

# [MS-DNSP]: Domain Name Service (DNS) Server Management Protocol

---

## 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](mailto: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. For a list of Microsoft trademarks, visit [www.microsoft.com/trademarks](http://www.microsoft.com/trademarks).
- **Fictitious Names.** The example companies, organizations, products, domain names, email 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
01/25/2008	0.1	Major	MCPP Initial Availability.
03/14/2008	0.1.1	Editorial	Revised and edited the technical content.
05/16/2008	0.1.2	Editorial	Revised and edited the technical content.
06/20/2008	1.0	Major	Updated and revised the technical content.
07/25/2008	1.0.1	Editorial	Revised and edited the technical content.
08/29/2008	1.0.2	Editorial	Revised and edited the technical content.
10/24/2008	2.0	Major	Updated and revised the technical content.
12/05/2008	3.0	Major	Updated and revised the technical content.
01/16/2009	4.0	Major	Updated and revised the technical content.
02/27/2009	5.0	Major	Updated and revised the technical content.
04/10/2009	6.0	Major	Updated and revised the technical content.
05/22/2009	7.0	Major	Updated and revised the technical content.
07/02/2009	8.0	Major	Updated and revised the technical content.
08/14/2009	9.0	Major	Updated and revised the technical content.
09/25/2009	10.0	Major	Updated and revised the technical content.
11/06/2009	11.0	Major	Updated and revised the technical content.
12/18/2009	12.0	Major	Updated and revised the technical content.
01/29/2010	13.0	Major	Updated and revised the technical content.
03/12/2010	14.0	Major	Updated and revised the technical content.
04/23/2010	15.0	Major	Updated and revised the technical content.
06/04/2010	16.0	Major	Updated and revised the technical content.
07/16/2010	17.0	Major	Significantly changed the technical content.
08/27/2010	18.0	Major	Significantly changed the technical content.
10/08/2010	19.0	Major	Significantly changed the technical content.
11/19/2010	20.0	Major	Significantly changed the technical content.
01/07/2011	21.0	Major	Significantly changed the technical content.

<b>Date</b>	<b>Revision History</b>	<b>Revision Class</b>	<b>Comments</b>
02/11/2011	22.0	Major	Significantly changed the technical content.
03/25/2011	22.0	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	22.0	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	22.1	Minor	Clarified the meaning of the technical content.
09/23/2011	23.0	Major	Significantly changed the technical content.
12/16/2011	24.0	Major	Significantly changed the technical content.
03/30/2012	24.0	No change	No changes to the meaning, language, or formatting of the technical content.
07/12/2012	24.1	Minor	Clarified the meaning of the technical content.
10/25/2012	25.0	Major	Significantly changed the technical content.
01/31/2013	25.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	26.0	Major	Significantly changed the technical content.
11/14/2013	27.0	Major	Significantly changed the technical content.
02/13/2014	27.0	No change	No changes to the meaning, language, or formatting of the technical content.

# Contents

<b>1 Introduction</b>	<b>10</b>
1.1 Glossary	10
1.2 References	15
1.2.1 Normative References	15
1.2.2 Informative References	18
1.3 Overview	18
1.4 Relationship to Other Protocols	19
1.5 Prerequisites/Preconditions	21
1.6 Applicability Statement	22
1.7 Versioning and Capability Negotiation	22
1.8 Vendor-Extensible Fields	22
1.9 Standards Assignments	22
<b>2 Messages</b>	<b>23</b>
2.1 Transport	23
2.1.1 Server Security Settings	23
2.1.2 Client Security Settings	23
2.2 Common Data Types	24
2.2.1 DNS RPC Common Messages	24
2.2.1.1 Enumerations and Constants	24
2.2.1.1.1 DNS_RPC_TYPEID	24
2.2.1.1.2 DNS_RPC_PROTOCOLS	29
2.2.1.1.3 TRUSTPOINT_STATE	29
2.2.1.1.4 TRUSTANCHOR_STATE	30
2.2.1.2 Structures	30
2.2.1.2.1 DNS_RPC_CURRENT_CLIENT_VER	30
2.2.1.2.2 DNS_RPC_BUFFER	31
2.2.1.2.3 DNS_RPC_UTF8_STRING_LIST	31
2.2.1.2.4 DNS_RPC_UNICODE_STRING_LIST	32
2.2.1.2.5 DNS_RPC_NAME_AND_PARAM	32
2.2.1.2.6 DNS_SRV_RPC_UNION	32
2.2.2 Resource Record Messages	37
2.2.2.1 Enumerations and Constants	37
2.2.2.1.1 DNS_RECORD_TYPE	37
2.2.2.1.2 DNS_RPC_NODE_FLAGS	40
2.2.2.2 Structures	41
2.2.2.2.1 DNS_RPC_NAME	41
2.2.2.2.2 DNS_COUNT_NAME	41
2.2.2.2.3 DNS_RPC_NODE	42
2.2.2.2.4 DNS_RPC_RECORD_DATA	42
2.2.2.2.4.1 DNS_RPC_RECORD_A	43
2.2.2.2.4.2 DNS_RPC_RECORD_NODE_NAME	43
2.2.2.2.4.3 DNS_RPC_RECORD_SOA	44
2.2.2.2.4.4 DNS_RPC_RECORD_NULL	45
2.2.2.2.4.5 DNS_RPC_RECORD_WKS	45
2.2.2.2.4.6 DNS_RPC_RECORD_STRING	45
2.2.2.2.4.7 DNS_RPC_RECORD_MAIL_ERROR	46
2.2.2.2.4.8 DNS_RPC_RECORD_NAME_PREFERENCE	46
2.2.2.2.4.9 DNS_RPC_RECORD_SIG	47
2.2.2.2.4.10 DNS_RPC_RECORD_RRSIG	48

2.2.2.2.4.11	DNS_RPC_RECORD_NSEC.....	49
2.2.2.2.4.12	DNS_RPC_RECORD_DS.....	49
2.2.2.2.4.13	DNS_RPC_RECORD_KEY.....	50
2.2.2.2.4.14	DNS_RPC_RECORD_DHCID.....	50
2.2.2.2.4.15	DNS_RPC_RECORD_DNSKEY.....	50
2.2.2.2.4.16	DNS_RPC_RECORD_AAAA.....	51
2.2.2.2.4.17	DNS_RPC_RECORD_NXT.....	51
2.2.2.2.4.18	DNS_RPC_RECORD_SRV.....	52
2.2.2.2.4.19	DNS_RPC_RECORD_ATMA.....	52
2.2.2.2.4.20	DNS_RPC_RECORD_NAPTR.....	53
2.2.2.2.4.21	DNS_RPC_RECORD_WINS.....	54
2.2.2.2.4.22	DNS_RPC_RECORD_WINSR.....	54
2.2.2.2.4.23	DNS_RPC_RECORD_TS.....	55
2.2.2.2.4.24	DNS_RPC_RECORD_NSEC3.....	56
2.2.2.2.4.25	DNS_RPC_RECORD_NSEC3PARAM.....	56
2.2.2.2.5	DNS_RPC_RECORD.....	57
2.2.3	Address Information Messages.....	61
2.2.3.1	Enumerations and Constants.....	61
2.2.3.1.1	DNS_IPVAL_CONTEXT.....	61
2.2.3.1.2	DNS_IP_VALIDATE_RETURN_FLAGS.....	61
2.2.3.2	Structures.....	62
2.2.3.2.1	IP4_ARRAY.....	62
2.2.3.2.2	DNS_ADDR.....	62
2.2.3.2.2.1	DNS_ADDR.....	63
2.2.3.2.2.2	DNS_ADD_USER.....	63
2.2.3.2.3	DNS_ADDR_ARRAY.....	65
2.2.3.2.4	DNS_RPC_IP_VALIDATE.....	65
2.2.4	Server Messages.....	67
2.2.4.1	Enumerations and Constants.....	67
2.2.4.1.1	DNS_BOOT_METHODS.....	67
2.2.4.1.2	DNS_NAME_CHECK_FLAGS.....	67
2.2.4.2	Structures.....	68
2.2.4.2.1	DNSSRV_VERSION.....	68
2.2.4.2.2	DNS_RPC_SERVER_INFO.....	68
2.2.4.2.2.1	DNS_RPC_SERVER_INFO_W2K.....	68
2.2.4.2.2.2	DNS_RPC_SERVER_INFO_DOTNET.....	72
2.2.4.2.2.3	DNS_RPC_SERVER_INFO_LONGHORN.....	75
2.2.5	Zone Messages.....	76
2.2.5.1	Enumerations and Constants.....	76
2.2.5.1.1	DNS_ZONE_TYPE.....	76
2.2.5.1.2	DNS_ZONE_SECONDARY_SECURITY.....	76
2.2.5.1.3	DNS_ZONE_NOTIFY_LEVEL.....	77
2.2.5.1.4	ZONE_REQUEST_FILTERS.....	77
2.2.5.1.5	ZONE_SKD_ROLLOVER_TYPE.....	78
2.2.5.1.6	ZONE_SKD_ROLLOVER_ACTION.....	78
2.2.5.2	Structures.....	79
2.2.5.2.1	DNS_RPC_ZONE.....	79
2.2.5.2.1.1	DNS_RPC_ZONE_W2K.....	79
2.2.5.2.1.2	DNS_RPC_ZONE_DOTNET.....	80
2.2.5.2.2	DNS_RPC_ZONE_FLAGS.....	80
2.2.5.2.3	DNS_RPC_ZONE_LIST.....	82
2.2.5.2.3.1	DNS_RPC_ZONE_LIST_W2K.....	82
2.2.5.2.3.2	DNS_RPC_ZONE_LIST_DOTNET.....	82

