msgOut^.cItems := msgIn.cNames
endif

return ERROR_SUCCESS

```

#### 4.1.3.2.10 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
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
 endif

```

```

 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

```

#### 4.1.3.2.11 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

```

#### 4.1.3.2.12 LookupCanonicalName

```

procedure LookupCanonicalName(name: uncodestring): 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: uncodestring

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

```

#### 4.1.3.2.13 GetCanonicalName

```

procedure GetCanonicalName(
 obj: DSName, extended: boolean): uncodestring

```

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

```

result: uncodestring

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

```

#### 4.1.3.2.14 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
endif

return rt

```

#### 4.1.3.2.15 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
```

#### 4.1.3.2.16 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
```

#### 4.1.3.2.17 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

```

#### 4.1.3.2.18 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^ :=
 pReferencedDomains^.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

```

#### 4.1.3.2.19 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=alerter,appmgmt,cisvc" }
```

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

#### 4.1.3.2.20 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.

#### 4.1.3.2.21 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".

#### 4.1.3.2.22 UserNameFromUPN

```
procedure UserNameFromUPN(upn: unicodestring): unicodestring
```

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

#### 4.1.3.2.23 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))
 end

 return NtStatus

```

#### 4.1.3.2.24 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 \(section 4.1.3.2.10\)](#).

```

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

```

### 4.1.3.3 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
```

### 4.1.3.4 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.

#### 4.1.3.4.1 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:

- `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;

#### 4.1.3.4.2 Client Request

A client invokes the `IDL_DRSCrackNames` method against DC1 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"

#### 4.1.3.4.3 Server Response

Returns code of 0 and the following values:

- pdwMessageOut = 1
- pdwOut = 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

#### 4.1.3.4.4 Final State

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

### 4.1.4 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.

#### 4.1.4.1 Method-Specific Concrete Types

##### 4.1.4.1.1 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.

##### 4.1.4.1.2 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.

**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**.

##### 4.1.4.1.3 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.

#### 4.1.4.1.4 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.

#### 4.1.4.1.5 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.

#### 4.1.4.1.6 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;

```

```
} DRS_MSG_DCINFOREPLY_V3;
```

**cItems:** Count of items in the **rItems** array.

**rItems:** DC information.

#### 4.1.4.1.7 DRS\_MSG\_DCINFOREPLY\_VFFFFFFFF

The **DRS\_MSG\_DCINFOREPLY\_VFFFFFFFF** structure defines the response message received from the [IDL\\_DRSDomainControllerInfo](#) method, when the client has requested InfoLevel = 0xFFFFFFFF.

```
typedef struct {
 [range(0,10000)] DWORD cItems;
 [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_FFFFFFFF* rItems;
} DRS_MSG_DCINFOREPLY_VFFFFFFFF;
```

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

**rItems:** Information about the active LDAP connections.

#### 4.1.4.1.8 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.

#### 4.1.4.1.9 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.

#### 4.1.4.1.10 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.

#### 4.1.4.1.11 DS\_DOMAIN\_CONTROLLER\_INFO\_FFFFFFFFW

The **DS\_DOMAIN\_CONTROLLER\_INFO\_FFFFFFFFW** structure defines DC information that is returned as a part of the response to an InfoLevel = 0xFFFFFFFF request. The struct contains information about a single LDAP connection to the current server.

```
typedef struct {
 DWORD IPAddress;
 DWORD NotificationCount;
 DWORD secTimeConnected;
 DWORD Flags;
 DWORD TotalRequests;
 DWORD Reserved1;
 [string, unique] WCHAR* UserName;
} DS_DOMAIN_CONTROLLER_INFO_FFFFFFFFW;
```

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

**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.

#### 4.1.4.2 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.[<12>](#)

**Note** The server behavior of the **IDL\_DRSDomainControllerInfo** method uses the [CrackNames](#) procedure defined in section [4.1.3.2.9](#).

```
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_FFFFFFFFW
rI1: ADDRESS OF DS_DOMAIN_CONTROLLER_INFO_1W
rI2: ADDRESS OF DS_DOMAIN_CONTROLLER_INFO_2W
rI3: ADDRESS OF DS_DOMAIN_CONTROLLER_INFO_3W
found: boolean
crackMsgIn: DRS_MSG_CRACKREQ_V1
crackOut: DS_NAME_RESULTW
outV: DWORD
userAccountControl: set of integer

ValidatedDRSInput(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
 end
end

```

```

 pmsgOut^.VFFFFFFFF.rItems[i].NotificationCount :=
 lc.notificationCount
 pmsgOut^.VFFFFFFFF.rItems[i].secTimeConnected :=
 lc.secTimeConnected
 pmsgOut^.VFFFFFFFF.rItems[i].Flags := lc.flags
 pmsgOut^.VFFFFFFFF.rItems[i].TotalRequests := lc.totalRequests
 pmsgOut^.VFFFFFFFF.rItems[i].UserName := lc.userName
 pmsgOut^.VFFFFFFFF.rItems[i].Reserved1 := 0

 i := i + 1
endfor

return 0
endif

/* Verify that the given domain name matches the default domain NC.
 * First check if it is the nETBiosName or dNSHostName of the default
 * domain NC by searching for the crossRef object. 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

/* 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^.V2.cItems := number(dcSet)

i := 0
foreach dcObj in dcSet
 rI3 := ADR(pmsgOut^.V2.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 0
 dcObj!userAccountControl) # null)
 endif
 i := i + 1
endfor
endif
endif
return 0

```

#### 4.1.4.3 Examples of the IDL\_DRSDomainControllerInfo Method

An application running on a client invokes the [DRSDomainControllerInfo](#) method on DC2 to retrieve the NetBIOS and DNS host names for all DCs in the domain NC CONTOSO.COM.

##### 4.1.4.3.1 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-AB04-00C04FC2D2D2/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 DC2 [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;

#### 4.1.4.3.2 Client Request

A client invokes the [IDL\\_DRSDomainControllerInfo](#) method against DC2 with the following parameters ([DRS\\_HANDLE](#) to DC2 is omitted):

- dwInVersion = 1
- pmsgIn = [DRS\\_MSG\\_DCINFOREQ\\_V1](#)
  - Domain = "contoso.com"
  - InfoLevel = 1

#### 4.1.4.3.3 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

#### 4.1.4.3.4 Final State

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

#### 4.1.5 IDL\_DRSExecuteKCC (Opnum 18)

The **IDL\_DRSExecuteKCC** method validates the replication interconnections of DCs and updates them if necessary. This method is used only to diagnose, monitor, and manage the replication topology implementation. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperability with Windows clients.

```
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`.

#### 4.1.5.1 Method-Specific Concrete Types

##### 4.1.5.1.1 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperability with Windows clients.

```
typedef
[switch_type(DWORD)]
union {
 [case(1)]
 DRS_MSG_KCC_EXECUTE_V1 V1;
} DRS_MSG_KCC_EXECUTE;
```

**V1:** Version 1 request.

#### 4.1.5.1.2 DRS\_MSG\_KCC\_EXECUTE\_V1

The **DRS\_MSG\_KCC\_EXECUTE\_V1** structure defines the request message sent to the [IDL DRSExecuteKCC](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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 | 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 | D<br>P | A<br>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:** 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.

#### 4.1.5.2 Method-Specific Abstract Types and Procedures

##### 4.1.5.2.1 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.

#### 4.1.5.3 Server Behavior of the IDL\_DRSExecuteKCC Method

*Informative summary of behavior:* The [IDL\\_DRSExecuteKCC](#) method triggers the implemented process that generates and maintains the replication topology between DCs. [<13>](#) 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()
```

#### 4.1.6 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. This method is used only to diagnose, monitor, and manage the implementation of server-to-server DC demotion. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperation with Windows clients.

```
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`.

#### 4.1.6.1 Method-Specific Concrete Types

##### 4.1.6.1.1 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```

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.

##### 4.1.6.1.2 DRS\_MSG\_FINISH\_DEMOTIONREQ\_V1

The **DRS\_MSG\_FINISH\_DEMOTIONREQ\_V1** structure defines the request message sent to the [IDL\\_DRSFinishDemotion](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```

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 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 20 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 30 | 1 |
|---|---|---|---|---|---|---|---|---|---|----|---|---|---|---|---|---|---|---|---|----|---|---|---|---|---|---|---|---|---|----|---|
| X | X | X | U | U | D | C | R | X | X | X  | X | X | X | X | X | X | X | X | X | X  | X | X | X | F | X | X | X | X | X | X  | X |
|   |   |   | 2 | 1 |   |   |   |   |   |    |   |   |   |   |   |   |   |   |   |    |   |   |   |   |   |   |   |   |   |    |   |

**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 \(section 4.1.6.2.1\)](#).

**U1 (DS\_DEMOTE\_UNREGISTER\_SCPS, 0x00000008):** Delete any [serviceConnectionPoint](#) objects for this DC from AD DS; see [RemoveADLDSSCP \(section 4.1.6.2.2\)](#).

**U2 (DS\_DEMOTE\_UNREGISTER\_SPNS, 0x00000010):** Delete any AD LDS SPNs from the object (in the external AD DS domain) that correspond to the security principal that the AD LDS service is running as (see [RemoveADLDSPNs \(section 4.1.6.2.3\)](#)).

**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**.

#### 4.1.6.1.3 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
typedef
[switch_type(DWORD)]
union {
 [case(1)]
 DRS_MSG_FINISH_DEMOTIONREPLY_V1 V1;
} DRS_MSG_FINISH_DEMOTIONREPLY;
```

**V1:** Version 1 reply.

#### 4.1.6.1.4 DRS\_MSG\_FINISH\_DEMOTIONREPLY\_V1

The **DRS\_MSG\_FINISH\_DEMOTIONREPLY\_V1** structure defines the response message received from the [IDL DRSFinishDemotion](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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 [4.1.6.3](#)) 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 [4.1.6.3](#)) 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.

### 4.1.6.2 Method-Specific Abstract Types and Procedures

#### 4.1.6.2.1 RemoveADLDSServer

```
procedure RemoveADLDSServer(): DWORD
```

The RemoveADLDSServer procedure remotely deletes, on another DC, the **nTDSDSA object** that corresponds to this DC. This remote deletion is performed by using a server-to-server operation not described in this document. This remote delete requires at least one other DC to be present for it to succeed. If RemoveADLDSServer is invoked and no other DC is present, ERROR\_NOT\_FOUND will be returned. If no such nTDSDSA object exists on the remote DC, or if the deletion is successful, RemoveADLDSServer returns ERROR\_SUCCESS; otherwise, it returns a Win32 error. This error MAY have meaning to peer DCs and applications but is not required for interoperation with Microsoft Windows® clients.

#### 4.1.6.2.2 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.

#### 4.1.6.2.3 RemoveADLDSSPNs

```
procedure RemoveADLDSSPNs(szScriptBase: uncodestring): 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 correspond to the security principal that the AD LDS service is running as. Section [2.2.4.2](#) specifies 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.

#### 4.1.6.3 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](#)).<14>

```
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

ValidateDRSInput(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 protocols specified in section 4.5 of MS-ADSO.

 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 := RemoveADLDSserver()
 if ret = ERROR_SUCCESS then
 msgOut.dwOperationDone :=
 msgOut.dwOperationDone + {DS_DEMOTE_DELETE_CSMETA}
 else
 msgOut.dwOpFailed =
 msgOut.dwOpFailed + {DS_DEMOTE_DELETE_CSMETA}
 if msgOut.dwOpError = ERROR_SUCCESS then
 msgOut.dwOpError := ret
 endif
 endif
 endif
endif

```

```

/* 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 := RemoveADLDSSCP()
if ret = ERROR_SUCCESS then
msgOut.dwOperationDone :=
msgOut.dwOperationDone + {DS_DEMOTE_UNREGISTER_SCPS}
else
msgOut.dwOpFailed =
msgOut.dwOpFailed + {DS_DEMOTE_UNREGISTER_SCPS}
if msgOut.dwOpError = ERROR_SUCCESS then
msgOut.dwOpError := ret
endif
endif
endif
/* 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 := RemoveADLDSSPNs(msgIn.szScriptBase)
if res = TRUE then
msgOut.dwOperationDone :=
msgOut.dwOperationDone + {DS_DEMOTE_UNREGISTER_SPNS}
else
msgOut.dwOpFailed =
msgOut.dwOpFailed + {DS_DEMOTE_UNREGISTER_SPNS}
endif
endif
/* End operations corresponding to dwOperations value of DS_DEMOTE_UNREGISTER_SPNS */
endif
endif
pmsgOut^ := msgOut
pdwMsgOut^ := 1
return ERROR_SUCCESS

```

#### 4.1.7 IDL\_DRSGetReplInfo (Opnum 19)

The **IDL\_DRSGetReplInfo** method retrieves the replication state of the server. This method is used only to diagnose, monitor, and manage the replication implementation. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperation with Windows clients.

```

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`.

#### 4.1.7.1 Method-Specific Concrete Types

##### 4.1.7.1.1 DRS\_MSG\_GETREPLINFO\_REQ

The **DRS\_MSG\_GETREPLINFO\_REQ** union defines the request message versions sent to the [IDL DRSGetReplInfo](#) method.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

##### 4.1.7.1.2 DRS\_MSG\_GETREPLINFO\_REQ\_V1

The **DRS\_MSG\_GETREPLINFO\_REQ\_V1** structure defines a version 1 request message sent to the [IDL DRSGetReplInfo](#) method.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.3 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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 | 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 | M<br>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.