2.2.5.2.4	DNS_RPC_ZONE_INFO.....	83
2.2.5.2.4.1	DNS_RPC_ZONE_INFO_W2K.....	83
2.2.5.2.4.2	DNS_RPC_ZONE_INFO_DOTNET.....	86
2.2.5.2.4.3	DNS_RPC_ZONE_INFO_LONGHORN.....	88
2.2.5.2.5	DNS_RPC_ZONE_SECONDARIES.....	89
2.2.5.2.5.1	DNS_RPC_ZONE_SECONDARIES_W2K.....	90
2.2.5.2.5.2	DNS_RPC_ZONE_SECONDARIES_DOTNET.....	90
2.2.5.2.5.3	DNS_RPC_ZONE_SECONDARIES_LONGHORN.....	91
2.2.5.2.6	DNS_RPC_ZONE_DATABASE.....	91
2.2.5.2.6.1	DNS_RPC_ZONE_DATABASE_W2K.....	91
2.2.5.2.6.2	DNS_RPC_ZONE_DATABASE_DOTNET.....	92
2.2.5.2.7	DNS_RPC_ZONE_CREATE_INFO.....	92
2.2.5.2.7.1	DNS_RPC_ZONE_CREATE_INFO_W2K.....	93
2.2.5.2.7.2	DNS_RPC_ZONE_CREATE_INFO_DOTNET.....	95
2.2.5.2.7.3	DNS_RPC_ZONE_CREATE_INFO_LONGHORN.....	96
2.2.5.2.8	DNS_RPC_ZONE_EXPORT_INFO.....	97
2.2.5.2.9	DNS_RPC_ENUM_ZONES_FILTER.....	98
2.2.5.2.10	DNS_RPC_FORWARDERS.....	98
2.2.5.2.10.1	DNS_RPC_FORWARDERS_W2K.....	99
2.2.5.2.10.2	DNS_RPC_FORWARDERS_DOTNET.....	99
2.2.5.2.10.3	DNS_RPC_FORWARDERS_LONGHORN.....	99
2.2.6	Zone Update Messages.....	100
2.2.6.1	Enumerations and Constants.....	100
2.2.6.1.1	DNS_ZONE_UPDATE.....	100
2.2.6.1.2	KeySignScope.....	100
2.2.6.1.3	ImportOpResult.....	101
2.2.6.2	Structures.....	101
2.2.6.2.1	DNS_RPC_SKD.....	101
2.2.6.2.2	DNS_RPC_SKD_LIST.....	103
2.2.6.2.3	DNS_RPC_SKD_STATE.....	103
2.2.6.2.4	DNS_RPC_TRUST_POINT.....	107
2.2.6.2.5	DNS_RPC_TRUST_POINT_LIST.....	108
2.2.6.2.6	DNS_RPC_TRUST_ANCHOR.....	108
2.2.6.2.7	DNS_RPC_TRUST_ANCHOR_LIST.....	110
2.2.6.2.8	DNS_RPC_SIGNING_VALIDATION_ERROR.....	111
2.2.6.2.9	DNS_RPC_ZONE_DNSSEC_SETTINGS.....	111
2.2.6.2.10	DNS_RPC_ZONE_SKD.....	113
2.2.6.2.11	DNS_RPC_SKD_STATE_EX.....	114
2.2.7	Application Directory Partition Messages.....	115
2.2.7.1	Enumerations and Constants.....	115
2.2.7.1.1	DNS_RPC_DP_FLAGS.....	115
2.2.7.2	Structures.....	116
2.2.7.2.1	DNS_RPC_DP_INFO.....	116
2.2.7.2.2	DNS_RPC_DP_REPLICA.....	118
2.2.7.2.3	DNS_RPC_DP_ENUM.....	118
2.2.7.2.4	DNS_RPC_DP_LIST.....	119
2.2.7.2.5	DNS_RPC_ENLIST_DP.....	119
2.2.7.2.6	DNS_RPC_ZONE_CHANGE_DP.....	120
2.2.8	AutoConfig Messages.....	121
2.2.8.1	Enumerations and Constants.....	121
2.2.8.1.1	DNS_RPC_AUTOCONFIG.....	121
2.2.8.2	Structures.....	123
2.2.8.2.1	DNS_RPC_AUTOCONFIGURE.....	123

2.2.9	Logging Messages .....	123
2.2.9.1	Enumerations and Constants .....	123
2.2.9.1.1	DNS_LOG_LEVELS .....	123
2.2.9.1.2	DNS_EVENTLOG_TYPES .....	125
2.2.10	Server Statistics Messages .....	125
2.2.10.1	Enumerations and Constants .....	125
2.2.10.1.1	DNSSRV_STATID_TYPES .....	125
2.2.10.2	Structures .....	127
2.2.10.2.1	DNSSRV_STAT_HEADER .....	127
2.2.10.2.2	DNSSRV_STATS .....	127
2.2.10.2.3	DNS_SYSTEMTIME .....	128
2.2.10.2.4	DNSSRV_TIME_STATS .....	128
2.2.10.2.5	DNSSRV_QUERY_STATS .....	130
2.2.10.2.6	DNSSRV_QUERY2_STATS .....	131
2.2.10.2.7	DNSSRV_RECURSE_STATS .....	133
2.2.10.2.8	DNSSRV_DNSSEC_STATS .....	141
2.2.10.2.9	DNSSRV_MASTER_STATS .....	141
2.2.10.2.10	DNSSRV_SECONDARY_STATS .....	144
2.2.10.2.11	DNSSRV_WINS_STATS .....	149
2.2.10.2.12	DNSSRV_UPDATE_STATS .....	149
2.2.10.2.13	DNSSRV_SKWANSEC_STATS .....	153
2.2.10.2.14	DNSSRV_DS_STATS .....	155
2.2.10.2.15	DNSSRV_MEMTAG_STATS .....	160
2.2.10.2.16	DNSSRV_MEMORY_STATS .....	161
2.2.10.2.17	DNSSRV_TIMEOUT_STATS .....	166
2.2.10.2.18	DNSSRV_DBASE_STATS .....	168
2.2.10.2.19	DNSSRV_RECORD_STATS .....	169
2.2.10.2.20	DNSSRV_PACKET_STATS .....	170
2.2.10.2.21	DNSSRV_NBSTAT_STATS .....	172
2.2.10.2.22	DNSSRV_PRIVATE_STATS .....	173
2.2.10.2.23	DNSSRV_ERROR_STATS .....	177
2.2.10.2.24	DNSSRV_CACHE_STATS .....	179
2.2.11	Key Structures .....	180
2.2.11.1	Enumerations and Constants .....	180
2.2.11.1.1	Cryptographic Algorithm Name .....	180
2.2.11.2	Structures .....	180
2.2.11.2.1	RSA Key Pair .....	180
2.2.11.2.2	ECDSA_P256 Key Pair .....	181
2.2.11.2.3	ECDSA_P384 Key Pair .....	183
2.2.11.2.4	Protection Key Identifier .....	184
2.2.11.2.5	Protection Key Attributes .....	186
2.2.11.2.6	Exported Key Pair .....	186
2.2.12	Zone Statistics Messages .....	187
2.2.12.1	Enumerations and Constants .....	187
2.2.12.1.1	DNS_ZONE_STATS_TYPE .....	188
2.2.12.2	Structures .....	189
2.2.12.2.1	DNSSRV_ZONE_TIME_STATS .....	189
2.2.12.2.2	DNSSRV_ZONE_QUERY_STATS .....	189
2.2.12.2.3	DNSSRV_ZONE_TRANSFER_STATS .....	190
2.2.12.2.4	DNSSRV_ZONE_UPDATE_STATS .....	190
2.2.12.2.5	DNS_RPC_ZONE_STATS_V1 .....	191
2.2.13	Zone Scope Messages .....	191
2.2.13.1	Enumerations and Constants .....	191

2.2.13.2	Structures.....	191
2.2.13.2.1	DNS_RPC_ENUM_ZONE_SCOPE_LIST .....	191
2.2.13.2.2	DNS_RPC_ZONE_SCOPE_CREATE_INFO.....	192
2.2.13.2.2.1	DNS_RPC_ZONE_SCOPE_CREATE_INFO_V1.....	192
2.2.13.2.3	DNS_RPC_ZONE_SCOPE_INFO.....	192
2.2.13.2.3.1	DNS_RPC_ZONE_SCOPE_INFO_V1 .....	192
2.3	Directory Service Schema Elements .....	193
2.3.1	Object Classes .....	194
2.3.1.1	msDNS-ServerSettings.....	194
2.3.2	Attributes.....	195
2.3.2.1	dnsProperty .....	195
2.3.2.1.1	Property Id.....	195
2.3.2.1.2	DcPromo Flag .....	197
2.3.2.2	dnsRecord .....	197
2.3.2.3	msDNS-IsSigned .....	198
2.3.2.4	msDNS-NSEC3OptOut.....	198
2.3.2.5	msDNS-SigningKeys .....	199
2.3.2.6	msDNS-SignWithNSEC3 .....	199
2.3.2.7	msDNS-NSEC3UserSalt .....	199
2.3.2.8	msDNS-DNSKEYRecords.....	199
2.3.2.9	msDNS-DSRecordSetTTL .....	199
2.3.2.10	msDNS-NSEC3Iterations.....	200
2.3.2.11	msDNS-PropagationTime .....	200
2.3.2.12	msDNS-NSEC3CurrentSalt .....	200
2.3.2.13	msDNS-RFC5011KeyRollovers .....	200
2.3.2.14	msDNS-NSEC3HashAlgorithm.....	200
2.3.2.15	msDNS-DSRecordAlgorithms .....	200
2.3.2.16	msDNS-DNSKEYRecordSetTTL.....	200
2.3.2.17	msDNS-MaintainTrustAnchor .....	200
2.3.2.18	msDNS-NSEC3RandomSaltLength.....	200
2.3.2.19	msDNS-SigningKeyDescriptors .....	200
2.3.2.20	msDNS-SignatureInceptionOffset.....	206
2.3.2.21	msDNS-ParentHasSecureDelegation .....	206
2.3.2.22	msDNS-SecureDelegationPollingPeriod .....	206
2.3.2.23	msDNS-KeymasterZones .....	206
<b>3</b>	<b>Protocol Details.....</b>	<b>207</b>
3.1	DnsServer Server Details.....	207
3.1.1	Abstract Data Model .....	207
3.1.1.1	DNS Server Configuration Information .....	211
3.1.1.1.1	DNS Server Integer Properties .....	211
3.1.1.1.2	DNS Server Address Array Properties .....	228
3.1.1.1.3	DNS Server String Properties.....	228
3.1.1.1.4	DNS Server String List Properties.....	229
3.1.1.2	DNS Zone Configuration Information.....	230
3.1.1.2.1	DNS Zone Integer Properties .....	230
3.1.1.2.2	DNS Zone Address Array Properties.....	233
3.1.1.2.3	DNS Zone String Properties .....	233
3.1.1.2.4	DNS Record Configuration Information.....	234
3.1.1.2.5	DNS Zone Scope Configuration Information.....	234
3.1.2	Timers .....	234
3.1.3	Initialization .....	234
3.1.4	Message Processing Events and Sequencing Rules.....	238



3.1.4.1	R_DnssrvOperation (Opnum 0).....	240
3.1.4.2	R_DnssrvQuery (Opnum 1).....	268
3.1.4.3	R_DnssrvComplexOperation (Opnum 2) .....	271
3.1.4.4	R_DnssrvEnumRecords (Opnum 3) .....	278
3.1.4.5	R_DnssrvUpdateRecord (Opnum 4).....	281
3.1.4.6	R_DnssrvOperation2 (Opnum 5).....	283
3.1.4.7	R_DnssrvQuery2 (Opnum 6) .....	284
3.1.4.8	R_DnssrvComplexOperation2 (Opnum 7).....	284
3.1.4.9	R_DnssrvEnumRecords2 (Opnum 8) .....	285
3.1.4.10	R_DnssrvUpdateRecord2 (Opnum 9) .....	286
3.1.4.11	R_DnssrvUpdateRecord3 (Opnum 10) .....	286
3.1.4.12	R_DnssrvEnumRecords3 (Opnum 11).....	287
3.1.4.13	R_DnssrvOperation3 (Opnum 12) .....	287
3.1.4.14	R_DnssrvQuery3 (Opnum 13) .....	288
3.1.5	Timer Events .....	289
3.1.6	Other Local Events .....	289
3.1.6.1	Three-phase authorization test.....	289
3.1.6.2	Directory server security descriptors reading and caching .....	290
3.1.6.3	dnsRecord in the Directory Server .....	290
3.1.6.4	Modifying Directory Server Security Descriptors .....	291
3.1.7	Key Processing Rules .....	291
3.1.7.1	Constructing an Exported Key .....	291
3.1.7.2	Importing an Exported Key .....	293
<b>4</b>	<b>Protocol Examples.....</b>	<b>295</b>
4.1	Querying a DNS server DWORD property .....	295
4.2	Modifying a DNS server DWORD property .....	295
4.3	Creating a New Zone.....	296
4.4	Enumerating Zones.....	296
4.5	Creating and Deleting a DNS Record .....	297
4.6	Creating a Zone Scope .....	298
4.7	Deleting a Zone Scope .....	299
4.8	Enumerating Zone Scopes .....	299
4.9	Creating and Deleting a DNS Record in a Zone Scope.....	300
<b>5</b>	<b>Security.....</b>	<b>301</b>
5.1	Security Considerations for Implementers.....	301
5.1.1	Security considerations specific to the DNS Server Management Protocol .....	301
5.2	Index of Security Parameters .....	301
<b>6</b>	<b>Appendix A: Full IDL.....</b>	<b>302</b>
<b>7</b>	<b>Appendix B: Product Behavior.....</b>	<b>337</b>
<b>8</b>	<b>Change Tracking.....</b>	<b>368</b>
<b>9</b>	<b>Index .....</b>	<b>369</b>

# 1 Introduction

The Domain Name Service (DNS) Server Management Protocol defines RPC interfaces that provide methods for remotely accessing and administering a DNS server. It is a client/server protocol based on RPC that is used in the configuration, management, and monitoring of a DNS server.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

## 1.1 Glossary

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

- Active Directory**
- Active Directory domain**
- application NC**
- ASCII**
- authentication level**
- Coordinated Universal Time (UTC)**
- crossRef object**
- distinguished name (DN)**
- domain controller (DC)**
- domain name (2), (3)**
- dynamic endpoint**
- forest**
- FSMO role**
- FSMO role owner**
- fully qualified domain name (FQDN) (1)**
- Interface Definition Language (IDL)**
- Internet Protocol version 4 (IPv4)**
- Internet Protocol version 6 (IPv6)**
- Lightweight Directory Access Protocol (LDAP)**
- naming context root**
- Network Data Representation (NDR)**
- opnum**
- read-only domain controller (RODC)**
- relative distinguished name (RDN) (1), (2)**
- remote procedure call (RPC)**
- root directory system agent-specific entry (rootDSE)**
- RPC protocol sequence**
- RPC transport**
- security descriptor**
- security provider**
- security support provider (SSP)**
- Security Support Provider Interface (SSPI)**
- Unicode**
- Unicode string**
- universally unique identifier (UUID)**
- UTF-8**
- UTF-16**

The following terms are specific to this document:

**Active Directory domain controller promotion (DCPROMO):** The act of causing a server to become a **domain controller (DC)**.

**Active Directory forest:** See **forest**.

**active refresh:** A self-generated DNS query for the DNSKEY records at a **trust point**, for the purpose of automatically retrieving new **trust anchors** and removing revoked **trust anchors**. See [\[RFC5011\]](#).

**aging:** A concept in which a DNS server keeps track of **time stamps** for the last update of individual resource records. Duration from last **time stamp** to current time is considered as the age of the resource-record and this value is used for scavenging, a process for cleaning out not-recently used records.

**application directory partition:** An **application NC**.

**authoritative:** A DNS server is **authoritative** for a portion of the DNS namespace if it hosts a primary or secondary **zone** for that portion of the DNS namespace.

**autocreated zone:** A **zone** that is created automatically by a DNS server, such as 0.in-addr.arpa, 127.in-addr.arpa or 255.in-addr.arpa.

**cache:** When DNS server receives information from other servers, it stores the information for a certain amount of time in its own in-memory zones, also referred to as a DNS cache. This improves performance of **domain name** resolution and reduces DNS-related query traffic. The cache contains only nodes that have unexpired records and expired but not-yet-freed records.

**delegation:** A name server (NS) record set in a parent **zone** that lists the name servers **authoritative** for a delegated subzone.

**directory server:** A persistent storage for DNS **zones** and records. A DNS server can access DNS data stored in a **directory server** using the **LDAP** protocol or a similar directory access mechanism.

**directory-server-integrated:** A DNS Server is **directory-server-integrated** if a **local directory server** such as **Active Directory** resides in the same machine as the DNS Server.

**directory server security descriptors:** The set of **security descriptors** read from the **directory server**, encompassing the DNS Server Configuration Access Control List, Zone Access Control List, and the Application Directory Partition Access Control List.

**DNS domain partition:** An **application directory partition** stored in the **directory server** that is replicated to all DNS servers in the domain.