**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](#).

#### 4.1.7.1.4 DS\_REPL\_INFO Codes

DS\_REPL\_INFO codes indicate the type of replication state information being requested.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Microsoft Windows® clients.

| Value                                                | Meaning                                                                                                                               |
|------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| DS_REPL_INFO_NEIGHBORS<br>0x00000000                 | Replication state data for each NC and source server pair, for all NC replicas hosted by this DC.                                     |
| DS_REPL_INFO_CURSORS_FOR_NC<br>0x00000001            | A portion of the replication state for the NC replica of a given NC.                                                                  |
| DS_REPL_INFO_METADATA_FOR_OBJ<br>0x00000002          | Stamps for all the replicated attributes of the given object.                                                                         |
| DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES<br>0x00000003  | Replication state data regarding connection failures with inbound replication partners.                                               |
| DS_REPL_INFO_KCC_DSA_LINK_FAILURES<br>0x00000004     | Replication state data regarding link failures with inbound replication partners.                                                     |
| DS_REPL_INFO_PENDING_OPS<br>0x00000005               | Replication tasks that are currently executing or that are queued to execute.                                                         |
| DS_REPL_INFO_METADATA_FOR_ATTR_VALUE<br>0x00000006   | Stamps for a specific <b>link attribute</b> of the given object.                                                                      |
| DS_REPL_INFO_CURSORS_2_FOR_NC<br>0x00000007          | A portion of the replication state for the NC replica of a given NC.                                                                  |
| DS_REPL_INFO_CURSORS_3_FOR_NC<br>0x00000008          | A portion of the replication state for the NC replica of a given NC.                                                                  |
| DS_REPL_INFO_METADATA_2_FOR_OBJ<br>0x00000009        | Stamps for all the replicated attributes of the given object.                                                                         |
| DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE<br>0x0000000A | Stamps for a specific link attribute of the given object.                                                                             |
| DS_REPL_INFO_SERVER_OUTGOING_CALLS<br>0xFFFFFFFFA    | A list of all outstanding RPC server call contexts.                                                                                   |
| DS_REPL_INFO_UPTODATE_VECTOR_V1<br>0xFFFFFFFFB       | The replication state for the NC replica of a given NC.                                                                               |
| DS_REPL_INFO_CLIENT_CONTEXTS<br>0xFFFFFFFFC          | A list of all outstanding RPC client contexts.                                                                                        |
| DS_REPL_INFO_REPSTO<br>0xFFFFFFFFE                   | Replication state data for each NC and destination server (which is notified of changes) pair, for all NC replicas hosted by this DC. |

#### 4.1.7.1.5 DRS\_MSG\_GETREPLINFO\_REPLY

The **DRS\_MSG\_GETREPLINFO\_REPLY** union defines response messages received from the [IDL DRSGetReplInfo](#) method.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperability with Windows clients.

```

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.

#### 4.1.7.1.6 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
typedef struct {
 DWORD cNumNeighbors;
 DWORD dwReserved;
 [size_is(cNumNeighbors)] DS_REPL_NEIGHBORW rgNeighbor[];
} DS_REPL_NEIGHBORSW;
```

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

**dwReserved:** Unused. MUST be 0 and ignored.

**rgNeighbor:** A set of replication neighbors.

#### 4.1.7.1.7 DS\_REPL\_NEIGHBORW

The **DS\_REPL\_NEIGHBORW** structure defines a replication neighbor. This structure is a concrete representation of a [RepsFrom](#) or [RepsTo](#) value.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.8 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.9 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.10 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.11 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.12 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```

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.

#### 4.1.7.1.13 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```

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**.

#### 4.1.7.1.14 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperability with Windows clients.

```
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.

#### 4.1.7.1.15 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperability with Windows clients.

```
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.

#### 4.1.7.1.16 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.17 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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`.

#### 4.1.7.1.18 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.19 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.20 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.21 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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 \(section 5.31\)](#).

**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.

#### 4.1.7.1.22 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.23 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.24 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperability with Windows clients.

```

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.

#### 4.1.7.1.25 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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**.

#### 4.1.7.1.26 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.27 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
typedef struct {
 ULONGLONG hCtx;
 LONG lReferenceCount;
 BOOL fIsBound;
 UUID uuidClient;
 DSTIME timeLastUsed;
 ULONG IPAddr;
 int pid;
} DS_REPL_CLIENT_CONTEXT;
```

**hCtx:** The unique ID of the client context.

**lReferenceCount:** The number of references to the context.

**fIsBound:** True if and only if the context has not yet been closed by the [IDL DRSUnbind](#) method.

**uuidClient:** Zeros, or the value pointed to by the *puuidClientDsa* parameter to [IDL DRSBind](#).

**timeLastUsed:** The date and 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**.

#### 4.1.7.1.28 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](#).

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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.

#### 4.1.7.1.29 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.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

```
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 an implemented cache.

**fIsHandleInCache:** True if and only if the context handle is still in the implemented cache.

**dwThreadId:** The implementation-specific 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     | <a href="#">IDL_DRSBind</a>                             |
| 3     | <a href="#">IDL_DRSUnbind</a>                           |
| 4     | <a href="#">IDL_DRSReplicaSync</a>                      |
| 5     | Implemented server-to-server replication method         |
| 6     | <a href="#">IDL_DRSUpdateRefs</a>                       |
| 7     | <a href="#">IDL_DRSReplicaAdd</a>                       |
| 8     | <a href="#">IDL_DRSReplicaDel</a>                       |
| 9     | Implemented server-to-server object verification method |
| 10    | Implemented server-to-server group expansion method     |
| 11    | Implemented server-to-server cross NC move method       |
| 12    | Implemented server-to-server replication method         |
| 13    | <a href="#">IDL_DRSCrackNames</a>                       |
| 14    | Implemented server-to-server object creation method     |
| 15    | Implemented server-to-server object lookup method       |
| 16    | Implemented server-to-server replication method         |
| 17    | <a href="#">IDL_DRSGetReplInfo</a>                      |
| 18    | <a href="#">IDL_DRSWriteSPN</a>                         |

#### 4.1.7.2 Method-Specific Abstract Types and Procedures

##### 4.1.7.2.1 GetDNFromInvocationID

```
procedure GetDNFromInvocationID(invocationID: GUID): DN
```

Returns the DN of the [nTDSDSA](#) object that has the specified invocation ID.

##### 4.1.7.2.2 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
```

#### 4.1.7.2.3 GetNCs

```
procedure GetNCs(): set of DSName
```

Returns a set containing the [DSNames](#) of all NCs hosted by this server.

#### 4.1.7.2.4 GetUpToDateenessVector

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

Returns a sequence of [ReplUpToDateVector \(section 5.111\)](#), 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.

#### 4.1.7.3 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 [4.1.7.1.4](#).

```
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_NEIGHBOR
```

```

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

```

```

ValidateDRSInput(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
 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
 endif
 endfor
 endforeach
 endif

```

```

 i := i + 1
 endif
endfor
endfor
pmsgOut^.pNeighbors^.cNumNeighbors := j
else
 /* DS_REPL_INFO_REPSTO case. */
 i := 0
 j := 0
 foreach nc in ncs
 foreach q 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^.pCursor3^.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
 endif
 endwhile
 endif
endif

```

```

pAttrValueMetaData^.ftimeDeleted := ls.timeDeleted
pAttrValueMetaData^.dwVersion := ls.dwVersion
pAttrValueMetaData^.ftimeLastOriginatingChange :=
 ls.timeChanged
pAttrValueMetaData^.uuidLastOriginatingDsaInvocationID :=
 ls.uuidOriginating
pAttrValueMetaData^.usnOriginatingChange := ls.usnOriginating
pAttrValueMetaData^.usnLocalChange :=
 implementation-specific value maintained for each link
 attribute value
else
 pAttrValueMetaData2 :=
 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
 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

```

#### 4.1.7.4 Examples of the IDL\_DRSGetReplInfo Method

##### 4.1.7.4.1 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 this by issuing a request to DC1 with **pszObjectDN** set to the DN of the domain NC.

###### 4.1.7.4.1.1 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);

#### 4.1.7.4.1.2 Client Request

The client invokes the [IDL DRSGetReplInfo](#) 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

#### 4.1.7.4.1.3 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

#### 4.1.7.4.1.4 Final State

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

#### 4.1.7.4.2 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 wants to see which NCs DCA1 receives replication updates for from DC1. The domain administrator does this by issuing a request to DCA1 with **pszObjectDN** set to null and **uuidSourceDsaObjGuid** set to the DSA GUID of DC1.

##### 4.1.7.4.2.1 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);

#### 4.1.7.4.2.2 Client Request

The client invokes the [IDL DRSGetReplInfo](#) 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

#### 4.1.7.4.2.3 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
- uidSourceDsaInvocationID = 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[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
  - uidSourceDsaInvocationID = 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

#### 4.1.7.4.2.4 Final State

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

#### 4.1.7.4.3 Calling IDL\_DRSGetReplInfo with infoType DS\_REPL\_INFO\_REPSTO to find replication neighbors for a specified NC

In this example, the domain administrator verifies to which DCs DC1 sends replication updates 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.

##### 4.1.7.4.3.1 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);`

##### 4.1.7.4.3.2 Client Request

The client invokes the [IDL\\_DRSGetReplInfo](#) 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`

##### 4.1.7.4.3.3 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
      - uuidSourceDsaInvocationID = 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

#### 4.1.7.4.3.4 Final State

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

#### 4.1.7.4.4 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.

##### 4.1.7.4.4.1 Initial State

Querying the NC root object for 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};

##### 4.1.7.4.4.2 Client Request

The client invokes the [IDL\\_DRSGetReplInfo](#) 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

##### 4.1.7.4.4.3 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

#### 4.1.7.4.4.4 Final State

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

#### 4.1.7.4.5 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.

##### 4.1.7.4.5.1 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;`

#### 4.1.7.4.5.2 Client Request

The client invokes the [IDL\\_DRSGetReplInfo](#) 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`

#### 4.1.7.4.5.3 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

#### 4.1.7.4.5.4 Final State

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

#### 4.1.7.4.6 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.

##### 4.1.7.4.6.1 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;

##### 4.1.7.4.6.2 Client Request

The client invokes the [IDL\\_DRSGetReplInfo](#) 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

#### 4.1.7.4.6.3 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

#### 4.1.7.4.6.4 Final State

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

#### 4.1.7.4.7 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.

##### 4.1.7.4.7.1 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;

##### 4.1.7.4.7.2 Client Request

The client invokes the [IDL\\_DRSGetReplInfo](#) 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

#### 4.1.7.4.7.3 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

#### 4.1.7.4.7.4 Final State

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

#### 4.1.7.4.8 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.

##### 4.1.7.4.8.1 Initial State

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

##### 4.1.7.4.8.2 Client Request

The client invokes the [IDL\\_DRSGetReplInfo](#) 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

##### 4.1.7.4.8.3 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

#### 4.1.7.4.8.4 Final State

The final state is the same as the [initial state \(section 4.1.7.4.8.1\)](#); there is no change.

#### 4.1.7.4.9 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.

##### 4.1.7.4.9.1 Initial State

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

##### 4.1.7.4.9.2 Client Request

The client invokes the [IDL\\_DRSGetReplInfo \(section 4.1.7\)](#) 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

##### 4.1.7.4.9.3 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

#### 4.1.7.4.9.4 Final State

The final state is the same as the [initial state \(section 4.1.7.4.9.1\)](#); there is no change.

### 4.1.8 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. This method is used only to diagnose, monitor, and manage the implementation of server-to-server DC demotion. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperation with Windows clients.

```
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`.

#### 4.1.8.1 Method-Specific Concrete Types

##### 4.1.8.1.1 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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.

##### 4.1.8.1.2 DRS\_MSG\_INIT\_DEMOTIONREQ\_V1

The **DRS\_MSG\_INIT\_DEMOTIONREQ\_V1** structure defines a request message sent to the [IDL `DRSInitDemotion`](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
typedef struct {
 DWORD dwReserved;
} DRS_MSG_INIT_DEMOTIONREQ_V1;
```

**dwReserved:** Unused. MUST be 0.

##### 4.1.8.1.3 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
typedef
[switch_type(DWORD)]
union {
```

```

 [case(1)]
 DRS_MSG_INIT_DEMOTIONREPLY_V1 V1;
 } DRS_MSG_INIT_DEMOTIONREPLY;

```

**V1:** Version 1 reply.

#### 4.1.8.1.4 DRS\_MSG\_INIT\_DEMOTIONREPLY\_V1

The **DRS\_MSG\_INIT\_DEMOTIONREPLY\_V1** structure defines a response message received from the [IDL DRSInitDemotion](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```

typedef struct {
 DWORD dwOpError;
} DRS_MSG_INIT_DEMOTIONREPLY_V1;

```

**dwOpError:** A Win32 error code, as specified in [\[MS-ERREF\]](#) section 2.2.

#### 4.1.8.2 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. [<15>](#)

```

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

ValidateDRSInput(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

pmsgOut^.V1.dwOpError := ERROR_SUCCESS
pdwOutVersion^ := 1
return ERROR_SUCCESS

```

### 4.1.9 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`.

#### 4.1.9.1 Method-Specific Concrete Types

##### 4.1.9.1.1 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.

#### 4.1.9.1.2 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.

#### 4.1.9.1.3 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.

#### 4.1.9.1.4 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 **rgszToSites** field in the request message.

**dwFlags:** Unused. MUST be 0 and ignored.

#### 4.1.9.1.5 DRS\_MSG\_QUERY\_SITES\_REPLY\_ELEMENT\_V1

The **DRS\_MSG\_QUERY\_SITES\_REPLY\_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.

### 4.1.9.2 Method-Specific Abstract Types and Procedures

#### 4.1.9.2.1 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
```

#### 4.1.9.2.2 WeightedArc and WeightedArcSet

```
type WeightedArc = [initial: DSName, final: DSName, cost: integer]
type WeightedArcSet = set of WeightedArc
```

The cost field of a WeightedArc is positive.

#### 4.1.9.2.3 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.

#### 4.1.9.3 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*. The server-to-server replication implementation 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.

```

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

ValidatedDRSInput(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.rgToSites = null then
 return ERROR_INVALID_PARAMETER
endif
for i := 0 to msgIn.cToSites - 1
 ul := ValidateSiteRDN(msgIn.rgToSites[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 7.1.1.2.3.1, "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
 endfor
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
endif
endfor