**DNS forest partition:** An **application directory partition** stored in the **directory server** that is replicated to all DNS servers in the forest.

**dynamic update:** A mechanism defined in [\[RFC2136\]](#) by which updates for DNS records can be sent to the authoritative DNS server for a **zone** through the DNS protocol.

**expired DNS record:** A DNS record stored in the **cache** whose age is greater than the value of its **TTL**.

**forwarders:** A DNS server that is designated to facilitate forwarding of queries for other DNS servers.

**full zone transfer (AXFR):** A DNS protocol mechanism [\[RFC1035\]](#) through which an entire copy of a DNS **zone** can be transmitted to a remote DNS server.

**global name zone (GNZ):** A zone that provides single-label name resolution for large enterprise networks that do not deploy **WINS** and where using **domain name** suffixes to provide single-label name resolution is not practical.

**glue record:** A record of type A or AAAA included in a zone to specify the IP address of a DNS server used in a delegation. The fully qualified domain name of each glue record will match the fully qualified domain name of an authoritative DNS server found in one of the NS records in the delegation.

**incremental zone transfer (IXFR):** A DNS protocol mechanism [\[RFC1995\]](#) through which a partial copy of a DNS **zone** can be transmitted to a remote DNS server. An incremental **zone transfer**, or IXFR, is represented as a sequence of DNS record changes that can be applied to one image of a **zone** to synchronize it with another image of a **zone**.

**key master:** A DNS server that is responsible for generating and maintaining DNSSEC signing keys for one particular **zone**.

**key rollover:** The process through which DNSSEC signing keys are updated as described in [\[RFC4641\]](#) and [\[RFC5011\]](#).

**key signing key (KSK):** A DNSKEY used to sign only the DNSKEY record set at the root of the **zone**, as defined in [\[RFC4641\]](#).

**lame delegation:** A **delegation** in which none of the name servers listed in the **delegation** host the delegated subzone or respond to DNS queries.

**local directory server:** A **directory server** instance on the same host as the DNS server.

**multizone operation:** An operation requested to be performed on a set of **zones** with one or more particular properties, rather than on a single zone.

**multizone operation string:** A string indicating a property defining a set of **zones** on which an operation is to be performed.

**network mask:** A bit vector that, when logically AND-ed with an IP address, indicates the subnet to which an IP address belongs. Also known as net mask.

**node:** An entry identified by name in a DNS **zone**. A **node** contains all of the DNS records sets associated with the name.

**nonkey master primary server:** In a file-backed signed zone, a nonkey master primary is a server that holds the primary copy of the signed zone. A nonkey master primary server can also do signature refreshes and Zone Signing using Zone Signing Keys but cannot generate or manage keys on its own.

**NoRefresh interval:** If an update which does not change the DNS data for a record set is received within the **NoRefresh interval** then the DNS server will not update the timestamp on the record. This allows the DNS server to avoid unnecessary updates to the data store.

**notify:** DNS notify [\[RFC1996\]](#) is a mechanism in which the primary DNS server for a **zone** notifies secondary servers about any changes in the **zone**.

**online signing:** Refers to the process of signing and maintaining DNSSEC characteristics of a zone. [<1>](#)

**primary DNS server:** A DNS server that holds a master authoritative copy of a particular zone's data in local persistent storage.

**primary zone:** A zone for which a master authoritative copy of data is held in persistent local storage or in a locally accessible **directory server**. A zone stored in a **directory server** is a **primary zone** for any DNS server that can retrieve a copy of it from its **local directory server**.

**refresh interval:** If the **NoRefresh interval** for a record has expired and the DNS server receives a DNS update that does not change the record data then the DNS server will commit a new timestamp to the data store. The combination of NoRefresh and Refresh intervals allows a DNS server to maintain a relatively accurate record timestamp without unnecessary updates to the data store.

**resource record (RR):** A single piece of DNS data. Each **resource record** consists of a DNS type, a DNS class, a **time to live (TTL)**, and record data (RDATA) appropriate for the **resource record's** DNS type.

**read only domain controller (RODC):** A **directory server** that can be read from but not written to.

**root hints:** DNS **root hints** contain host information that is needed to resolve names outside of the **authoritative** DNS domains. It contains names and addresses of the root DNS servers.

**scavenging:** A regularly scheduled process on a DNS server during which DNS records that have not been updated within a certain interval may be deleted.

**secondary DNS server:** A DNS server that holds an authoritative read-only copy of a particular zone's data. The copy is periodically copied from another authoritative DNS server. Each zone can have any number of secondary DNS servers.

**secondary zone:** A zone for which an authoritative read-only copy of data is hosted by a particular DNS server. The data for a secondary zone is periodically copied from another DNS server that is authoritative for the zone.

**secret key transaction authentication (TSIG):** An authentication mechanism specified in [\[RFC2845\]](#) for DNS **dynamic updates** that uses a one-way hashing function to provide a cryptographically secure means of identifying each endpoint.

**secure delegation:** A delegation in a parent zone (name server (NS) record set), along with a signed delegation signer (DS) record set, signifying a delegation to a signed subzone.

**secure dynamic update:** A modification of the **dynamic update** mechanism defined in [\[RFC3645\]](#) by which updates for DNS records can be sent securely to the authoritative DNS server for a **zone** through the DNS protocol.

**security context:** The result of a TSIG [\[RFC2845\]](#) security negotiation between the server and a client machine.

**serial number:** A field in the **SOA** record [\[RFC1035\]](#) for a **zone**. This value is used to compare different versions of **zone**.

**signing key descriptor (SKD):** A collection of DNSSEC signing key characteristics such as algorithm, key length, and signature validity period that describe how DNSSEC signing keys and corresponding signatures should be generated and maintained by the DNS server.

**single-label name:** A **domain name** consisting of exactly one label [for example "contoso." (an absolute name) or "contoso" (a relative name)]. When written in dotted-notation [\[RFC1034\]](#), a **single-label name** will contain at most one period (".").

**start of authority (SOA):** Every **zone** contains a **SOA** record as defined in [\[RFC1035\]](#) section 3.3.13 and clarified in [\[RFC2181\]](#) section 7 at the beginning of the **zone** that provides information relevant for a **zone**.

**stub zone:** A specialized version of a secondary **zone**. A stub **zone** contains only those resource records that are necessary to identify the **authoritative** DNS server for that **zone**. A stub **zone** consists of the **zone** root **SOA resource record** [\[RFC1035\]](#) and [\[RFC2181\]](#), **zone** root NS resource records [\[RFC1035\]](#), and glue **resource records** for the **zone** root **SOA** and NS records.

**time stamp:** An integer value representing the number of hours that have elapsed since midnight (00:00:00), January 1, 1601 UTC.

**time to live (TTL):** Each DNS record has a **TTL** value [\[RFC1035\]](#) and [\[RFC2181\]](#) that specifies how long any DNS resolver (client or server) may hold the record in its **cache**.

**tombstone:** An inactive DNS node which is not considered to be part of a DNS **zone** but has not yet been deleted from the **zone** database in the **directory server**. **Tombstones** may be permanently deleted from the **zone** once they reach a certain age. **Tombstones** are not used for DNS **zones** that are not stored in the **directory server**. A node is a tombstone if its `dnsTombstoned` attribute has been set to "TRUE".

**trust anchor:** A DNSKEY (public key) or DS (public key hash) record that is presumed to be authentic (that is trusted); a DNSKEY or DS record that is in the "TrustAnchors" zone. A DS **trust anchor** cannot be used in a DNSSEC proof, but it can serve as an authentication of a retrieved DNSKEY record, allowing it to become a DNSKEY **trust anchor**.

**trust point:** An **FQDN** that has one or more **trust anchors**; a point in the DNS namespace from which a DNSSEC proof can begin, via the presumption of trust anchor authenticity; a node in the "TrustAnchors" zone that contains a DS or DNSKEY record.

**Windows Internet Name Service Reverse Lookup (WINS-R):** A form of reverse lookup performed by the DNS server using NBSTAT [\[RFC1002\]](#) lookups to map IPv4 addresses to single-label names.

**zone:** A domain namespace is divided up into several sections called zones [\[RFC1034\]](#) and [\[RFC2181\]](#). A **zone** represents authority over a portion of the DNS namespace, excluding any subzones that are below **delegations**.

**zone signing key (ZSK):** A DNSKEY used to sign all of the records in a zone, as defined in [\[RFC4641\]](#).

**zone transfer:** A DNS protocol mechanism [\[RFC1035\]](#) by which a full or partial copy of a DNS **zone** can be transmitted from one DNS server to another.

**zone scope:** A unique version of a **zone** that can be created inside an existing zone. Resource records can then be added (and subsequently managed) to the zone scope.

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

## 1.2 References

References to Microsoft Open Specifications documentation 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.

A reference marked "(Archived)" means that the reference document was either retired and is no longer being maintained or was replaced with a new document that provides current implementation details. We archive our documents online [[Windows Protocol](#)].

### 1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact [dochelp@microsoft.com](mailto:dochelp@microsoft.com). We will assist you in finding the relevant information.

[ATMA] Technical Committee, ATM Forum, "ATM Name System Specification Version 1.0", <http://www.ipmplsforum.org/ftp/pub/approved-specs/af-saa-0069.000.pdf>

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, <https://www2.opengroup.org/ogsys/catalog/c706>

[DRAFT-DNSSEC-ECDsa] Hoffman, P., and Wijngaards, W., "Elliptic Curve DSA for DNSSEC", draft-hoffman-dnssec-ecdsa-04, December 2010, <http://tools.ietf.org/html/draft-hoffman-dnssec-ecdsa-04.txt>

[IANA-DNS] IANA, "Domain Name System (DNS) Parameters", April 2009, <http://www.iana.org/assignments/dns-parameters>

[IANA-DNSSECAN] IANA, "Domain Name System Security (DNSSEC) Algorithm Numbers", <http://www.iana.org/assignments/dns-sec-alg-numbers>

[IANA-PROTO-NUM] IANA, "Protocol Numbers", February 2007, <http://www.iana.org/assignments/protocol-numbers>

[IANAPORT] IANA, "Port Numbers", November 2006, <http://www.iana.org/assignments/port-numbers>

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

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

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

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

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

[MS-GKDI] Microsoft Corporation, "[Group Key Distribution Protocol](#)".

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

- [MS-RPCE] Microsoft Corporation, "[Remote Procedure Call Protocol Extensions](#)".
- [RFC1034] Mockapetris, P., "Domain Names - Concepts and Facilities", STD 13, RFC 1034, November 1987, <http://www.ietf.org/rfc/rfc1034.txt>
- [RFC1035] Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1035, November 1987, <http://www.ietf.org/rfc/rfc1035.txt>
- [RFC1183] Everhart, C., Mamakos, L., Ullman, R. and Mockapetris, P., "New DNS RR Definitions", RFC 1183, October 1990, <http://www.ietf.org/rfc/rfc1183.txt>
- [RFC1738] Berners-Lee, T., Masinter, L., and McCahill, M., "Uniform Resource Locators (URL)", RFC 1738, December 1994, <http://www.ietf.org/rfc/rfc1738.txt>
- [RFC1876] Davis, C., Vixie, P., Goodwin, T., et al., "A Means for Expression Location Information in the Domain Name System", RFC 1876, January 1996, <http://www.ietf.org/rfc/rfc1876.txt>
- [RFC1982] Elz, R., and Bush, R., "Serial Number Arithmetic", RFC 1982, August 1996, <http://www.ietf.org/rfc/rfc1982.txt>
- [RFC1995] Ohta, M., "Incremental Zone Transfer in DNS", RFC 1995, August 1996, <http://www.ietf.org/rfc/rfc1995.txt>
- [RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, August 1996, <http://www.ietf.org/rfc/rfc1996.txt>
- [RFC2065] Eastlake, D., and Kaufman, C., "Domain Name System Security Extensions", RFC 2065, January 1997, <http://www.ietf.org/rfc/rfc2065.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>
- [RFC2136] Thomson, S., Rekhter Y. and Bound, J., "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997, <http://www.ietf.org/rfc/rfc2136.txt>
- [RFC2181] Elz, R., and Bush, R., "Clarifications to the DNS Specification", RFC 2181, July 1997, <http://www.ietf.org/rfc/rfc2181.txt>
- [RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, March 1998, <http://www.ietf.org/rfc/rfc2308.txt>
- [RFC2535] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999, <http://tools.ietf.org/html/rfc2535.txt>
- [RFC2671] Vixie, P., "Extension mechanism for DNS", RFC 2671, August 1999, <http://www.ietf.org/rfc/rfc2671.txt>
- [RFC2672] Crawford, M., and Fermilab, "Non-Terminal DNS Name Redirection", RFC 2672, August 1999, <http://www.ietf.org/rfc/rfc2672.txt>
- [RFC2782] Gulbrandsen, A., Vixie, P., and Esibov, L., "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000, <http://www.ietf.org/rfc/rfc2782.txt>
- [RFC2845] Vixie, P., Gudmundsson, O., Eastlake III, D., and Wellington, B., "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000, <http://www.ietf.org/rfc/rfc2845.txt>
- [RFC2915] Mealling, M., and Daniel, R., "The Naming Authority Pointer (NAPTR) DNS Resource Record", RFC 2915, September 2000, <http://www.ietf.org/rfc/rfc2915.txt>



- [RFC2930] Eastlake III, D., "Secret Key Establishment for DNS (TKEY RR)", RFC 2930, September 2000, <http://www.ietf.org/rfc/rfc2930.txt>
- [RFC2931] Eastlake, D., "DNS Request and Transaction Signature (SIG (0) s)", RFC 2931, September 2000, <http://www.ietf.org/rfc/rfc2931.txt>
- [RFC3403] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database", RFC 3403, October 2002, <http://www.ietf.org/rfc/rfc3403.txt>
- [RFC3447] Jonsson, J., and Kaliski, B., "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003, <http://www.ietf.org/rfc/rfc3447.txt>
- [RFC3565] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, July 2003, <http://www.ietf.org/rfc/rfc3565.txt>
- [RFC3596] Thomson, S., Huitema, C., Ksinant, V., and Souissi, M., "DNS Extensions to Support IP version 6", RFC 3596, October 2003, <http://www.ietf.org/rfc/rfc3596.txt>
- [RFC3629] Yergeau, F., "UTF-8, A Transformation Format of ISO 10646", STD 63, RFC 3629, November 2003, <http://www.ietf.org/rfc/rfc3629.txt>
- [RFC3645] Kwan, S., Garg, P., Gilroy, J., et al., "Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG)", RFC 3645, October 2003, <http://www.ietf.org/rfc/rfc3645.txt>
- [RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004, <http://www.ietf.org/rfc/rfc3852.txt>
- [RFC4033] Arends, R., Austein, R., Lason, M., et al., "DNS Security Introduction and Requirements", RFC 4033, March 2005, <http://www.ietf.org/rfc/rfc4033.txt>
- [RFC4034] Arends, R., Austein, R., Lason, M., et al., "Resource Records for the DNS Security Extensions", RFC 4034, March 2005, <http://www.ietf.org/rfc/rfc4034.txt>
- [RFC4035] Arends, R., Austein, R., Lason, M., et al., "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005, <http://www.ietf.org/rfc/rfc4035.txt>
- [RFC4511] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006, <http://www.rfc-editor.org/rfc/rfc4511.txt>
- [RFC4641] Kolkman, O., and Gieben, R., "DNSSEC Operational Practices", RFC 4641, September 2006, <http://www.ietf.org/rfc/rfc4641.txt>
- [RFC4701] Stapp, M., Lemon, T., and Gustafsson, A., "A DNS Resource Record (RR) for Encoding Dynamic Host Configuration Protocol (DHCP) Information (DHCID RR)", RFC 4701, October 2006, <http://www.ietf.org/rfc/rfc4701.txt>
- [RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", RFC 5011, September 2007, <http://www.ietf.org/rfc/rfc5011.txt>
- [RFC5084] Housley, R., "Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)", RFC 5084, November 2007, <http://www.ietf.org/rfc/rfc5084.txt>
- [RFC5155] Laurie, B., Sisson, G., Arends, R., and Blacka, D., "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, March 2008, <http://www.ietf.org/rfc/rfc5155.txt>

[UNICODE5.0.0/2007] The Unicode Consortium, "Unicode 5.0.0", 2007, <http://www.unicode.org/versions/Unicode5.0.0/>

## 1.2.2 Informative References

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

[MS-WINSRA] Microsoft Corporation, "[Windows Internet Naming Service \(WINS\) Replication and Autodiscovery Protocol](#)".

[MSDN-RPC] Microsoft Corporation, "Remote Procedure Call", <http://msdn.microsoft.com/en-us/library/aa378651.aspx>

[RFC1002] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications", STD 19, RFC 1002, March 1987, <http://www.ietf.org/rfc/rfc1002.txt>

## 1.3 Overview

The Domain Name Service (DNS) Server Management Protocol is a client/server protocol that is used to remotely query, monitor and configure DNS server settings, its **zones**, and **resource records**. The protocol allows a client to access DNS server settings and properties and also to enumerate all DNS data stored on the server (DNS zones and DNS records).

The DNS Server Management Protocol is a simple protocol with no state shared across **RPC** method calls. Each RPC method call contains one complete request. Output from one method call can be used as input to another call but the DNS Server Management Protocol does not provide for locking of DNS data across method calls. For example, a client may enumerate DNS zones with one call and then retrieve the properties of one or more of the enumerated DNS zones with another call, but no guarantee is made that the **zone** has not been deleted by another DNS Server Management Protocol client between these two method calls.

When the DNS server is **directory server-integrated**, some client requests may require or trigger **LDAP** requests from the DNS server to the **local directory server** or another directory server.