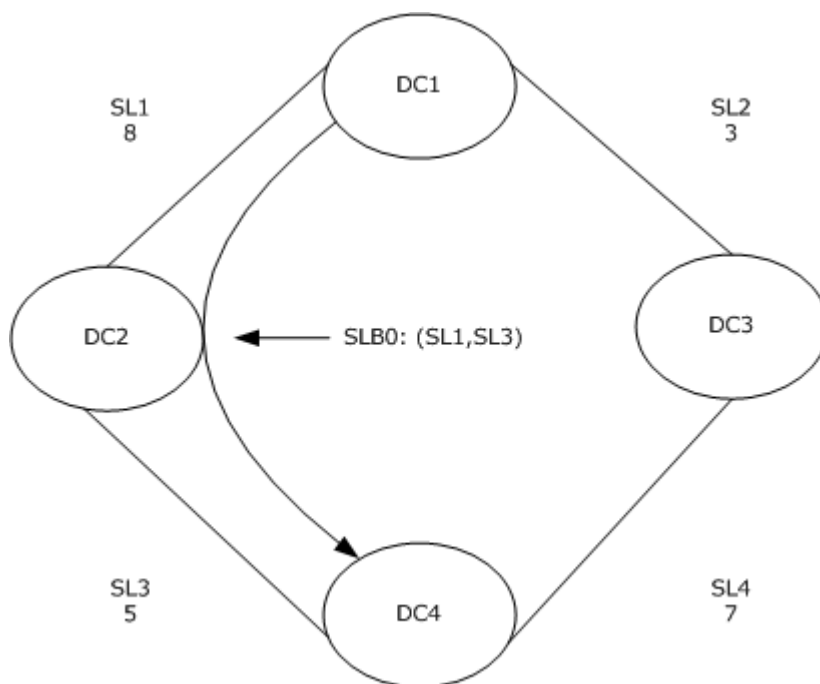
return 0

```

#### 4.1.9.4 Examples of IDL\_DRSQuerySitesByCost Method

##### 4.1.9.4.1 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 | 8          |

| VERTEX | ARC                           | ARC WEIGHT |
|--------|-------------------------------|------------|
|        | SL2<br>SLB0 (SL1, SL3)        | 3<br>13    |
| DC2    | SL1<br>SL3                    | 8<br>5     |
| DC3    | SL2<br>SL4                    | 3<br>7     |
| DC4    | SL3<br>SL4<br>SLB0 (SL1, SL3) | 7<br>13    |

#### 4.1.9.4.1.1 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(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=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 DNs:
- 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;

#### 4.1.9.4.1.2 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

#### 4.1.9.4.1.3 Server Response

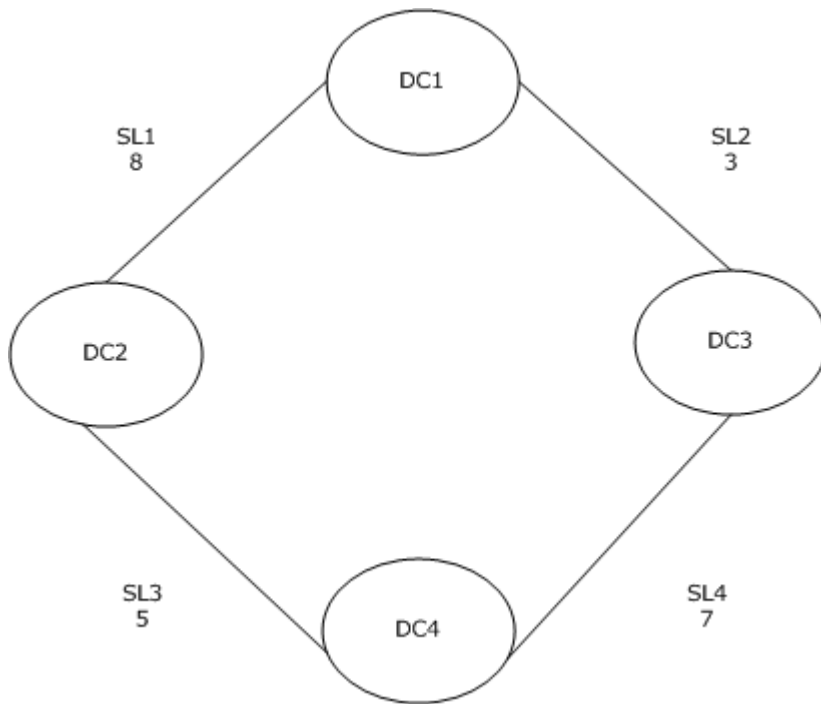
- pdwOutVersion^ = 1
- pmsgOut = DRS\_MSG\_QUERY\_SITESREPLY\_V1
  - cToSites = 3
  - rgCostInfo[0]: DRS\_MSG\_QUERY\_SITESREPLYELEMENT\_V1
    - dwErrorCode = 0
    - dwCost: = 8
  - rgCostInfo[1]: DRS\_MSG\_QUERY\_SITESREPLYELEMENT\_V
    - dwErrorCode = 0
    - dwCost: = 3
  - rgCostInfo[2]: DRS\_MSG\_QUERY\_SITESREPLYELEMENT\_V1
    - dwErrorCode = 0
    - dwCost: = 13
- dwFlags = 0

#### 4.1.9.4.1.4 Final State

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

#### 4.1.9.4.2 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 | 8          |
|        | SL2 | 3          |
| DC2    | SL1 | 8          |
|        | SL3 | 5          |
| DC3    | SL2 | 3          |
|        | SL4 | 7          |
| DC4    | SL3 | 5          |
|        | SL4 | 7          |

##### 4.1.9.4.2.1 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(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
  - > 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 DN's;
- 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;

#### 4.1.9.4.2.2 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
- rgpszToSites = {"DC2", "DC3", "DC4"}
- dwFlags = 0

#### 4.1.9.4.2.3 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

#### 4.1.9.4.2.4 Final State

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

### 4.1.10 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`.

#### 4.1.10.1 Method-Specific Concrete Types

##### 4.1.10.1.1 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.

##### 4.1.10.1.2 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.

##### 4.1.10.1.3 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.

#### 4.1.10.1.4 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.

### 4.1.10.2 Method-Specific Abstract Types and Procedures

#### 4.1.10.2.1 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.

#### 4.1.10.3 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.

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

```

#### 4.1.11 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`.

##### 4.1.11.1 Method-Specific Concrete Types

###### 4.1.11.1.1 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.

#### 4.1.11.1.2 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.)

#### 4.1.11.1.3 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.

#### 4.1.11.1.4 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.

#### 4.1.11.2 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 replication SPNs associated with the DC's [computer](#) object. This method is typically used 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). 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 the replication SPNs from the computer object. */

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

```

#### 4.1.12 IDL\_DRSReplicaAdd (Opnum 5)

The **IDL\_DRSReplicaAdd** method adds a replication source reference for the specified NC. This method is used only to diagnose, monitor, and manage the replication implementation. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperability with Windows clients.

```

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`.

##### 4.1.12.1 Method-Specific Concrete Types

###### 4.1.12.1.1 DRS\_MSG\_REPADD

The **DRS\_MSG\_REPADD** union defines request messages that are sent to the [IDL\\_DRSReplicaAdd](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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).

#### 4.1.12.1.2 DRS\_MSG\_REPADD\_V1

The **DRS\_MSG\_REPADD\_V1** structure defines a request message sent to the [IDL DRSReplicaAdd](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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.

#### 4.1.12.1.3 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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 to be used in the server-to-server replication implementation with the specified DC.

**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.

#### 4.1.12.2 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 at the server. 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 replication by sending a server-to-server replication 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

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
 /* 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
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
 Perform a server-to-server call to enable replication notifications
 by requesting the server DC specified by msgIn.pszSourceDsaAddress^
 to add a repsTo for this DC.

 endif

err := PerformReplication(nc, msgIn.pSourceDsaDN)

rf.timeLastAttempt := current time
rf.resultLastAttempt := err
/* Update the repsFrom. */
if err==ERROR_SUCCESS then
 rf.timeLastSuccess := current time
 rf.consecutiveFailures = 0
else
 rf.consecutiveFailures := rf.consecutiveFailures + 1
 rf.timeLastSuccess := 0
endif
endif

```

```
return err
```

#### 4.1.13 IDL\_DRSReplicaDel (Opnum 6)

The **IDL\_DRSReplicaDel** method deletes a replication source reference for the specified NC. This method is used only to diagnose, monitor, and manage the replication implementation. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperation with Windows clients.

```
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.

##### 4.1.13.1 Method-Specific Concrete Types

###### 4.1.13.1.1 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
typedef
[switch_type(DWORD)]
union {
 [case(1)]
 DRS_MSG_REPDEL_V1 V1;
} DRS_MSG_REPDEL;
```

**V1:** The version 1 request.

###### 4.1.13.1.2 DRS\_MSG\_REPDEL\_V1

The **DRS\_MSG\_REPDEL\_V1** structure defines a request message sent to the [IDL\\_DRSReplicaDel](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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.

#### 4.1.13.2 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
rf: RepsFrom
msgIn: DRS_MSG_REPDEL_V1
rt: ULONG

ValidateDRSInput(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 stil 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
 rt := RemoveObj(nc, false)
 if rt ≠ 0 then
 return rt
 endif
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
 Perform a server-to-server call to disable replication notifications
 by requesting the server DC specified by msgIn.pszDsaSrc to remove
 this DC from its repsTo.

```

```

endif
return 0
endif

```

#### 4.1.14 IDL\_DRSReplicaDemotion (Opnum 26)

The **IDL\_DRSReplicaDemotion** method initiates server-to-server replication to replicate off all changes to the specified NC and moves any FSMOs held to another server. This method is used only to diagnose, monitor, and manage the replication and FSMO implementation related to DC demotion. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperability with Windows clients.

```

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`.

##### 4.1.14.1 Method-Specific Concrete Types

###### 4.1.14.1.1 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperability with Windows clients.

```

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.

#### 4.1.14.1.2 DRS\_MSG\_REPLICA\_DEMOTIONREQ\_V1

The **DRS\_MSG\_REPLICA\_DEMOTIONREQ\_V1** structure defines a request message sent to the [IDL DRSReplicaDemotion](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```

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.

#### 4.1.14.1.3 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```

typedef
[switch_type(DWORD)]
union {
 [case(1)]
 DRS_MSG_REPLICA_DEMOTIONREPLY_V1 V1;
} DRS_MSG_REPLICA_DEMOTIONREPLY;

```

**V1:** The version 1 reply.

#### 4.1.14.1.4 DRS\_MSG\_REPLICA\_DEMOTIONREPLY\_V1

The **DRS\_MSG\_REPLICA\_DEMOTIONREPLY\_V1** structure defines a response message received from the [IDL DRSReplicaDemotion](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
typedef struct {
 DWORD dwOpError;
} DRS_MSG_REPLICA_DEMOTIONREPLY_V1;
```

**dwOpError:** The Win32 error code, as specified in [\[MS-ERREF\]](#) section 2.2.

#### 4.1.14.2 Method-Specific Abstract Types and Procedures

##### 4.1.14.2.1 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.

##### 4.1.14.2.2 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

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
 /* Let's try to transfer the role. This operation is not
 described in this document. */

 ret = Perform a server-to-server call
 to abandon the held FSMO role to targetDSA[i]
 if ret = ERROR_SUCCESS then
 /* successfully given away */
 bGivenAway := true
 endif

 i := i + 1
endwhile
/* if execution got here, the role was given away */
return ERROR_SUCCESS

```

#### 4.1.14.2.3 ReplicateOffChanges()

```

procedure ReplicateOffChanges(nc: DSNAME): DWORD

```

The ReplicateOffChanges procedure replicates all local changes in the NC to a randomly selected replication partner. If it finds a replication partner hosting this NC and can successfully replicate to it, the method returns ERROR\_SUCCESS; otherwise, it returns a Win32 error. The server-to-server replication implementation that performs this operation is not included in this document.

#### 4.1.14.3 Server Behavior of the IDL\_DRSReplicaDemotion Method

*Informative summary of behavior:* For a given NC, the [IDL\\_DRSReplicaDemotion](#) method initiates server-to-server replication to replicate 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

ValidateDRSInput(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

```

#### 4.1.15 IDL\_DRSReplicaModify (Opnum 7)

The **IDL\_DRSReplicaModify** method updates the value for [repsFrom](#) for the NC replica. This method is used only to diagnose, monitor, and manage the replication implementation. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperation with Windows clients.

```

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`.

#### 4.1.15.1 Method-Specific Concrete Types

##### 4.1.15.1.1 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
typedef
[switch_type(DWORD)]
union {
 [case(1)]
 DRS_MSG_REPMOD_V1 V1;
} DRS_MSG_REPMOD;
```

**V1:** The version 1 request.

##### 4.1.15.1.2 DRS\_MSG\_REPMOD\_V1

The **DRS\_MSG\_REPMOD\_V1** structure defines a request message for the [IDL DRSReplicaModify](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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, displayed 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 | X | X | U<br>S | U<br>A | U<br>F | X | X | X  | X | X | X | X | X | X | X | X | X | X  | X | X | X | X | X | X | X | X | X | X  | X |

**X:** 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.

#### 4.1.15.2 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

ValidatedDRSInput(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

```

#### 4.1.16 IDL\_DRSReplicaSync (Opnum 2)

The **IDL\_DRSReplicaSync** method triggers replication from another DC. This method is used only to diagnose, monitor, and manage the replication implementation. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperation with Windows clients.

```
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`.

##### 4.1.16.1 Method-Specific Concrete Types

###### 4.1.16.1.1 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
typedef
[switch_type(DWORD)]
union {
 [case(1)]
 DRS_MSG_REPSYNC_V1 V1;
} DRS_MSG_REPSYNC;
```

**V1:** The version 1 request.

###### 4.1.16.1.2 DRS\_MSG\_REPSYNC\_V1

The **DRS\_MSG\_REPSYNC\_V1** structure defines a request message sent to the [IDL\\_DRSReplicaSync](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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.

#### 4.1.16.2 Server Behavior of the IDL\_DRSReplicaSync Method

*Informative summary of behavior:* The server begins replication by sending a server-to-server replication 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

ValidateDRSInput(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
 err := PerformReplication(nc, r)

 rf.timeLastAttempt := current time
 rf.resultLastAttempt := err
 /* Update the repsFrom. */
 if err==ERROR_SUCCESS then
 rf.timeLastSuccess := current time
 rf.consecutiveFailures := 0
 else
 rf.consecutiveFailures := rf.consecutiveFailures + 1
 rf.timeLastSuccess := 0
 endif
endfor

return 0

```

#### 4.1.17 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. This method is used only to diagnose, monitor, and manage the replication implementation. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperability with Windows clients.

```

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`.

#### 4.1.17.1 Method-Specific Concrete Types

##### 4.1.17.1.1 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
typedef
[switch_type(DWORD)]
union {
 [case(1)]
 DRS_MSG_REPVERIFYOBJ_V1 V1;
} DRS_MSG_REPVERIFYOBJ;
```

**V1:** The version 1 request.

##### 4.1.17.1.2 DRS\_MSG\_REPVERIFYOBJ\_V1

The **DRS\_MSG\_REPVERIFYOBJ\_V1** structure defines a request message sent to the [IDL\\_DRSReplicaVerifyObjects](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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.

#### 4.1.17.2 Method-Specific Abstract Types and Procedures

##### 4.1.17.2.1 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. The server-to-server replication implementation that performs this operation is not included in this document.