In particular, the DNS Server may use the "defaultNamingContext" of the directory server's **rootDSE**, a **DNS Domain Partition** named "DnsDomainZones", or a **DNS Forest Partition** named "DnsForestZones" to store zone information and zone records. (See section [2.3](#) for a discussion of the schemas used to store this information.) A DNS Server integrated with a directory server creates and automatically enlists itself in these default **Application Directory Partitions**. Alternatively, zone information and zone records may be stored in additional Application Directory Partitions, which can be created (and removed) by the DNS Server Management Protocol client in order to control the granularity of zone replication. Zones created in these additional Application Directory Partitions will only be visible to directory servers enlisted in those partitions, thus allowing for granular control over replication.

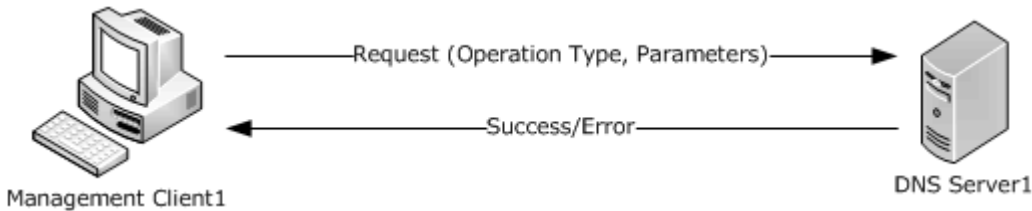
A typical remote management involves the client querying or setting the configuration parameters of the DNS server. The client may also enumerate DNS zones and the DNS records stored in one or more zones. The client can modify the configuration of the DNS server as required. The client can also add, delete, or modify DNS zones or the DNS records held in zones as required. For example, a remote management client can:

- Set or retrieve the server's **forwarders**.
- Set or retrieve various DNS server settings.

- Create or modify zones.
- Create or modify zone records.

This usually involves sending a request to the DNS server specifying the type of operation (get, set and execute are examples of types of operations) to perform and any specific parameters that are associated with that operation. The DNS server responds to the client with the result of the operation.

The following diagram shows an example of a remote client creating a zone on the DNS server using the DNS server Management Protocol. The client sends a request to the server with the operation type and parameters. The server responds with a success or an error.

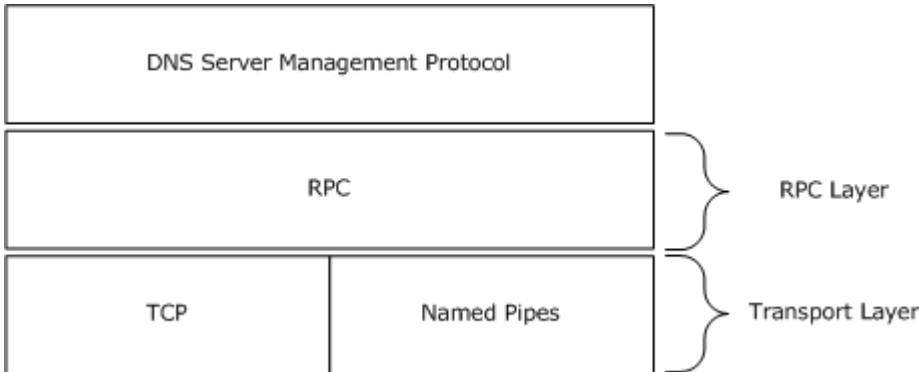


**Figure 1: DNS Server Management Protocol**

### 1.4 Relationship to Other Protocols

The Domain Name Service (DNS) Server Management Protocol relies on RPC [\[MS-RPCE\]](#) as a transport. It is used to manage servers that implement DNS [\[RFC1035\]](#), [\[RFC1183\]](#), [\[RFC1876\]](#), [\[RFC1995\]](#), [\[RFC1996\]](#), [\[RFC2065\]](#), [\[RFC2136\]](#), [\[RFC2535\]](#), [\[RFC2671\]](#), [\[RFC2672\]](#), [\[RFC2782\]](#), [\[RFC2845\]](#), [\[RFC2915\]](#), [\[RFC2931\]](#), [\[RFC3596\]](#), [\[RFC4034\]](#), and [\[RFC4701\]](#). It also interacts with the Netlogon protocol [\[MS-NRPC\]](#).

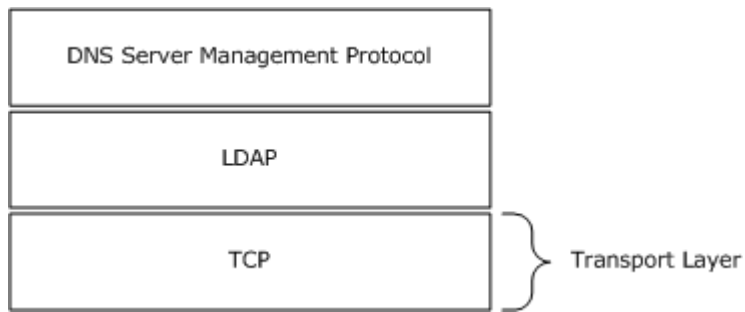
The following diagram illustrates the relationship of the DNS Server Management Protocol and how it relates to RPC [\[MS-RPCE\]](#).



**Figure 2: How the DNS Server Management Protocol uses RPC**

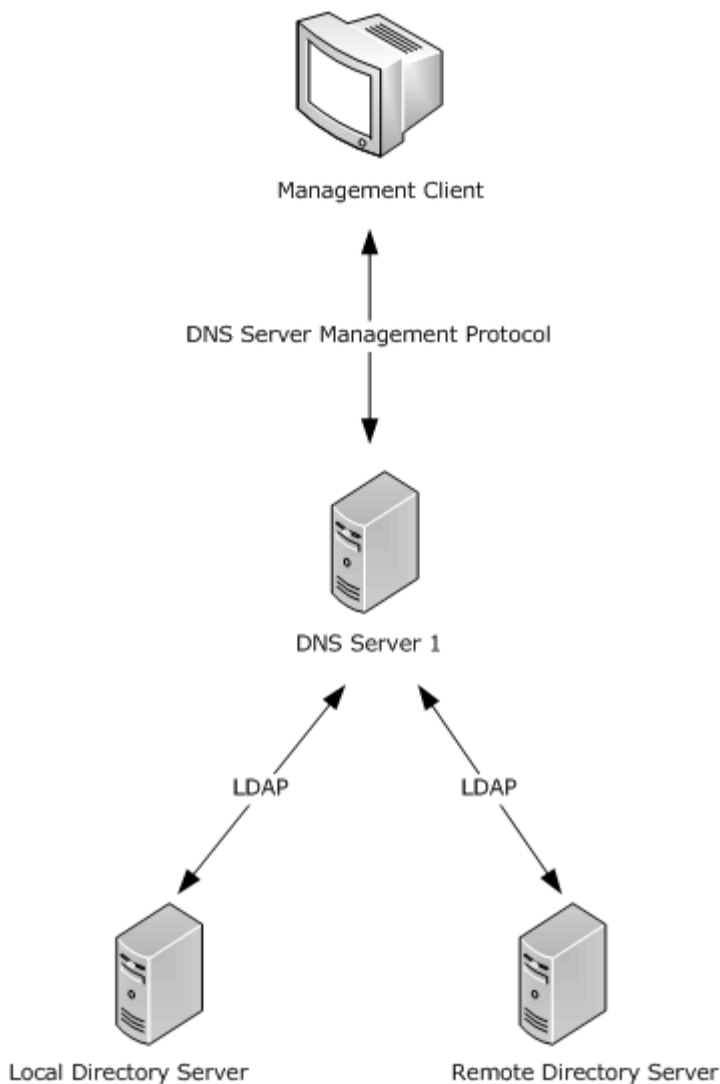
The DNS Server relies on the LDAP protocol [\[RFC4511\]](#) to retrieve and modify DNS information when it is stored in a directory server. In this case, the DNS Server is the client of the LDAP protocol, acting on behalf of and impersonating (using RPC impersonation), the client of the DNS Server Management Protocol. LDAP's relationship with other protocols is covered in [\[RFC4511\]](#) section 5. Note that although an LDAP provider may support transports other than TCP, the DNS Server is required to open LDAP connections specifying the TCP transport using port 389.

The following diagram shows the relationship of the DNS Server Management Protocol to LDAP and TCP:



**Figure 3: How the DNS Server Management Protocol uses LDAP**

The following diagram illustrates the interaction between DNS Server Management Protocol Clients, DNS Servers, and directory servers.



**Figure 4: Relationship between DNS Server Management Protocol Clients, DNS Servers, and Directory Servers**

## 1.5 Prerequisites/Preconditions

This protocol is implemented on top of RPC and, as a result, has the prerequisites identified in [\[MS-RPCE\]](#).

The Domain Name Service (DNS) Server Management Protocol assumes that a client has obtained the name of a server that supports this protocol before the protocol is invoked. It also assumes that if a local directory server is available, the DNS Server will establish an LDAP session to it, and has appropriate credentials for its requests. If no local directory server is available, or if an appropriate connection cannot be established, the DNS Server will operate only with zones loaded from files in the local file system rather than with zones replicated in the directory service, and operations requiring a directory service will fail. All LDAP operations described in this protocol are performed with the local directory server, unless otherwise stated. Consistency of DNS data stored in the local directory server is not guaranteed, since complete or partial updates to the LDAP directory may be

replicated to the local directory server at any time. The protocol assumes that the DNS server polls the local directory server for changes that should be synchronized with the in-memory zones.

## 1.6 Applicability Statement

This protocol is applicable when an application needs to remotely configure a DNS server. It is not applicable to scenarios with multiple clients simultaneously managing a DNS server, if the ability to prevent interference is required.

## 1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- Supported Transports: The DNS server Management Protocol uses the RPC protocol as a transport and multiple **RPC transports**.
- Protocol Versions: This protocol has a version number of 5.0.
- Security and Authentication Methods: Authentication and security are provided as specified in [\[MS-RPCE\]](#).

The DNS server requests the principal name for the **security provider** available on the system. Then Providers, for whom a principal name was obtained, are registered as supported authenticating mechanism for RPC calls. An RPC client using TCP, immediately after creating a binding attempts to negotiate authentication using `RPC_C_AUTHN_GSS_NEGOTIATE` and **authentication level** as `RPC_C_AUTHN_LEVEL_PKT_INTEGRITY` as specified in sections [2.1.1](#) and [2.1.2](#).

- Localization: This protocol passes text strings in various methods. Localization considerations for such strings are specified where relevant.
- Capability Negotiation: The DNS Server Management Protocol does not support negotiation of the protocol version to use. Instead, this protocol uses only the protocol version number specified in the **IDL** for versioning and capability negotiation. It should be noted that the present version of the IDL includes a client version input parameter (*dwClientVersion*) for some of the method calls (section [2.2.1.2.1](#)). This parameter allows the server to provide responses conforming to earlier versions of certain data structures associated with those method calls, while allowing extensibility of the present version of the protocol.

## 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 runs the risk of a collision in the future.

## 1.9 Standards Assignments

The following parameters are private Microsoft assignments.

Parameter	Value	Reference
RPC Interface <b>UUID</b> for DNS	50ABC2A4-574D-40B3-9D66-EE4FD5FBA076	<a href="#">[C706]</a> section A.2.5
Named Pipe name	\\PIPE\DNSSERVER	

## 2 Messages

The following sections specify how the DNS Server Management Protocol messages are transported and what common data types are used.

### 2.1 Transport

All implementations MUST support the RPC over TCP protocol sequence (ncacn\_ip\_tcp), as specified in [\[MS-RPCE\]](#), with **dynamic endpoints**. Implementations MAY also support the RPC over named pipes protocol sequence (ncacn\_np), as specified in [\[MS-RPCE\]](#), with named pipe name `\PIPE\DNSSERVER`. [<2>](#) The choice of transport for any given communication is up to the client application or higher-layer protocol.

The protocol MUST use the following UUID:

DnsServer: 50ABC2A4-574D-40B3-9D66-EE4FD5FBA076

The protocol MUST use an IDL version of 5.0.

#### 2.1.1 Server Security Settings

The DNS Server Management Protocol uses **Security Support Provider Interface (SSPI)** security provided by RPC, as specified in section [3.3.1.5.2](#) of [\[MS-RPCE\]](#), for sessions using TCP as the transport protocol. The server SHOULD register the following as security providers:

- `RPC_C_AUTHN_GSS_NEGOTIATE`
- `RPC_C_AUTHN_GSS_KERBEROS`
- `RPC_C_AUTHN_WINNT`

The DNS server MUST allow only authenticated access to RPC clients. The DNS server MUST NOT allow anonymous RPC clients. The DNS RPC server MUST perform a three-phase authorization test to ensure that the client is authorized to perform the specific RPC operation. The three-phase authorization test is specified in section [3.1.6.1](#). If the server is **directory server** integrated, the server MUST cache **directory server security descriptors** until the next LDAP read operation that reads them and perform LDAP read operations for security descriptors as specified in section [3.1.6.2](#).

The DNS server SHOULD support up to 1,234 concurrent RPC calls.

The DNS server MUST limit access to only clients that negotiate an authentication level higher than that of `RPC_C_AUTHN_LEVEL_NONE` (see [\[MS-RPCE\]](#) section 2.2.1.1.8).

#### 2.1.2 Client Security Settings

The DNS RPC client SHOULD use **SSP** security provided over RPC as specified in [\[MS-RPCE\]](#), for sessions using TCP as RPC transport protocol. A client SHOULD authenticate using:

- `RPC_C_AUTHN_GSS_NEGOTIATE`

A client using TCP as the RPC transport SHOULD request `RPC_C_AUTHN_LEVEL_PKT_INTEGRITY` authentication with the DNS server.

For negotiating RPC security, the DNS RPC client SHOULD use the following parameters:

- The client SHOULD request mutual authentication by requesting the `RPC_C_QOS_CAPABILITIES_MUTUAL_AUTH` capability. The client MAY additionally request the `RPC_C_QOS_CAPABILITIES_IGNORE_DELEGATE_FAILURE` capability. <3>
- The identity tracking type SHOULD be set to `RPC_C_QOS_IDENTITY_STATIC`.
- The impersonation type SHOULD be set to `RPC_C_IMP_LEVEL_IMPERSONATE`, indicating that the server can impersonate the client; the client MAY instead specify `RPC_C_IMP_LEVEL_DELEGATE`. <4>

## 2.2 Common Data Types

In addition to RPC base types and definitions specified in [C706] and [MS-RPCE], additional data types are defined below. This protocol also uses the types `WORD` and `DWORD` defined in [MS-DTYP].

All multi-byte integer values in the messages declared in this section use little-endian byte order unless otherwise noted.

All fields designated as Unicode strings MUST be implemented using UTF-16LE strings as defined in [UNICODE5.0.0/2007] section 2.6.

### 2.2.1 DNS RPC Common Messages

#### 2.2.1.1 Enumerations and Constants

##### 2.2.1.1.1 DNS\_RPC\_TYPEID

The DNS Server Management Protocol RPC methods use a generic and extensible data structure of type [DNSSRV\\_RPC\\_UNION \(section 2.2.1.2.6\)](#), which is a union of pointers to different data types. A `DNS_RPC_TYPEID` value is used to specify what data is being stored in an instance of the `DNSSRV_RPC_UNION` structure. The `DNS_RPC_TYPEID` enumeration combined with a `DNSSRV_RPC_UNION` structure allow the DNS RPC interface to communicate many different types of DNS server configuration and data in a single structure. The type of data carried inside the union is qualified by one of the following values:

Constant/value	Description
<code>DNSSRV_TYPEID_ANY</code> 0xFFFFFFFF	Type is invalid.
<code>DNSSRV_TYPEID_NULL</code> 0x00000000	No data is provided.
<code>DNSSRV_TYPEID_DWORD</code> 0x00000001	A <code>DWORD</code> value.
<code>DNSSRV_TYPEID_LPSTR</code> 0x00000002	A pointer to null-terminated <b>UTF-8</b> [RFC3629] string.
<code>DNSSRV_TYPEID_LPWSTR</code> 0x00000003	A pointer to a null-terminated <b>Unicode</b> string.
<code>DNSSRV_TYPEID_IPARRAY</code> 0x00000004	A pointer to an <b>IP4_ARRAY</b> . This structure is used to specify a list of <b>IPv4</b> addresses.



Constant/value	Description
DNSSRV_TYPEID_BUFFER 0x00000005	A pointer to a <a href="#">DNS RPC BUFFER (section 2.2.1.2.2)</a> . This structure is used to hold a generic buffer of the DNS server information. Interpretation of the buffer depends upon the request.
DNSSRV_TYPEID_SERVER_INFO_W2K 0x00000006	A pointer to a structure of type <a href="#">DNS RPC SERVER INFO W2K (section 2.2.4.2.2.1)</a> . This structure is used to specify general DNS server state and configuration.
DNSSRV_TYPEID_STATS 0x00000007	A pointer to a structure of type <a href="#">DNSSRV STATS (section 2.2.10.2.2)</a> . The structure exposes internal statistics and counters.
DNSSRV_TYPEID_FORWARDERS_W2K 0x00000008	A pointer to a structure of type <a href="#">DNS RPC FORWARDERS W2K (section 2.2.5.2.10.1)</a> . This structure specifies the set of DNS servers this DNS server will forward unresolved queries to.
DNSSRV_TYPEID_ZONE_W2K 0x00000009	A pointer to a structure of type <a href="#">DNS RPC ZONE W2K (section 2.2.5.2.1.1)</a> . This structure is used to specify basic information about a DNS zone.
DNSSRV_TYPEID_ZONE_INFO_W2K 0x0000000A	A pointer to a structure of type <a href="#">DNS RPC ZONE INFO W2K (section 2.2.5.2.4.1)</a> . This structure is used to specify detailed DNS zone information.
DNSSRV_TYPEID_ZONE_SECONDARIES_W2K 0x0000000B	A pointer to a structure of type <a href="#">DNS RPC ZONE SECONDARIES W2K (section 2.2.5.2.5.1)</a> . This structure is used to specify information about the secondary servers for a primary DNS zone.
DNSSRV_TYPEID_ZONE_DATABASE_W2K 0x0000000C	A pointer to a structure of type <a href="#">DNS RPC ZONE DATABASE W2K (section 2.2.5.2.6.1)</a> . This structure specifies how a DNS zone is stored in persistent storage.
DNSSRV_TYPEID_ZONE_TYPE_RESET_W2K 0x0000000D	This value is not used.
DNSSRV_TYPEID_ZONE_CREATE_W2K 0x0000000E	A pointer to a structure of type <a href="#">DNS RPC ZONE CREATE INFO W2K (section 2.2.5.2.7.1)</a> . This structure is used to specify parameters required when creating a new DNS zone.
DNSSRV_TYPEID_NAME_AND_PARAM 0x0000000F	A pointer to a structure of type <a href="#">DNS RPC NAME AND PARAM (section 2.2.1.2.5)</a> . This is a general purpose structure used to associate a parameter name with a DWORD value.
DNSSRV_TYPEID_ZONE_LIST_W2K 0x00000010	A pointer to a structure of type <a href="#">DNS RPC ZONE LIST W2K (section 2.2.5.2.3.1)</a> . This structure is used to enumerate