##### 4.1.17.2.2 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.

##### 4.1.17.3 Server Behavior of the IDL\_DRSReplicaVerifyObjects Method

*Informative summary of behavior:* Let *N* be the NC  $pNC^{\wedge}$ , 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

ValidatedDRSInput(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
```

#### 4.1.17.4 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.

##### 4.1.17.4.1 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 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.                                  |

| Users at DC1 | Users at DC2                                                                                                                               | Notes                                      |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
|              | <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 | uuidLastOriginatingDsaInvocationID   | 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 | uuidLastOriginatingDsaInvocationID   | 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:

| usnLocalChange | uuidLastOriginatingDsaInvocationID   | usnOriginatingChange | ftimeLastOriginatingChange | dwVersion | pszAttributeNames |
|----------------|--------------------------------------|----------------------|----------------------------|-----------|-------------------|
| 9071           | 4875e25f-11a9-4c70-abf4-5fb39529f84b | 9071                 | 1/15/2009 11:05:42         | 1         | whenCreated       |

Relevant metadata entries for "CN=Jim Daly,CN=Users,DC=CONTOSO,DC=COM" at DC1:

| usnLocalChange | uuidLastOriginatingDsaInvocationID   | usnOriginatingChange | ftimeLastOriginatingChange | dwVersion | pszAttributeNames |
|----------------|--------------------------------------|----------------------|----------------------------|-----------|-------------------|
| 14085          | 4875e25f-11a9-4c70-abf4-5fb39529f84b | 14085                | 5/21/2010 19:06:32         | 1         | whenCreated       |
| 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:

| usnLocalChange | uuidLastOriginatingDsaInvocationID   | usnOriginatingChange | ftimeLastOriginatingChange | dwVersion | pszAttributeNames |
|----------------|--------------------------------------|----------------------|----------------------------|-----------|-------------------|
| 12324          | 4875e25f-11a9-4c70-abf4-5fb39529f84b | 13964                | 5/21/2010 18:08:30         | 1         | whenCreated       |

Relevant metadata entries for "CN=Jim Daly,CN=Users,DC=CONTOSO,DC=COM" at DC2:

| usnLocalC<br>hange | uuidLastOriginatingDsa<br>InvocationID   | usnOriginatin<br>gChange | ftimeLastOriginati<br>ngChange | dwVer<br>sion | pszAttribut<br>eName |
|--------------------|------------------------------------------|--------------------------|--------------------------------|---------------|----------------------|
| 12432              | 4875e25f-11a9-4c70-<br>abf4-5fb39529f84b | 14085                    | 5/21/2010<br>19:06:32          | 1             | whenCreate<br>d      |

Relevant metadata entries for "CN=Ebru Ersan,CN=Users,DC=CONTOSO,DC=COM" at DC2:

| usnLocalC<br>hange | uuidLastOriginatingDsa<br>InvocationID   | usnOriginatin<br>gChange | ftimeLastOriginati<br>ngChange | dwVer<br>sion | pszAttribut<br>eName |
|--------------------|------------------------------------------|--------------------------|--------------------------------|---------------|----------------------|
| 12451              | 7526f625-51db-4022-<br>8150-59c0286efd82 | 12451                    | 5/21/2010<br>19:19:14          | 1             | whenCreate<br>d      |

Relevant metadata entries for "CN=Kari Furse,CN=Users,DC=CONTOSO,DC=COM" at DC2:

| usnLocalC<br>hange | uuidLastOriginatingDsa<br>InvocationID   | usnOriginatin<br>gChange | ftimeLastOriginati<br>ngChange | dwVer<br>sion | pszAttribut<br>eName |
|--------------------|------------------------------------------|--------------------------|--------------------------------|---------------|----------------------|
| 441                | 4875e25f-11a9-4c70-<br>abf4-5fb39529f84b | 5099                     | 11/1/2008<br>04:29:47          | 1             | whenCreate<br>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;

#### 4.1.17.4.2 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

#### 4.1.17.4.3 Server Response

The server returns a code of 0.

#### 4.1.17.4.4 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,<br>CN=Users,DC=CONTOSO,DC=COM | <GUID=f5ef2f4b-a3db-464c-8403-b27aa00b0d5d>;<SID=S-1-5-21-1583212203-607051668-819563750-1107>;CN=Kim Akers,<br>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 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.                                                               |

#### 4.1.18 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.

#### 4.1.18.1 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
```

#### 4.1.19 IDL\_DRSUpdateRefs (Opnum 4)

The **IDL\_DRSUpdateRefs** method adds or deletes a value from the [repsTo](#) of a specified NC replica. This method is used only to diagnose, monitor, and manage the replication implementation. The structures requested and returned through this method MAY have meaning to peer DCs and applications but are not required for interoperability with Windows clients.

```
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.

### 4.1.19.1 Method-Specific Concrete Types

#### 4.1.19.1.1 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.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
typedef
[switch_type(DWORD)]
union {
 [case(1)]
 DRS_MSG_UPDREFS_V1 V1;
} DRS_MSG_UPDREFS;
```

**V1:** The version 1 request.

#### 4.1.19.1.2 DRS\_MSG\_UPDREFS\_V1

The **DRS\_MSG\_UPDREFS\_V1** structure defines a request message sent to the [IDL\\_DRSUpdateRefs](#) method.

This structure MAY have meaning to peer DCs and applications, but it is not required for interoperation with Windows clients.

```
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.

### 4.1.19.2 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
rt: RepsTo

ValidatedDRSInput(hDrs, 4)

if dwInVersion ≠ 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 ∩ {DRS_ADD_REF, DRS_DEL_REF} = null)
 return ERROR_DS_DRA_INVALID_PARAMETER
endif
nc := msgIn.pNC^

if ((IT_WRITE in nc!instanceType) ≠ (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

if DRS_ASYNC_OP in options then
 Asynchronous Processing: Initiate a logical thread of control
 to process the remainder of this request asynchronously
 return 0
endif

if DRS_DEL_REF in options then
 rt := select one v from nc!repsTo where
 (v.naDsa = msgIn.pszDsaDest or
 v.uuidDsa = msgIn.uuidDsaObjDest)
 if rt = null then
 err := ERROR_DS_DRA_REF_NOT_FOUND
 else
 nc!repsTo := nc!repsTo - {rt}
 err := 0
 endif
endif

/* If DRS_DEL_REF and DRS_ADD_REF are both specified, the return
 * value is that associated with the DRS_ADD_REF. */
if DRS_ADD_REF in options then
 rt := select one v from nc!repsTo where
 (v.naDsa = msgIn.pszDsaDest or

```

```

 v.uuidDsa = msgIn.uuidDsaObjDest)
if rt = null then
 rt.naDsa := msgIn.pszDsaDest
 rt.uuidDsa := msgIn.uuidDsaObjDest
 rt.options := options ∩ {DRS_WRIT_REP}
 rt.timeLastAttempt := 0
 rt.timeLastSuccess := current time
 rt.consecutiveFailures := 0
 rt.resultLastAttempt := 0
 nc!repsTo := nc!repsTo + {rt}
 err := 0
else
 err := ERROR_DS_DRA_REF_ALREADY_EXISTS
endif
endif
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)
 err := 0
return err

```

### 4.1.19.3 Examples of the IDL\_DRSUpdateRefs Method

#### 4.1.19.3.1 Adding a repsTo Entry

This example shows how to add a new **repsTo** entry by calling [IDL\\_DRSUpdateRefs \(section 4.1.19\)](#) with the *DRS\_ADD\_REF* parameter.

##### 4.1.19.3.1.1 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
```

##### 4.1.19.3.1.2 Client Request

A client 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

#### 4.1.19.3.1.3 Server Response

Return code of 0.

#### 4.1.19.3.1.4 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);

#### 4.1.19.3.2 Replacing a repsTo Entry

This example shows how to semantically update an existing **repsTo** entry by calling [IDL DRSUpdateRefs \(section 4.1.19\)](#) with the *DRS\_ADD\_REF* and *DRS\_DEL\_REF* parameters.

##### 4.1.19.3.2.1 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: 12924315918, 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);

##### 4.1.19.3.2.2 Client Request

A client 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: \_GUID { 5fe84f18-3765-4ca3-b895-47802a7ab74f }
- ulOptions: DRS\_WRIT\_REP | DRS\_DEL\_REF | DRS\_ADD\_REF

#### 4.1.19.3.2.3 Server Response

Return code of 0.

#### 4.1.19.3.2.4 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.pszDSIInstance 5fe84f18-3765-4ca3-b895-47802a7ab74f.\_msdcs.CONTOSO.COM
- v2.pguidDSIInstance (null);

#### 4.1.20 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`.

#### 4.1.20.1 Method-Specific Concrete Types

##### 4.1.20.1.1 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.

##### 4.1.20.1.2 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.

#### 4.1.20.1.3 DRS\_MSG\_SPNREPLY

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

```
typedef
[switch_type(DWORD)]
union {
 [case(1)]
 DRS_MSG_SPNREPLY_V1 V1;
} DRS_MSG_SPNREPLY;
```

**V1:** The version 1 response.

#### 4.1.20.1.4 DRS\_MSG\_SPNREPLY\_V1

The **DRS\_MSG\_SPNREPLY\_V1** structure defines a response message received from the [IDL DRSWriteSPN](#) method.

```
typedef struct {
 DWORD retVal;
} DRS_MSG_SPNREPLY_V1;
```

**retVal:** 0, or a Windows error code.

#### 4.1.20.1.5 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<br>(0x00000000) | Adds the specified values to the existing set of SPNs. |

| Value                                 | Meaning                                                                                                                                     |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| DS_SPN_REPLACE_SPN_OP<br>(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<br>(0x00000002)  | Removes all the existing SPNs.                                                                                                              |

## 4.1.20.2 Method-Specific Abstract Types and Procedures

### 4.1.20.2.1 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, and 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 method call.

### 4.1.20.3 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

ValidatedDRSInput(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
 end
end

```

```

 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 opnums for the dsaop RPC interface of the DRDM Remote Protocol. The methods themselves are not documented here and are not required for interoperation with Windows client operating systems.

Methods in RPC Opnum Order

| Method                          | Description |
|---------------------------------|-------------|
| <b>Opnum0ServerToServerOnly</b> | Opnum: 0    |

| Method                          | Description |
|---------------------------------|-------------|
| <b>Opnum1ServerToServerOnly</b> | Opnum: 1    |

## 5 Common Data Types, Variables, and Procedures

This section contains types that are used by two or more [drsuapi](#) 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. 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.

### 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. The AccessCheckCAR procedure returns false otherwise.

See [\[MS-ADTS\]](#) section 5.1.3 for the specification of this procedure.

See [\[MS-ADTS\]](#) section 5.1.3.2.1 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 AccessCheckObject 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 AmIRODC

```

procedure AmIRODC() : boolean

```

The AmIRODC procedure returns true if the DC is an RODC.

```
return DSAObj()!objectCategory = SchemaObj(nTDSASAR0)
```

## 5.7 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 > 0xFFFFFFFF 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
```

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  $x.uuidOriginating < y.uuidOriginating$

## 5.8 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 [5.12.1](#), for example, "Object(DS-DN)" and "Object(DN-Binary)".

## 5.9 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.

## 5.10 ATTRTYP

**ATTRTYP** is a concrete type for a compact representation of an **OID**.

This type is declared as follows:

```
typedef ULONG ATTRTYP;
```

Section [5.12.2](#) specifies the procedures that map between ATTRTYP and OID with the aid of a [SCHEMA\\_PREFIX\\_TABLE](#).

## 5.11 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
```

## 5.12 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.

### 5.12.1 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        | <a href="#">[RFC2252]</a> section 6.4              |
| Enumeration (2.5.5.9)   | INTEGER        | <a href="#">[RFC2252]</a> section 6.16             |
| Integer (2.5.5.9)       | INTEGER        | <a href="#">[RFC2252]</a> section 6.16             |
| LargeInteger (2.5.5.16) | INTEGER        | <a href="#">[RFC2252]</a> section 6.16             |

| LDAP syntax name (OID)                  | [RFC2252] name       | Reference section in [RFC2252] or in this document |
|-----------------------------------------|----------------------|----------------------------------------------------|
| Object(Presentation-Address) (2.5.5.13) | Presentation Address | <a href="#">[RFC2252]</a> section 6.28             |
| Object(Replica-Link) (2.5.5.10)         | Binary               | <a href="#">[RFC2252]</a> section 6.2              |
| String(IA5) (2.5.5.5)                   | IA5 String           | <a href="#">[RFC2252]</a> section 6.15             |
| String(Numeric) (2.5.5.6)               | Numeric String       | <a href="#">[RFC2252]</a> section 6.23             |
| String(Object-Identifier) (2.5.5.2)     | OID                  | <a href="#">[RFC2252]</a> section 6.25             |
| String(Octet) (2.5.5.10)                | Binary               | <a href="#">[RFC2252]</a> section 6.2              |
| String(Printable) (2.5.5.5)             | Printable String     | <a href="#">[RFC2252]</a> section 6.29             |
| String(Unicode) (2.5.5.12)              | Directory String     | <a href="#">[RFC2252]</a> section 6.10             |
| String(UTC-Time) (2.5.5.11)             | UTC Time             | <a href="#">[RFC2252]</a> section 6.31             |
| String(Generalized-Time) (2.5.5.11)     | Generalized Time     | <a href="#">[RFC2252]</a> section 6.14             |
| Object(DS-DN) (2.5.5.1)                 | -                    | Section <a href="#">5.12.1.1</a>                   |
| Object(DN-String) (2.5.5.14)            | -                    | Section <a href="#">5.12.1.2</a>                   |
| Object(DN-Binary) (2.5.5.7)             | -                    | Section <a href="#">5.12.1.3</a>                   |
| Object(Access-Point) (2.5.5.14)         | -                    | Section <a href="#">5.12.1.4</a>                   |
| Object(OR-Name) (2.5.5.7)               | -                    | Section <a href="#">5.12.1.5</a>                   |
| String(NT-Sec-Desc) (2.5.5.15)          | -                    | Section <a href="#">5.12.1.6</a>                   |
| String(SID) (2.5.5.17)                  | -                    | Section <a href="#">5.12.1.7</a>                   |
| String(Teletex) (2.5.5.4)               | -                    | Section <a href="#">5.12.1.8</a>                   |

The LDAP syntaxes that are not defined in [\[RFC2252\]](#) are described in the following sections.

#### 5.12.1.1 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.

#### 5.12.1.2 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\(DS-DN\)](#).

#### 5.12.1.3 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\(DS-DN\)](#).

#### 5.12.1.4 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\(DS-DN\)](#).

#### 5.12.1.5 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\(DS-DN\)](#).

#### 5.12.1.6 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.

#### 5.12.1.7 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.

#### 5.12.1.8 String(Teletex)

A value with the String(Teletex) syntax is a UTF-8 string restricted to characters with values between 0x20 and 0x7e, inclusive.

### 5.12.2 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 [\[ITUX690\]](#) 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: uncodestring,
 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: uniocesting) : 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) : uniocesting
```

Converts a binary OID to its string representation, as described in [\[ITU690\]](#) section 8.19; returns null if the conversion fails. For example, the binary OID \x55\x06\x02 is converted to the OID string "2.5.6.2".

```
procedure ToBinaryOID(s: uniocesting) : sequence of BYTE
```

Converts an OID string representation to a binary OID, as described in [\[ITU690\]](#) section 8.19; returns null if the conversion fails. For example, the OID string "2.5.6.2" is converted to the binary form \x55\x06\x02.

```
procedure ToByte(i: integer) : BYTE
```

Converts an integer into a byte representation, truncating to the least significant digits, if needed. For example, 2 converts to \x02.

```
procedure SubBinary(b: sequence of BYTE,
 start: integer, end: integer) : sequence of BYTE
```

Returns the sequence [start .. end] of bytes in b.

```
procedure AddPrefixTableEntry(var t: PrefixTable, o: sequence of BYTE)
```

Sets `t[t.length].prefixString` to `o`. Generates a random number between 0 and 65535 that is unique in the values of `prefixIndex` in `t`, and sets `t[t.length].prefixIndex` to the generated random number. Increases `t.length` by one.

```
procedure ToInteger(s: unicodestring) : integer
```

Converts a string to its integer representation. For example, "127" is 127. Strings with 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

```

### 5.13 BOOL

A concrete type for a Boolean value, as specified in [\[MS-DTYP\]](#) section 2.2.3.

### 5.14 BYTE

A concrete type for a single byte, as specified in [\[MS-DTYP\]](#) section 2.2.6.

### 5.15 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.

### 5.16 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.

### 5.17 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

```

## 5.18 ConfigNC

```

procedure ConfigNC(): DSName

```

The ConfigNC procedure returns the dsname of [dc.configNC](#).

## 5.19 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,
 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
```

## 5.20 DefaultNC

```
procedure DefaultNC(): DSName
```

The DefaultNC procedure returns the dsname of the [dc](#).defaultNC.

## 5.21 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
```

## 5.22 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.

## 5.23 DN

DN is an abstract type that is a *unicodestring* (section [3.4.3](#)) that contains a DN of the form specified in [RFC2253](#).

## 5.24 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](#)]

## 5.25 DomainNameFromNT4AccountName

```
procedure DomainNameFromNT4AccountName(
 nt4AccountName: uncodestring): uncodestring
```

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.

## 5.26 DRS\_EXTENSIONS

The **DRS\_EXTENSIONS** structure defines a concrete type for capabilities information used in version negotiation.

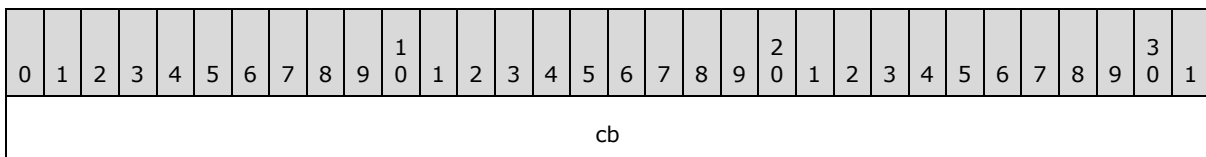
```
typedef struct {
 [range(1,10000)] DWORD cb;
 [size_is(cb)] BYTE rgb[];
} DRS_EXTENSIONS;
```

**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.

## 5.27 DRS\_EXTENSIONS\_INT

The **DRS\_EXTENSIONS\_INT** structure is a concrete type for structured capabilities information used in version negotiation. Each client and server produces a **DRS\_EXTENSIONS\_INT** structure.





Additionally, it signifies that the server supports a version of the server-to-server replication implementation. In this latter usage, this flag MAY have meaning to peer servers and applications but is not required for interoperability with Windows clients.

**RM (DRS\_EXT\_REMOVEAPI, 0x00000004):** If present, signifies that the server supports [IDL DRSRemoveDsServer](#) and [IDL DRSRemoveDsDomain](#).

**MV (DRS\_EXT\_MOVEREQ\_V2, 0x00000008):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperability with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**DF (DRS\_EXT\_GETCHG\_DEFLATE, 0x00000010):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperability with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**DC (DRS\_EXT\_DCINFO\_V1, 0x00000020):** If present, signifies that the server 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 server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperability with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**KE (DRS\_EXT\_KCC\_EXECUTE, 0x00000100):** If present, signifies that the server supports [IDL DRSExecuteKCC](#).

**AE2 (DRS\_EXT\_ADDENTRY\_V2, 0x00000200):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperability with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**LVR (DRS\_EXT\_LINKED\_VALUE\_REPLICATION, 0x00000400):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperability with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**DC2 (DRS\_EXT\_DCINFO\_V2, 0x00000800):** If present, signifies that the server 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\_CRYPTOP\_BIND, 0x00002000):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperability with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**GRI (DRS\_EXT\_GET\_REPL\_INFO, 0x00004000):** If present, signifies that the server supports [IDL DRSGetReplInfo](#).

**SE (DRS\_EXT\_STRONG\_ENCRYPTION, 0x0008000):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperoperation with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**DCF (DRS\_EXT\_DCINFO\_VFFFFFFFF, 0x00010000):** If present, signifies that the server supports [DRS MSG DCINFOREPLY VFFFFFFFF](#).

**TM (DRS\_EXT\_TRANSITIVE\_MEMBERSHIP, 0x00020000):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperoperation with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**SH (DRS\_EXT\_ADD\_SID\_HISTORY, 0x00040000):** If present, signifies that the server 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 server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperoperation with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**GM2 (DRS\_EXT\_GETMEMBERSHIP2, 0x00200000):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperoperation with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**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 server supports **application NCs**.

**GC8 (DRS\_EXT\_GETCHGREQ\_V8, 0x01000000):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperoperation with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**GR5 (DRS\_EXT\_GETCHGREPLY\_V5, 0x02000000):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperoperation with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**GR6 (DRS\_EXT\_GETCHGREPLY\_V6, 0x04000000):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperoperation with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**WB3 (DRS\_EXT\_WHISTLER\_BETA3, 0x08000000):** If present, signifies that the server supports [DRS MSG REPVERIFYOBJ](#).

Additionally, it signifies that the server supports versions of server-to-server replication implementation methods. In this latter usage, this flag MAY have meaning to peer servers and applications but is not required for interoperation with Windows clients.

**DF2 (DRS\_EXT\_W2K3\_DEFLATE, 0x10000000):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperation with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**GC10 (DRS\_EXT\_GETCHGREQ\_V10, 0x20000000):** If present, signifies that the server supports a version for the server-to-server replication implementation. This flag MAY have meaning to peer servers and applications but is not required for interoperation with Windows clients. Licensees may implement any behavior of their choosing for this flag.

**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 server DC's **DSA object** is a descendant. For 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. [<19>](#)

**dwReplEpoch (4 bytes):** A 32-bit, unsigned integer value. 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. [<20>](#)

**dwFlagsExt (4 bytes):** An extension of the dwFlags field that contains individual bit flags that describe the capabilities of the server that produced the **DRS\_EXTENSIONS\_INT** structure. For 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 in little-endian byte order. [<21>](#)

| 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 server 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 server supports the DRS\_SPECIAL\_SECRET\_PROCESSING flag as well as **InfoLevel** 3 in [DRS\\_MSG\\_DCINFOREQ\\_V1](#).

**RB (DRS\_EXT\_RECYCLE\_BIN, 0x00000004):** If present, signifies that the server has enabled the **Recycle Bin optional feature**.

**ConfigObjGUID (16 bytes):** A GUID (as specified in [\[MS-DTYP\]](#) section 2.3.2.2). 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 a NULL GUID (see [\[MS-GLOS\]](#)).<22>

5.28 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 [4.1.2](#).

5.29 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 here in little-endian byte order, that contains the following values.

Six elements of the set are interpreted differently by different methods; such elements have multiple symbolic names. These flags control the behavior of the server-to-server replication implementation. These flags MAY have meaning to peer DCs and applications but are not required for interoperation with Windows clients. Licensees may implement any behavior of their choosing for these flags.

|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| M | P | I | W | A | A | G | A | F | S | N  | G  | G  | C  | T  | A  | I  | S  | N  | X  | X  | X  | F  | X  | X  | 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  |    |    |    | S  |    |    | P  | N  | C  | P  | A  | F  | E  |
|   |   |   |   | L |   |   |   | / | / | R  | /  |    |    |    | R  | N  |    | Y  |    |    |    | P  |    |    |    |    |    | S  |    |    |    |
|   |   |   |   | / |   |   |   | N | R |    | L  |    |    |    | /  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   | D |   |   |   | S | F |    | O  |    |    |    | I  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|   |   |   |   | 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 \(section 4.1.19\)](#).

**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):** This flag is used only in server-to-server replication implementation. It MAY have meaning to peer servers, but it is not required for interoperation with Windows clients. Licensees can implement any behavior of their choosing for this flag.

**GS (DRS\_GET\_NC\_SIZE, 0x00001000):** This flag is used only in server-to-server replication implementation. It MAY have meaning to peer servers, but it is not required for interoperation with Windows clients. Licensees can implement any behavior of their choosing for this flag.

**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):** This flag is used only in server-to-server replication implementation. It MAY have meaning to peer servers, but it is not required for interoperation with Windows clients. Licensees can implement any behavior of their choosing for this flag.

**NS (DRS\_NO\_SOURCE, 0x00008000):** The NC replica has no server DCs.

**FSP (DRS\_FULL\_SYNC\_PACKET, 0x00020000):** This flag is used only in server-to-server replication implementation. It MAY have meaning to peer servers, but it is not required for interoperation with Windows clients. Licensees can implement any behavior of their choosing for this flag.

**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):** This flag is used only in server-to-server replication implementation. It MAY have meaning to peer servers, but it is not required for interoperation with Windows clients. Licensees can implement any behavior of their choosing for this flag.

**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, 0x40000000):** This flag is used only in server-to-server replication implementation. It MAY have meaning to peer servers, but it is not required for interoperation with Windows clients. Licensees can implement any behavior of their choosing for this flag.

For information about the Windows versions in which these flags were introduced and supported, see the following behavior note. [<23>](#)

### 5.30 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". This SPN is used for mutual authentication in the server-to-server replication implementation.

### 5.31 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.

### 5.32 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
```

### 5.33 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 \(section 5.87\)](#) 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.

### 5.34 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.

## 5.35 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 [\[MS-WSO\]](#) 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. 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 null terminator, 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 null terminator. Each Unicode value is encoded as 2 bytes. The byte ordering is little-endian. [<24>](#)

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.

5.35.1 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  $\neq$  0.
  - x.SidLen = y.SidLen.
  - x.Sid and y.Sid contain identical values in the first x.SidLen array items.

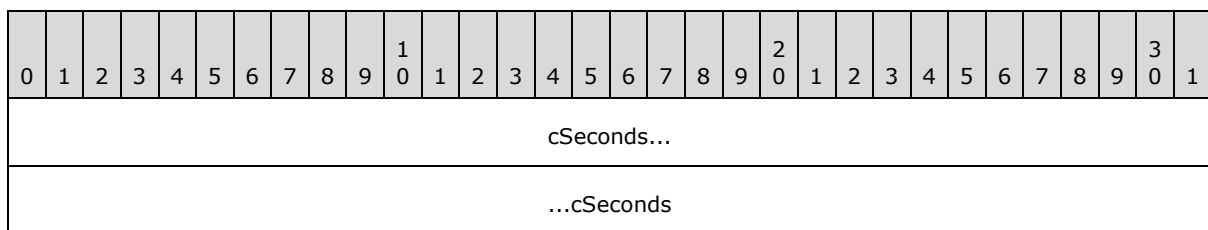
### 5.36 DSTIME

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.

### 5.37 DWORD

A concrete type for a 32-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.9.

### 5.38 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.

5.39 FILETIME

FILETIME is a concrete type for a time, as specified in [\[MS-DTYP\]](#) section 2.3.1.

5.40 FindCharRev

```
procedure FindCharRev(
 s: unicodestring,
 start: integer,
 c: UCHAR): integer
```

*Informative summary of behavior:* The FindCharRev procedure returns the zero-based index of the last occurrence of c in the portion of s between the start and the end of s.

If s = null, start < 0 or start > s.length-1, this procedure returns -1. Otherwise, let s be represented as the sequence of characters {s[0], ... s[s.length - 1]}. Let i be such that i >= start, i <= s.length -1, s[i] = c, and s[i+1] ≠ c, ..., s[s.length - 1] ≠ c. If such an i exists, this procedure returns i. Otherwise, this procedure returns -1.

5.41 FOREST\_TRUST\_INFORMATION

FOREST\_TRUST\_INFORMATION is a concrete type for state information about trust relationships with other forests. This data is stored in objects of [class trustedDomain](#) in the domain NC replica of the forest root domain. Specifically, the [msDS-TrustForestTrustInfo](#) attribute on such objects contains information about the trusted forest or realm. The structure of the information contained in this attribute is represented in the following manner.

|                    |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|--------------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0                  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Version            |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RecordCount        |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Records (variable) |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| ...                |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

- Version (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 [5.41.1](#).

**Note** All fields have little-endian byte ordering.

### 5.41.1 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 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 20 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 30 | 1 |
|---|---|---|---|---|---|---|---|---|---|----|---|---|---|---|---|---|---|---|---|----|---|---|---|---|---|---|---|---|---|----|---|
| X | X | X | X | N | N | S | S | X | X | X  | X | X | X | X | X | X | X | X | X | X  | X | X | X | X | X | X | X | X | X | X  | X |
|   |   |   |   | D | D | D | D |   |   |    |   |   |   |   |   |   |   |   |   |    |   |   |   |   |   |   |   |   |   |    |   |
|   |   |   |   | C | A | C | A |   |   |    |   |   |   |   |   |   |   |   |   |    |   |   |   |   |   |   |   |   |   |    |   |

**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](#) 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 [5.42](#).

**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.

### 5.41.2 Determining If a Name Is in a Trusted Forest

This section describes procedures that use the forest trust information contained in the [msDS-TrustForestTrustInfo](#) attribute to determine if a given domain is in a trusted forest.

The procedures described in this subsection use the following data structures.

```
struct {
 ULONG RecordCount;
 PX_FOREST_TRUST_RECORD *Entries;
} X_FOREST_TRUST_INFORMATION;

struct {
 ULONG Flags;
 FOREST_TRUST_RECORD_TYPE ForestTrustType;
 LARGE_INTEGER Time;
 union {
 LPWSTR TopLevelName;
 X_FOREST_TRUST_DOMAIN_INFO DomainInfo;
 X_FOREST_TRUST_BINARY_DATA Data;
 } ForestTrustData;
} X_FOREST_TRUST_RECORD, *PX_FOREST_TRUST_RECORD;

struct {
 SID *Sid;
 LPWSTR DnsName;
 LPWSTR NetbiosName;
} X_FOREST_TRUST_DOMAIN_INFO;

struct {
 ULONG Length;
 BYTE *Buffer;
} X_FOREST_TRUST_BINARY_DATA;
```

The X\_FOREST\_TRUST\_INFORMATION structure previously defined is used by the procedure to determine if a given domain is in a trusted forest. To unmarshal the content of the [msDS-TrustForestTrustInfo](#) attribute into this structure, the UnmarsallForestTrustInfo procedure described below can be used.

```
procedure ExtractString(
 buffer: sequence of BYTE,
 index: DWORD, size: DWORD): unicodestring;
```

The sequence [index .. index + size] of bytes in 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

```

```

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
return false

procedure IsUPNInTrustedForest(upn: unicodestring): 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: unicodestring,
 superiordomainName: unicodestring): 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: unicodestring): 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

```

```

 IsSubdomainOf(name, e.TopLevelName))) then
 return false
 endif
 endfor

 /* if a suffix of the name is in the inclusion list and is
 * not disabled, the forest owns this name */
 foreach e in f.Entries
 if (e.ForestTrustType = ForestTrustTopLevelName and
 LSA_FTRECORD_DISABLED_REASONS not in e.Flags and
 (e.TopLevelName = 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
 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

```

## 5.42 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;

```

## 5.43 ForestRootDomainNC

```

procedure ForestRootDomainNC(): DSName

```

The ForestRootDomainNC procedure returns the [DSName](#) of the **forest root domain NC**.

## 5.44 FullReplicaExists

```

procedure FullReplicaExists(nc : DSName) : boolean

```

The FullReplicaExists procedure returns true if the NC replica with root nc is a full replica.

```

if not ObjExists(nc) then
 return false
endif
return nc in (DSAObj()!msDS-hasMasterNCs +

```

```
DSASObj() !msDS-hasFullReplicaNCs)
```

## 5.45 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.

## 5.46 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
```

## 5.47 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.

## 5.48 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     |

| Symbolic constant   | Value |
|---------------------|-------|
| FSMO_DOMAIN_NAMING  | 2     |
| FSMO_PDC            | 3     |
| FSMO_RID            | 4     |
| FSMO_INFRASTRUCTURE | 5     |

## 5.49 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".

## 5.50 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 [\[RFC2253\]](#).

## 5.51 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".

## 5.52 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".

## 5.53 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 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 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.

## 5.54 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.

## 5.55 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.

## 5.56 GuidToString

```
procedure GuidToString(guid: GUID): uncodestring
```

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}.

## 5.57 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.

## 5.58 instanceType Bit Flags

The instanceType bit flags are bits that may appear in values of the [instanceType](#) attribute. The bit flags 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 | 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.

## 5.59 Is2PartSPN

```
procedure Is2PartSPN(spn: uncodestring): boolean
```

The Is2PartSPN procedure returns true if spn is an SPN consisting of two parts, and false otherwise.

## 5.60 Is3PartSPN

```
procedure Is3PartSPN(spn: uncodestring): boolean
```

The Is3PartSPN procedure returns true if spn is an SPN consisting of three parts, and false otherwise.

## 5.61 IsAdlds

```
procedure IsAdlds() : boolean
```

If the local DC is running Active Directory as AD LDS, this procedure returns TRUE. It returns FALSE otherwise.

## 5.62 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.

## 5.63 IsDomainNameInTrustedForest

```
procedure IsDomainNameInTrustedForest(name: unicodestring,
var referredDomain: unicodestring) : boolean
```

The IsDomainNameInTrustedForest procedure returns true if the domain identified by *name* is in a forest trusted by the caller's forest, as determined by the [FOREST TRUST INFORMATION](#) state of the caller's forest, and false otherwise. The *name* parameter can be either 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 [5.41](#) for the specification of this procedure.

## 5.64 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 [5.41.2](#) for the specification of this procedure.

## 5.65 IsDCAccount

```
procedure IsDCAccount(o: DSName): boolean
```

The IsDCAccount procedure returns true if the object represents the computer account of a DC.

## 5.66 IsForwardLinkAttribute

```
procedure IsForwardLinkAttribute (att:ATTYTP): 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.

## 5.67 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.

## 5.68 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>".

## 5.69 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.

## 5.70 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.

## 5.71 IsUPNInTrustedForest

```
procedure 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 [5.41](#) for the specification of this procedure.

## 5.72 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.

## 5.73 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.

## 5.74 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.

## 5.75 LARGE\_INTEGER

A concrete type for a 64-bit signed integer, as specified in [\[MS-DTYP\]](#) section 2.3.3.

## 5.76 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 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 20 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 30 | 1 |   |
|---|---|---|---|---|---|---|---|---|---|----|---|---|---|---|---|---|---|---|---|----|---|---|---|---|---|---|---|---|---|----|---|---|
| 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.

## 5.77 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.ldapConnections](#), which maintains the DC's LDAPConnections state.

## 5.78 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.

## 5.79 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

## 5.80 LinkValueStampCompare

```
procedure LinkValueStampCompare(
 LinkValueStamp stamp1,
 LinkValueStamp stamp2): integer
```

*Informative summary of behavior:* The LinkValueStampCompare procedure compares two [LinkValueStamps](#), stamp1 and stamp2. If stamp1 is greater than stamp2, then the procedure

returns an integer with a value greater than 0. If stamp1 is equal to stamp2, then the procedure returns 0. If stamp1 is less than 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
```

## 5.81 LONG

A concrete type for a 32-bit, signed integer, as specified in [\[MS-DTYP\]](#) section 2.2.27.

## 5.82 LONGLONG

A concrete type for a 64-bit, signed integer, as specified in [\[MS-DTYP\]](#) section 2.2.28.

### 5.83 LPWSTR

A concrete type for a pointer to a string of double-byte Unicode characters, as specified in [\[MS-DTYP\]](#) section 2.2.35.

### 5.84 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 [5.12.2](#) for the specification of this procedure.

### 5.85 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.

### 5.86 MTX\_ADDR

The **MTX\_ADDR** structure defines a concrete type for the network name of a DC.

```
typedef struct {
 [range(1,256)] unsigned long mtx_namelen;
 [size_is(mtx_namelen)] char mtx_name[];
} MTX_ADDR;
```

**mtx\_namelen:** A 32-bit, unsigned integer that specifies the number of bytes in **mtx\_name**, including a null terminating character.

**mtx\_name:** The UTF-8 encoding of a [NetworkAddress](#).

The following table shows an alternative representation of this structure.

|                                |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|--------------------------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0                              | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| mtx_namelen                    |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| mtx_name (variable length) ... |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

## 5.87 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 [\[RFC2821\]](#)). 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 **ldapPort** 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.

## 5.88 NewPrefixTable

```
procedure NewPrefixTable() : PrefixTable
```

The NewPrefixTable procedure creates a new [PrefixTable](#) that contains a set of default prefixes. See section [5.12.2](#) for the specification of this procedure.

## 5.89 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.

## 5.90 NTSAPI\_CLIENT\_GUID

NTDSAPI\_CLIENT\_GUID is a value of type GUID that is defined as {e24d201a-4fd6-11d1-a3da-0000f875ae0d}.

## 5.91 NTDSTRANSPORT\_OPT Values

The valid system flags used on directory objects are specified in [\[MS-ADTS\]](#) section 7.1.1.2.2.3.1.

## 5.92 NULLGUID

NULLGUID is a value of type [GUID](#) that is entirely zero, that is, {00000000-0000-0000-0000-000000000000}.

## 5.93 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
```

## 5.94 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".

## 5.95 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.

## 5.96 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 [5.12.2](#) for the specification of this procedure.

## 5.97 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.

## 5.98 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.

## 5.99 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.

## 5.100 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
```

## 5.101 PAS\_DATA

PAS\_DATA is a concrete type for a list of attributes in a **partial attribute set**.

|         |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0       | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| version |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

|                |
|----------------|
| size           |
| flag           |
| pas (variable) |
| ...            |

**version (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.

### 5.102 PerformReplication

```
procedure PerformReplication(
 nc: DSName,
 dc: DC) : ULONG
```

The PerformReplication procedure executes the tasks necessary for replicating the specified NC from the specified DC. The success or failure of these tasks should be returned as a Microsoft Windows® error code.

### 5.103 PrefixTable

PrefixTable is an abstract type for a prefix table. See section [5.12.2](#) for the specification of this type.

### 5.104 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.

### 5.105 RDN

RDN is an abstract type for representing the relative distinguished name (RDN) (as specified in [RFC2253](#)).

## 5.106 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.

## 5.107 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. The delete operation transforms, as described in [\[MS-ADTS\]](#) section 3.1.1.5.5, 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.

## 5.108 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.

## 5.109 ReplicationQueue

The server-to-server replication implementation is synchronized on a queue for inbound replication tasks. `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.

### 5.110 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.



### 5.111 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*.

### 5.112 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. This attribute exposes a structure that controls the server-to-server replication implementation—specifically, the options for inbound replication of changes.

This structure MAY have meaning to peer DCs and applications, but is not required for interoperation with Windows clients.

**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. [<25>](#)

**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 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.

**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.

### 5.113 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 attribute exposes a structure that controls the server-to-server replication implementation—specifically, the options for outbound replication of changes.

This structure MAY have meaning to peer DCs and applications but is not required for interoperation with Windows clients.

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. [<26>](#)

**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 [5.29](#)) 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.

### 5.114 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](#). These flags are used by the server-to-server replication implementation. 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.
- 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 attempt, 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](#).

**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.

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.

### 5.115 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.

### 5.116 Rid

Rid is an abstract type that consists of an integer that represents the RID component of a SID, as specified in [\[MS-DTYP\]](#) section 2.4.2 and [\[MS-WSO\]](#) section 3.1.2.1.3.

### 5.117 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.

### 5.118 RIGHT Values

The valid access rights used in **ACLs** in security descriptors are defined in [\[MS-ADTS\]](#) section 5.1.3.2.

### 5.119 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\_DRSSBind 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.

## 5.120 `RPCOutgoingContexts`

`RPCOutgoingContexts` is an abstract type that is a sequence of tuples, one tuple per RPC context for an outgoing server-to-server RPC session from the DC to another DC. This implementation uses a cache, and some of the fields in the tuples reference this implementation. 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 an implementation-specific 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.

## 5.121 `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      |
|--------------------------------------------|------------|
| <code>SAM_GROUP_OBJECT</code>              | 0x10000000 |
| <code>SAM_NON_SECURITY_GROUP_OBJECT</code> | 0x10000001 |
| <code>SAM_ALIAS_OBJECT</code>              | 0x20000000 |
| <code>SAM_NON_SECURITY_ALIAS_OBJECT</code> | 0x20000001 |
| <code>SAM_USER_OBJECT</code>               | 0x30000000 |
| <code>SAM_MACHINE_ACCOUNT</code>           | 0x30000001 |

| Symbolic name     | Value      |
|-------------------|------------|
| SAM_TRUST_ACCOUNT | 0x30000002 |

Only the values used by this protocol are contained in this table.

## 5.122 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.

## 5.123 SchemaNC

```
procedure SchemaNC(): DSName
```

The SchemaNC procedure returns the [DSName](#) of [dc.schemaNC](#).

## 5.124 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
```

## 5.125 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*<sup>^</sup> parameter of IDL\_DRSBind are set to 0.

## 5.126 SID

A concrete type for the Microsoft Windows NT® operating system **SID** structure, as specified in [\[MS-DTYP\]](#) section 2.4.2.

### 5.127 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.

### 5.128 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) in the server-to-server replication implementation.

```
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
```

### 5.129 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.

### 5.130 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.

### 5.131 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.

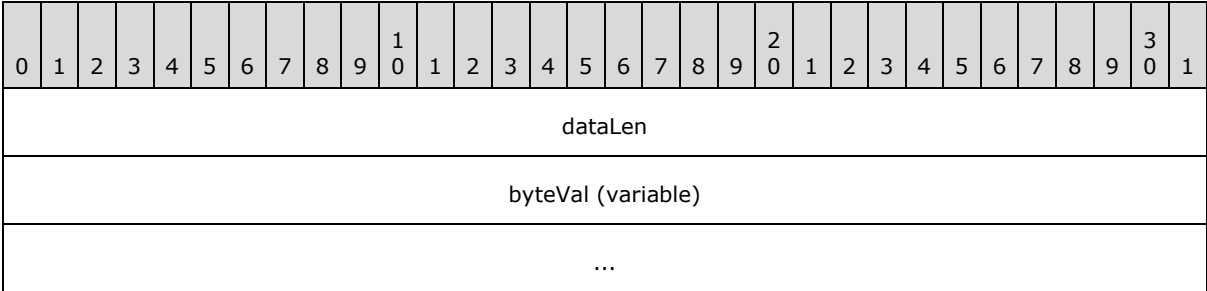
5.132 Syntax

```
procedure Syntax(attr: ATTRTYP): AttributeSyntax
```

The Syntax procedure returns the syntax of the attribute attr.

5.133 SYNTAX\_ADDRESS

The SYNTAX\_ADDRESS packet is the concrete type for a sequence of bytes or Unicode characters.



**dataLen (4 bytes):** The size of the entire structure (including this field), in bytes.

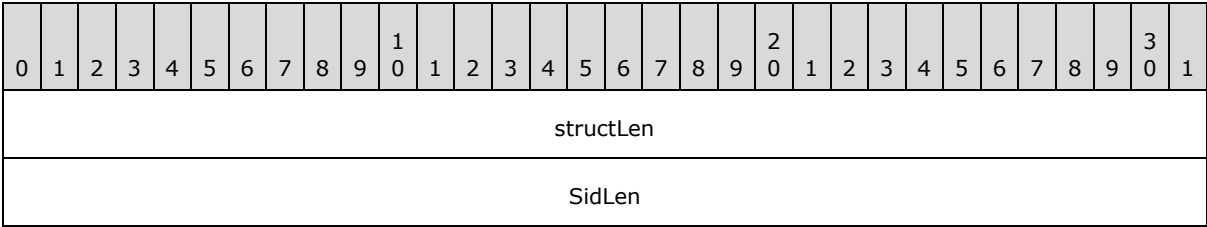
**byteVal (variable):** The byte or character data.

The following structure definition shows an alternative representation of this data type.

```
typedef struct {
 DWORD dataLen;
 union {
 BYTE byteVal[];
 wchar_t uVal[];
 };
} SYNTAX_ADDRESS;
```

5.134 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.





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 null terminator, used to represent the object's [distinguishedName](#) attribute value. Zero indicates that the SYNTAX\_DISTNAME\_BINARY does not identify the [distinguishedName](#) value of the directory object.

**StringName (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 null terminator. 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;
```

### 5.135 systemFlags Values

The valid system flags used on directory objects are defined in [\[MS-ADTS\]](#) section 2.2.10.

### 5.136 UCHAR

A concrete type, as defined in [\[MS-DTYP\]](#) section 2.2.44. A UCHAR is an 8-bit, unsigned quantity.

### 5.137 ULONG

A concrete type for a 32-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.50.

### 5.138 ULONGLONG

A concrete type for a 64-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.54.

### 5.139 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*.

## 5.140 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*.

## 5.141 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.

## 5.142 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.

## 5.143 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.

|   |   |   |   |   |   |   |   |   |   | 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 | L | X | X | A | X | X | X | S | W | I | X | N | D | X | X | X | X | X | X | X | X | X | X | X | X | X | P | X | X |  |  |  |  |  |  |  |  |
|   |   |   | O |   |   | D |   |   |   | T | T | D |   | A | A |   |   |   |   |   |   |   |   |   |   |   |   | S |   |   |   |  |  |  |  |  |  |  |  |

**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.

### 5.144 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.

### 5.145 USHORT

A concrete type for a 16-bit, unsigned integer, as specified in [\[MS-DTYP\]](#) section 2.2.57.

### 5.146 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;
```

### 5.147 USN\_VECTOR

The **USN\_VECTOR** structure defines a concrete type to pass implementation-specific state between calls to server-to-server replication.

```
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 DRDM IDL. All uses of this structure are implementation-specific.

## 5.148 UUID

UUID is a type that is equivalent to the [GUID](#) type.

## 5.149 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. [<27>](#)

The server MUST raise an ERROR\_INVALID\_PARAMETER exception when opnum = 27 and IsAdlds() == false.

## 5.150 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_DRSSWriteSPN*/
 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)

```

### 5.151 Value

Value is an abstract type for attribute values used for abstract value representation (see section [5.12.1](#)).

### 5.152 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 client-to-DC authentication | Section <a href="#">2.2.4</a> |

## 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
{

typedefef LONGLONG DSTIME;

typedefef [context_handle] void * DRS_HANDLE;

typedefef struct {
 char Data[28];
} NT4SID;

typedefef struct {
 unsigned long structLen;
 unsigned long SidLen;
 GUID Guid;
 NT4SID Sid;
 unsigned long NameLen;
 [range(0, 1048576)] [size_is(NameLen + 1)] WCHAR StringName[];
} DSNAME;

typedefef LONGLONG USN;

typedefef struct {
 USN usnHighObjUpdate;
 USN usnReserved;
 USN usnHighPropUpdate;
} USN_VECTOR;

typedefef struct {
 UUID uuidDsa;
 USN usnHighPropUpdate;
} UPTODATE_CURSOR_V1;

typedefef struct {
 DWORD dwVersion;
 DWORD dwReserved1;
 [range(0,1048576)] DWORD cNumCursors;
 DWORD dwReserved2;
 [size_is(cNumCursors)] UPTODATE_CURSOR_V1 rgCursors[];
} UPTODATE_VECTOR_V1_EXT;

typedefef struct {
 [range(0,10000)] unsigned int length;
 [size_is(length)] BYTE *elements;
} OID_t;

typedefef struct {
```

```

 unsigned long ndx;
 OID_t prefix;
 } PrefixTableEntry;

typedef struct {
 [range(0,1048576)] DWORD PrefixCount;
 [size_is(PrefixCount)] PrefixTableEntry *pPrefixEntry;
} SCHEMA_PREFIX_TABLE;

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 {
 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 {
 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_FFFFFFFW;

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 rgMetaEntry[];
} 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 {
 [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 {
 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 operation;
 DWORD flags;
 [string] const WCHAR *pwszAccount;
 [range(0,10000)] DWORD cSPN;
 [string, size_is(cSPN)] const WCHAR **rpwszSPN;
} DRS_MSG_SPNREQ_V1;

typedef [switch_type(DWORD)] union {
 [case(1)] DRS_MSG_SPNREQ_V1 V1;
} DRS_MSG_SPNREQ;

typedef struct {
 DWORD retVal;
} DRS_MSG_SPNREPLY_V1;

typedef [switch_type(DWORD)] union {
 [case(1)] DRS_MSG_SPNREPLY_V1 V1;
} DRS_MSG_SPNREPLY;

typedef struct {
 [string] LPWSTR ServerDN;
 [string] LPWSTR DomainDN;
 BOOL fCommit;
} DRS_MSG_RMSVRREQ_V1;

typedef [switch_type(DWORD)] union {
 [case(1)] DRS_MSG_RMSVRREQ_V1 V1;
} DRS_MSG_RMSVRREQ;

typedef struct {
 BOOL fLastDcInDomain;
} DRS_MSG_RMSVRREPLY_V1;

typedef [switch_type(DWORD)] union {
 [case(1)] DRS_MSG_RMSVRREPLY_V1 V1;
} DRS_MSG_RMSVRREPLY;

typedef struct {
 [string] LPWSTR DomainDN;
} DRS_MSG_RMDMREQ_V1;

```

```

typedef [switch_type(DWORD)] union {
 [case(1)] DRS_MSG_RMDMNREQ_V1 V1;
} DRS_MSG_RMDMNREQ;

typedef struct {
 DWORD Reserved;
} DRS_MSG_RMDMNREPLY_V1;

typedef [switch_type(DWORD)] union {
 [case(1)] DRS_MSG_RMDMNREPLY_V1 V1;
} DRS_MSG_RMDMNREPLY;

typedef struct {
 [string] WCHAR *Domain;
 DWORD InfoLevel;
} DRS_MSG_DCINFOREQ_V1;

typedef [switch_type(DWORD)] union {
 [case(1)] DRS_MSG_DCINFOREQ_V1 V1;
} DRS_MSG_DCINFOREQ, *PDRS_MSG_DCINFOREQ;

typedef struct {
 [range(0,10000)] DWORD cItems;
 [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_1W *rItems;
} DRS_MSG_DCINFOREPLY_V1;

typedef struct {
 [range(0,10000)] DWORD cItems;
 [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_2W *rItems;
} DRS_MSG_DCINFOREPLY_V2;

typedef struct {
 [range(0,10000)] DWORD cItems;
 [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_3W* rItems;
} DRS_MSG_DCINFOREPLY_V3;

typedef struct {
 [range(0,10000)] DWORD cItems;
 [size_is(cItems)] DS_DOMAIN_CONTROLLER_INFO_FFFFFFFF *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 {
 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 {
 [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 {
 [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_QUERY_SITESREQ;

typedef struct {
 DWORD dwErrorCode;
 DWORD dwCost;
} DRS_MSG_QUERY_SITESREPLYELEMENT_V1;

typedef struct {
 [range(0,10000)] DWORD cToSites;
 [size_is(cToSites)] DRS_MSG_QUERY_SITESREPLYELEMENT_V1 *rgCostInfo;
 DWORD dwFlags;
} DRS_MSG_QUERY_SITESREPLY_V1;

typedef [switch_type(DWORD)] union {
 [case(1)] DRS_MSG_QUERY_SITESREPLY_V1 V1;
} DRS_MSG_QUERY_SITESREPLY;

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
void Opnum3ServerToServerOnly(void);

// 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
void Opnum8ServerToServerOnly(void);

// Opnum 9
void Opnum9ServerToServerOnly(void);

// Opnum 10
void Opnum10ServerToServerOnly(void);

// Opnum 11
void Opnum11ServerToServerOnly(void);

// 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
void Opnum17ServerToServerOnly(void);

// 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
void Opnum21ServerToServerOnly(void);

// opnum 22
ULONG
IDL_DRSReplicaVerifyObjects(
 [in, ref] DRS_HANDLE hDrs,
 [in] DWORD dwVersion,
 [in, ref, switch_is(dwVersion)] DRS_MSG_REPVERIFYOBJ *pmsgVerify);

// Opnum 23
void Opnum23ServerToServerOnly(void);

// 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_QUERY_SITES_REPLY *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);

}

// The dsaop interface.
[
 uuid(7c44d7d4-31d5-424c-bd5e-2b3e1f323d22), version(1.0),
 pointer_default(unique)
]
interface dsaop
{

// opnum 0
void Opnum0ServerToServerOnly(void);

// opnum 1
void Opnum1ServerToServerOnly(void);

}

```

## 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® 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
- Active Directory® Application Mode (ADAM)
- 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_GETREPLINFO_REQ_V2 | <a href="#">4.1.4.1.6</a> | Windows Server® 2003 operating system |

| Data type or IDL method      | Section                    | Windows version introduced            |
|------------------------------|----------------------------|---------------------------------------|
| DRS_MSG_DCINFOREPLY_V3       | <a href="#">4.1.4.1.6</a>  | Windows Server 2008                   |
| DS_DOMAIN_CONTROLLER_INFO_3W | <a href="#">4.1.4.1.10</a> | Windows Server 2008                   |
| IDL_DRSReplicaVerifyObjects  | <a href="#">4.1.17</a>     | Windows Server® 2003 operating system |
| IDL_DRSFinishDemotion        | <a href="#">4.1.6</a>      | Windows Server 2008                   |
| IDL_DRSInitDemotion          | <a href="#">4.1.8</a>      | Windows Server® 2008 operating system |
| IDL_DRSReplicaDemotion       | <a href="#">4.1.14</a>     | 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 |
|--------------------------------------------------|--------------------------------------------------------------|
| <a href="#">IDL DRSBind</a><br>Opnum: 0          | Yes                                                          |
| <a href="#">IDL DRSUnbind</a><br>Opnum: 1        | Yes                                                          |
| <a href="#">IDL DRSReplicaSync</a><br>Opnum: 2   | Yes                                                          |
| <b>Opnum3ServerToServerOnly</b><br>Opnum: 3      | No                                                           |
| <a href="#">IDL DRSUpdateRefs</a><br>Opnum: 4    | Yes                                                          |
| <a href="#">IDL DRSReplicaAdd</a><br>Opnum: 5    | Yes                                                          |
| <a href="#">IDL DRSReplicaDel</a><br>Opnum: 6    | Yes                                                          |
| <a href="#">IDL DRSReplicaModify</a><br>Opnum: 7 | Yes                                                          |
| <b>Opnum8ServerToServerOnly</b><br>Opnum: 8      | No                                                           |
| <b>Opnum9ServerToServerOnly</b><br>Opnum: 9      | No                                                           |
| <b>Opnum10ServerToServerOnly</b><br>Opnum: 10    | No                                                           |
| <b>Opnum11ServerToServerOnly</b><br>Opnum: 11    | No                                                           |

| Method                                                    | Windows client operating systems can implement a client role |
|-----------------------------------------------------------|--------------------------------------------------------------|
| <a href="#">IDL_DRS CrackNames</a><br>Opnum: 12           | Yes                                                          |
| <a href="#">IDL_DRS WriteSPN</a><br>Opnum: 13             | Yes                                                          |
| <a href="#">IDL_DRS RemoveDsServer</a><br>Opnum: 14       | Yes                                                          |
| <a href="#">IDL_DRS RemoveDsDomain</a><br>Opnum: 15       | Yes                                                          |
| <a href="#">IDL_DRS DomainControllerInfo</a><br>Opnum: 16 | Yes                                                          |
| <b>Opnum17ServerToServerOnly</b><br>Opnum: 17             | No                                                           |
| <a href="#">IDL_DRS ExecuteKCC</a><br>Opnum: 18           | Yes                                                          |
| <a href="#">IDL_DRS GetReplInfo</a><br>Opnum: 19          | Yes                                                          |
| <a href="#">IDL_DRS AddSidHistory</a><br>Opnum: 20        | Yes                                                          |
| <b>Opnum21ServerToServerOnly</b><br>Opnum: 21             | No                                                           |
| <a href="#">IDL_DRS ReplicaVerifyObjects</a><br>Opnum: 22 | Yes                                                          |
| <b>Opnum23ServerToServerOnly</b><br>Opnum: 23             | No                                                           |
| <a href="#">IDL_DRS QuerySitesByCost</a><br>Opnum: 24     | Yes                                                          |
| <a href="#">IDL_DRS InitDemotion</a><br>Opnum: 25         | Yes                                                          |
| <a href="#">IDL_DRS ReplicaDemotion</a><br>Opnum: 26      | Yes                                                          |
| <a href="#">IDL_DRS FinishDemotion</a><br>Opnum: 27       | Yes                                                          |

<6> [Section 4.1](#): Gaps in the opnum numbering sequence apply to Windows as follows.

| Opnum | Description                           |
|-------|---------------------------------------|
| 3     | Server-to-server implementation only. |
| 8     | Server-to-server implementation only. |
| 9     | Server-to-server implementation only. |
| 10    | Server-to-server implementation only. |
| 11    | Server-to-server implementation only. |
| 17    | Server-to-server implementation only. |
| 21    | Server-to-server implementation only. |
| 23    | Server-to-server implementation only. |

<7> [Section 4.1.1.2.6](#): 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] 0
{ 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.

<8> [Section 4.1.1.2.12](#): 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.

<9> [Section 4.1.2.1](#): 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.

<10> [Section 4.1.2.1](#): Windows DC clients not running Windows Server 2008 or Windows Server 2008 R2 do not include the **ConfigObjGUID** and **dwFlagsExt** field of the [DRS\\_EXTENSIONS\\_INT](#) structure.

<11> [Section 4.1.2.2](#): 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.

<12> [Section 4.1.4.2](#): All of the information levels listed in section [4.1.4.2](#) have existed in the [drsapi RPC interface](#) since Windows 2000, except as noted in the following table.

| Infolevel | Introduced in Windows version |
|-----------|-------------------------------|
| 3         | Windows Server 2008           |

<13> [Section 4.1.5.3](#): 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.

<14> [Section 4.1.6.3](#): 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.

<15> [Section 4.1.8.2](#): 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.

<16> [Section 5.27](#): 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.

<17> [Section 5.27](#): 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.

<18> [Section 5.27](#): Client callers set **dwFlags** to zero.

<19> [Section 5.27](#): This field contains the process ID of the client.

<20> [Section 5.27](#): 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.

<21> [Section 5.27](#): The **ConfigObjGUID** and **dwFlagsExt** fields in the **DRS\_EXTENSIONS\_INT** structure are included only by AD DS servers running Windows Server 2008 or Windows Server 2008 R2 and AD LDS servers running Windows Server 2003 R2, Windows Server 2008, or Windows Server 2008 R2.

<22> [Section 5.27](#): The **ConfigObjGUID** and **dwFlagsExt** fields in the **DRS\_EXTENSIONS\_INT** structure are included only by AD DS servers running Windows Server 2008 or Windows Server 2008 R2 and AD LDS servers running Windows Server 2003 R2, Windows Server 2008, or Windows Server 2008 R2.

<23> [Section 5.29](#): All the DRS\_OPTIONS listed in section [5.29](#) 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 [5.29](#) 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
```

<24> [Section 5.35](#): 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.

<25> [Section 5.112](#): 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.

<26> [Section 5.113](#): 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.

<27> [Section 5.149](#): Windows Server 2008 and Windows Server 2008 R2 do not raise ERROR\_INVALID\_PARAMETER exception when opnum==26 and IsAdlds() == false. Instead, the method [IDL\\_DRSReplicaDemotion \(section 4.1.14\)](#) 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-DRDM] 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](mailto:protocol@microsoft.com).

| Section                        | Tracking number (if applicable) and description                                                                                 | Major change (Y or N) | Change type      |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------|-----------------------|------------------|
| <a href="#">1.2 References</a> | 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](#) 38, [section 3.4.3](#) 40)
- [Abstract value representations](#) 258
- [AbstractPTFromConcretePT](#) 253
- [AccessCheckAttr](#) 253
- [AccessCheckCAR](#) 254
- [AccessCheckObject](#) 254
- [AccessCheckWriteToSpnAttribute](#) 254
- [AD LDS specifics](#) 49
- [AmIRODC](#) 255
- [Applicability](#) 31
- [Asynchronous processing](#) 46
- [Attributes](#) 43
- [AttributeStamp](#) 256
- [AttributeSyntax](#) 257
- [AttrStamp](#) 257
- [ATTRTYP](#) 257
- [AttrtypFromSchemaObj](#) 258
- [ATTRTYP-to-OID conversion](#) 261
- ATTRVAL
  - [abstract value representations](#) 258
  - [ATTRTYP-to-OID conversion](#) 261
  - [overview](#) 258
- [ATTRVAL structure](#) 258

### B

- [BOOL](#) 267
- [BYTE](#) 267

### C

- [Capability negotiation](#) 31
- [Change tracking](#) 355
- [Client initialization](#) 49
- [ClientExtensions](#) 267
- Client-to-DC operations
  - [security](#) 34
  - [security provider](#) 34
  - [SPN for target DC in AD DS](#) 34
  - [SPN for target DC in AD LDS](#) 35
- [Common configuration example](#) 47
- Concrete types ([section 3.3.3](#) 38, [section 3.4.2](#) 40)
- [ConcretePTFromAbstractPT](#) 267
- [ConfigNC](#) 268
- [Configuration example](#) 47

### D

- [Data display conventions](#) 48
- [Data types](#) 253
- [dc/DC](#) 268
- [DC-to-DC operations](#) 33
- [DefaultNC](#) 269
- [DescendantObject](#) 269
- [DN](#) 270
- [DNBinary](#) 270

- [DomainNameFromNT4AccountName](#) 270
- [DRS\\_EXTENSIONS](#) 270
- [DRS\\_EXTENSIONS structure](#) 270
- [DRS\\_EXTENSIONS\\_INT](#) 270
- [DRS\\_EXTENSIONS\\_INT packet](#) 270
- [DRS\\_HANDLE](#) 275
- [DRS\\_MSG\\_ADDSIDREPLY\\_V1 structure](#) 54
- [DRS\\_MSG\\_ADDSIDREQ\\_V1 structure](#) 53
- [DRS\\_MSG\\_CRACKREPLY\\_V1 structure](#) 85
- [DRS\\_MSG\\_CRACKREQ\\_V1 structure](#) 81
- [DRS\\_MSG\\_DCINFOREPLY\\_V1 structure](#) 107
- [DRS\\_MSG\\_DCINFOREPLY\\_V2 structure](#) 107
- [DRS\\_MSG\\_DCINFOREPLY\\_V3 structure](#) 107
- [DRS\\_MSG\\_DCINFOREPLY\\_VFFFFFFF structure](#) 108
- [DRS\\_MSG\\_DCINFOREQ\\_V1 structure](#) 106
- [DRS\\_MSG\\_FINISH\\_DEMOTIONREPLY\\_V1 structure](#) 126
- [DRS\\_MSG\\_FINISH\\_DEMOTIONREQ\\_V1 structure](#) 124
- [DRS\\_MSG\\_GETREPLINFO\\_REQ\\_V1 structure](#) 130
- [DRS\\_MSG\\_GETREPLINFO\\_REQ\\_V2 structure](#) 131
- [DRS\\_MSG\\_INIT\\_DEMOTIONREPLY\\_V1 structure](#) 183
- [DRS\\_MSG\\_INIT\\_DEMOTIONREQ\\_V1 structure](#) 182
- [DRS\\_MSG\\_KCC\\_EXECUTE\\_V1 structure](#) 122
- [DRS\\_MSG\\_QUERYsitesREPLY\\_V1 structure](#) 185
- [DRS\\_MSG\\_QUERYsitesREPLYELEMENT\\_V1 structure](#) 186
- [DRS\\_MSG\\_QUERYsitesREQ\\_V1 structure](#) 185
- [DRS\\_MSG\\_REPADD\\_V1 structure](#) 213
- [DRS\\_MSG\\_REPADD\\_V2 structure](#) 213
- [DRS\\_MSG\\_REPDEL\\_V1 structure](#) 217
- [DRS\\_MSG\\_REPLICA\\_DEMOTIONREPLY\\_V1 structure](#) 223
- [DRS\\_MSG\\_REPLICA\\_DEMOTIONREQ\\_V1 structure](#) 222
- [DRS\\_MSG\\_REPMOD\\_V1 structure](#) 226
- [DRS\\_MSG\\_REPSYNC\\_V1 structure](#) 229
- [DRS\\_MSG\\_REPVERIFYOBJ\\_V1 structure](#) 232
- [DRS\\_MSG\\_RMDMNREPLY\\_V1 structure](#) 206
- [DRS\\_MSG\\_RMDMNRREQ\\_V1 structure](#) 205
- [DRS\\_MSG\\_RMSVRREPLY\\_V1 structure](#) 209
- [DRS\\_MSG\\_RMSVRREQ\\_V1 structure](#) 209
- [DRS\\_MSG\\_SPNREPLY\\_V1 structure](#) 248
- [DRS\\_MSG\\_SPNREQ\\_V1 structure](#) 247
- [DRS\\_MSG\\_UPDREFS\\_V1 structure](#) 241
- [DRS\\_OPTIONS](#) 275
- [DRS\\_SPN\\_CLASS](#) 277
- [DS\\_DOMAIN\\_CONTROLLER\\_INFO\\_1W structure](#) 108
- [DS\\_DOMAIN\\_CONTROLLER\\_INFO\\_2W structure](#) 109
- [DS\\_DOMAIN\\_CONTROLLER\\_INFO\\_3W structure](#) 110
- [DS\\_DOMAIN\\_CONTROLLER\\_INFO\\_FFFFFFFW structure](#) 111
- [DS\\_NAME\\_FORMAT enumeration](#) 83
- [DS\\_NAME\\_RESULT\\_ITEMW structure](#) 84
- [DS\\_NAME\\_RESULTW structure](#) 84
- [DS\\_REPL\\_ATTR\\_META\\_DATA structure](#) 138
- [DS\\_REPL\\_ATTR\\_META\\_DATA\\_2 structure](#) 139
- [DS\\_REPL\\_ATTR\\_VALUE\\_META\\_DATA structure](#) 142

[DS\\_REPL\\_ATTR\\_VALUE\\_META\\_DATA\\_2 structure](#) 143  
[DS\\_REPL\\_CLIENT\\_CONTEXT structure](#) 145  
[DS\\_REPL\\_CLIENT\\_CONTEXTS structure](#) 145  
[DS\\_REPL\\_CURSOR structure](#) 136  
[DS\\_REPL\\_CURSOR\\_2 structure](#) 136  
[DS\\_REPL\\_CURSOR\\_3W structure](#) 137  
[DS\\_REPL\\_CURSORS structure](#) 135  
[DS\\_REPL\\_CURSORS\\_2 structure](#) 136  
[DS\\_REPL\\_CURSORS\\_3W structure](#) 137  
[DS\\_REPL\\_KCC\\_DSA\\_FAILURESW structure](#) 140  
[DS\\_REPL\\_KCC\\_DSA\\_FAILUREW structure](#) 140  
[DS\\_REPL\\_NEIGHBORSW structure](#) 134  
[DS\\_REPL\\_NEIGHBORW structure](#) 134  
[DS\\_REPL\\_OBJ\\_META\\_DATA structure](#) 138  
[DS\\_REPL\\_OBJ\\_META\\_DATA\\_2 structure](#) 139  
[DS\\_REPL\\_OP\\_TYPE enumeration](#) 277  
[DS\\_REPL\\_OP\\_TYPE enumeration](#) 277  
[DS\\_REPL\\_OPW structure](#) 141  
[DS\\_REPL\\_PENDING\\_OPSW structure](#) 141  
[DS\\_REPL\\_SERVER\\_OUTGOING\\_CALL structure](#) 146  
[DS\\_REPL\\_SERVER\\_OUTGOING\\_CALLS structure](#) 146  
[DS\\_REPL\\_VALUE\\_META\\_DATA structure](#) 142  
[DS\\_REPL\\_VALUE\\_META\\_DATA\\_2 structure](#) 144  
[DSA\\_RPC\\_INST](#) 278  
[DSA\\_RPC\\_INST structure](#) 278  
[DSAObj](#) 278  
[DSName](#) ([section 5.34](#) 278, [section 5.35](#) 279)  
[DSNAME equality](#) 280  
[DSNAME structure](#) 279  
[DSTIME](#) 281  
[DWORD](#) 281

## E

### Examples

[common configuration example](#) 47  
[data display conventions](#) 48  
[IDL\\_DRSBind method example](#) 78  
[Expunge](#) 281

## F

[Fields - vendor-extensible](#) 32  
[FILETIME](#) 282  
[FindCharRev](#) 282  
[FOREST\\_TRUST\\_INFORMATION](#) 282  
[FOREST\\_TRUST\\_INFORMATION packet](#) 282  
[FOREST\\_TRUST\\_RECORD\\_TYPE](#) 293  
[FOREST\\_TRUST\\_RECORD\\_TYPE enumeration](#) 293  
[ForestRootDomainNC](#) 293  
[Full IDL](#) 332  
[FullReplicaExists](#) 293

## G

[GetAttrVals](#) 294  
[GetDefaultObjectCategory](#) 294  
[GetDSNameFromDN](#) 294  
[GetFSMORoleOwner](#) 294  
[GetInstanceNameFromSPN](#) 295

[GetObjectNC](#) 295  
[GetServiceClassFromSPN](#) 295  
[GetServiceNameFromSPN](#) 295  
[Glossary](#) 16  
[groupType bit flags](#) 295  
[GUID](#) 296  
[GuidFromString](#) 296  
[GuidToString](#) 297

## H

[handle\\_t](#) 297

## I

[IDL](#) 332  
[IDL\\_DRSAddSidHistory method](#) 52  
[IDL\\_DRSBind method](#) 71  
[IDL\\_DRSBind method example](#) 78  
[IDL\\_DRSCrackNames method](#) 80  
[IDL\\_DRSDomainControllerInfo method](#) 105  
[IDL\\_DRSExecuteKCC method](#) 121  
[IDL\\_DRSFinishDemotion method](#) 123  
[IDL\\_DRSGetReplInfo method](#) 129  
[IDL\\_DRSInitDemotion method](#) 181  
[IDL\\_DRSQuerySitesByCost method](#) 184  
[IDL\\_DRSRemoveDsDomain method](#) 204  
[IDL\\_DRSRemoveDsServer method](#) 208  
[IDL\\_DRSReplicaAdd method](#) 212  
[IDL\\_DRSReplicaDel method](#) 217  
[IDL\\_DRSReplicaDemotion method](#) 221  
[IDL\\_DRSReplicaModify method](#) 225  
[IDL\\_DRSReplicaSync method](#) 229  
[IDL\\_DRSReplicaVerifyObjects method](#) 231  
[IDL\\_DRSUnbind method](#) 239  
[IDL\\_DRSUpdateRefs method](#) 240  
[IDL\\_DRSWriteSPN method](#) 246  
[Implementer - security considerations](#) 331  
[Index of security parameters](#) 331  
[Informative references](#) 28  
[Initialization](#) 49  
[instanceType bit flags](#) 297  
[Introduction](#) 15  
[Is2PartSPN](#) 297  
[Is3PartSPN](#) 297  
[IsBuiltinPrincipal](#) 298  
[IsDCAccount](#) 298  
[IsDomainNameInTrustedForest](#) 298  
[IsGC](#) 299  
[IsGUIDBasedDNSName](#) 299  
[IsMemberOfBuiltinAdminGroup](#) 299  
[IsValidServiceName](#) 300

## K

[KCCFailedConnections](#) 300  
[KCCFailedLinks](#) 300

## L

Language constructs ([section 3.4.2](#) 40, [section 3.4.3](#) 40, [section 3.4.4](#) 42)

[LARGE\\_INTEGER](#) 300  
[LDAP\\_CONN\\_PROPERTIES](#) 300  
[LDAPConnections](#) 301  
[LinkStamp](#) 302  
[LinkValueStamp](#) 302  
[LONG](#) 303  
[LONGLONG](#) 303  
[LPWSTR](#) 304

## M

[MakeAttid](#) 304  
[MergeUTD](#) 304  
Messages  
    [overview](#) 33  
    [transport](#) 33  
Methods ([section 1.3.1](#) 28, [section 4](#) 50)  
[MTX\\_ADDR](#) 304  
[MTX\\_ADDR structure](#) 304

## N

[Naming conventions](#) 39  
[NetworkAddress](#) 305  
[NewPrefixTable](#) 305  
[Normative references](#) 27  
[NT4SID](#) 305  
[NT4SID structure](#) 305  
[NTDSTRANSPORT\\_OPT values](#) 305  
[NULLGUID](#) 306

## O

[Object attributes](#) 43  
[Object\(Access-Point\)](#) 260  
[Object\(DN-Binary\)](#) 260  
[Object\(DN-String\)](#) 260  
[Object\(DS-DN\)](#) 259  
[Object\(OR-Name\)](#) 261  
[ObjExists](#) 306  
[OID](#) 306  
[OID\\_t](#) 306  
[OID\\_t structure](#) 306  
[OidFromAttid](#) 306  
[Organization](#) 37  
[Overview](#) 37  
[Overview \(synopsis\)](#) 28

## P

[Parameters - security index](#) 331  
[parent](#) 306  
[PARTIAL\\_ATTR\\_VECTOR\\_V1\\_EXT](#) 307  
[PARTIAL\\_ATTR\\_VECTOR\\_V1\\_EXT structure](#) 307  
[partialAttributeSet](#) 307  
[PartialGCReplicaExists](#) 307  
[PAS\\_DATA](#) 307  
[PAS\\_DATA packet](#) 307  
[PDS\\_NAME\\_RESULT\\_ITEMW](#) 84  
[PDS\\_NAME\\_RESULTW](#) 84  
[PDSA\\_RPC\\_INST](#) 278  
[PerformReplication](#) 308

[Preconditions](#) 31  
[PrefixTable](#) 308  
[PrefixTableEntry](#) 308  
[PrefixTableEntry structure](#) 308  
Prerequisites ([section 1.5](#) 31, [section 3.3.1](#) 38)  
[Procedures](#) 253  
[Processing - asynchronous](#) 46  
[Product behavior](#) 348  
[Pseudocode](#) 39

## R

[RDN](#) 308  
[rdnType](#) 309  
[Record packet](#) 283  
References  
    [informative](#) 28  
    [normative](#) 27  
[Relationship to other protocols](#) 31  
[RemoveObj](#) 309  
[ReplicationQueue](#) 309  
[REPLTIMES](#) 310  
[REPLTIMES structure](#) 310  
[replUpToDateVector/ReplUpToDateVector](#) 311  
[REPS\\_FROM](#) 311  
[REPS\\_FROM packet](#) 311  
[REPS\\_TO](#) 314  
[REPS\\_TO packet](#) 314  
[repsFrom/RepsFrom](#) 317  
[repsTo/RepsTo](#) 318  
[Rid](#) 319  
[Right](#) 319  
[RIGHT values](#) 319  
[RPCClientContexts](#) 319  
[RPCOutgoingContexts](#) 320

## S

[sAMAccountType values](#) 320  
[SCHEMA\\_PREFIX\\_TABLE](#) 321  
[SCHEMA\\_PREFIX\\_TABLE structure](#) 321  
[SchemaNC](#) 321  
[SchemaObj](#) 321  
Security  
    [background](#) 33  
    [client-to-DC operations](#) 34  
    [DC-to-DC operations](#) 33  
    [implementer considerations](#) 331  
    [overview](#) 33  
    [parameter index](#) 331  
    [provider](#) 34  
    [service authentication](#) 33  
    [SPN for target DC in AD DS](#) 34  
    [SPN for target DC in AD LDS](#) 35  
[Sequencing issues](#) 29  
[Server extensions](#) 321  
[Server initialization](#) 49  
[Service authentication](#) 33  
[SID](#) 321  
[SidFromStringSid](#) 322  
[SPN for target DC in AD DS](#) 34  
[SPN for target DC in AD LDS](#) 35

[StampLessThanOrEqualUTD](#) 322  
[Standards assignments](#) 32  
[StartsWith](#) 322  
[State model](#) 38  
[String\(NT-Sec-Desc\)](#) 261  
[String\(Sid\)](#) 261  
[String\(Teletex\)](#) 261  
[StringSidFromSid](#) 322  
[SubString](#) 322  
[Syntax](#) 323  
[SYNTAX\\_ADDRESS packet](#) 323  
[SYNTAX\\_DISTNAME\\_BINARY packet](#) 323  
[systemFlags values](#) 325

## T

[Tracking changes](#) 355  
[Transactions](#) 38  
[Transport](#) 33  
Types ([section 1.3.3](#) 30, [section 3.3.3](#) 38)  
[Typographical conventions](#) 37

## U

[UCHAR](#) 325  
[ULONG](#) 325  
[ULONGLONG](#) 325  
[UPTODATE\\_CURSOR\\_V1 structure](#) 325  
[UPTODATE\\_CURSOR\\_V2 structure](#) 326  
[UPTODATE\\_VECTOR\\_V1\\_EXT structure](#) 326  
[UPTODATE\\_VECTOR\\_V2\\_EXT structure](#) 327  
[userAccountControl bits](#) 327  
[UserNameFromNT4AccountName](#) 328  
[USHORT](#) 328  
[USN](#) 328  
[USN\\_VECTOR structure](#) 328  
[UUID](#) 329

## V

[Value](#) 330  
[Values](#) 258  
[Variables](#) 253  
[Vendor-extensible fields](#) 32  
[Versioning](#) 31

## W

[WCHAR](#) 330