Constant/value	Description
	zones.
DNSSRV_TYPEID_ZONE_RENAME 0x00000011	This value is not used.
DNSSRV_TYPEID_ZONE_EXPORT 0x00000012	A pointer to a structure of type <a href="#">DNS RPC ZONE EXPORT INFO (section 2.2.5.2.8)</a> . This structure is used to specify how a zone should be exported to file.
DNSSRV_TYPEID_SERVER_INFO_DOTNET 0x00000013	A pointer to a structure of type <a href="#">DNS RPC SERVER INFO DOTNET (section 2.2.4.2.2.2)</a> . This structure is used to specify general DNS server state and configuration.
DNSSRV_TYPEID_FORWARDERS_DOTNET 0x00000014	A pointer to a structure of type <a href="#">DNS RPC FORWARDERS DOTNET (section 2.2.5.2.10.2)</a> . This structure specifies the set of DNS servers this DNS server will forward unresolved queries to.
DNSSRV_TYPEID_ZONE 0x00000015	A pointer to a structure of type <a href="#">DNS RPC ZONE (section 2.2.5.2.1)</a> . This structure is used to specify basic information and a DNS zone.
DNSSRV_TYPEID_ZONE_INFO_DOTNET 0x00000016	A pointer to a structure of type <a href="#">DNS RPC ZONE INFO DOTNET (section 2.2.5.2.4.2)</a> . This structure is used to specify detailed information about a DNS zone.
DNSSRV_TYPEID_ZONE_SECONDARIES_DOTNET 0x00000017	A pointer to a structure of type <a href="#">DNS RPC ZONE SECONDARIES DOTNET (section 2.2.5.2.5.2)</a> . This structure is used to specify information about the secondary servers for a primary DNS zone.
DNSSRV_TYPEID_ZONE_DATABASE 0x00000018	A pointer to a structure of type <a href="#">DNS RPC ZONE DATABASE (section 2.2.5.2.6)</a> . This structure specifies how a DNS zone is stored in persistent storage.
DNSSRV_TYPEID_ZONE_TYPE_RESET_DOTNET 0x00000019	This value is not used.
DNSSRV_TYPEID_ZONE_CREATE_DOTNET 0x0000001A	A pointer to a structure of type <a href="#">DNS RPC ZONE CREATE INFO DOTNET</a> . This structure is used to specify parameters required when creating a new DNS zone.
DNSSRV_TYPEID_ZONE_LIST 0x0000001B	A pointer to a structure of type <a href="#">DNS RPC ZONE LIST (section 2.2.5.2.3)</a> . This structure is used to enumerate zones.
DNSSRV_TYPEID_DP_ENUM 0x0000001C	A pointer to a structure of type <a href="#">DNS RPC DP ENUM (section 2.2.7.2.3)</a> . This structure is used to specify basic information about an application directory partition.

Constant/value	Description
DNSSRV_TYPEID_DP_INFO 0x0000001D	A pointer to a structure of type <a href="#">DNS RPC DP INFO (section 2.2.7.2.1)</a> . This structure specifies detailed information about a single application directory partition.
DNSSRV_TYPEID_DP_LIST 0x0000001E	A pointer to a structure of type <a href="#">DNS RPC DP LIST (section 2.2.7.2.4)</a> . This structure is used to enumerate application directory partitions.
DNSSRV_TYPEID_ENLIST_DP 0x0000001F	A pointer to a structure of type <a href="#">DNS RPC ENLIST DP (section 2.2.7.2.5)</a> . This structure is used to request enlistment changes for an application directory partition.
DNSSRV_TYPEID_ZONE_CHANGE_DP 0x00000020	A pointer to a structure of type <a href="#">DNS RPC_ZONE_CHANGE_DP (section 2.2.7.2.6)</a> . This structure is used to request that a DNS zone be moved from one application directory partition to another.
DNSSRV_TYPEID_ENUM_ZONES_FILTER 0x00000021	A pointer to a structure of type <a href="#">DNS RPC ENUM_ZONES_FILTER (section 2.2.5.2.9)</a> . This structure is used to filter DNS zones during enumeration.
DNSSRV_TYPEID_ADDRARRAY 0x00000022	A pointer to a structure of type <a href="#">DNS_ADDR_ARRAY (section 2.2.3.2.3)</a> . This structure is used to specify a list of IPv4 or <b>IPv6</b> addresses.
DNSSRV_TYPEID_SERVER_INFO 0x00000023	A pointer to a structure of type <a href="#">DNS RPC_SERVER_INFO (section 2.2.4.2.2)</a> . This structure is used to specify general DNS server state and configuration.
DNSSRV_TYPEID_ZONE_INFO 0x00000024	A pointer to a structure of type <a href="#">DNS RPC_ZONE_INFO (section 2.2.5.2.4)</a> . This structure is used to specify detailed information about a DNS zone.
DNSSRV_TYPEID_FORWARDERS 0x00000025	A pointer to a structure of type <a href="#">DNS RPC_FORWARDERS (section 2.2.5.2.10)</a> . This structure specifies the set of DNS servers this DNS server will forward unresolved queries to.
DNSSRV_TYPEID_ZONE_SECONDARIES 0x00000026	A pointer to a structure of type <a href="#">DNS RPC_ZONE_SECONDARIES (section 2.2.5.2.5)</a> . This structure is used to specify information about the secondary servers for a primary DNS zone.
DNSSRV_TYPEID_ZONE_TYPE_RESET 0x00000027	This value is not used.
DNSSRV_TYPEID_ZONE_CREATE 0x00000028	A pointer to a structure of type <a href="#">DNS RPC_ZONE_CREATE_INFO (section 2.2.5.2.7)</a> . This structure is used to specify parameters required when creating a new DNS zone.

Constant/value	Description
DNSSRV_TYPEID_IP_VALIDATE 0x00000029	A pointer to a structure of type <a href="#">DNS RPC IP_VALIDATE (section 2.2.3.2.4)</a> . This structure is used to request IP validation and to return the results of IP validation.
DNSSRV_TYPEID_AUTOCONFIGURE 0x0000002A	A pointer to a structure of type <a href="#">DNS RPC AUTOCONFIGURE (section 2.2.8.2.1)</a> . This structure is used to request DNS server autoconfiguration.
DNSSRV_TYPEID_UTF8_STRING_LIST 0x0000002B	A pointer to a structure of type <a href="#">DNS RPC UTF8 STRING LIST (section 2.2.1.2.3)</a> . This structure is used to represent a list of UTF-8 [RFC3629] strings.
DNSSRV_TYPEID_UNICODE_STRING_LIST 0x0000002C	A pointer to a structure of type <a href="#">DNS RPC UNICODE STRING LIST (section 2.2.1.2.4)</a> . This structure is used to represent a list of Unicode strings.
DNSSRV_TYPEID_SKD 0x0000002D	A pointer to a structure of type <a href="#">DNS RPC SKD (section 2.2.6.2.1)</a> . This structure is used to specify detailed signing key descriptor information.
DNSSRV_TYPEID_SKD_LIST 0x0000002E	A pointer to a structure of type <a href="#">DNS RPC SKD LIST (section 2.2.6.2.2)</a> . This structure is used to enumerate signing key descriptors.
DNSSRV_TYPEID_SKD_STATE 0x0000002F	A pointer to a structure of type <a href="#">DNS RPC SKD STATE (section 2.2.6.2.3)</a> . This structure is used to specify detailed signing key descriptor state information.
DNSSRV_TYPEID_SIGNING_VALIDATION_ERROR 0x00000030	A pointer to a structure of type <a href="#">DNS RPC SIGNING VALIDATION ERROR (section 2.2.6.2.8)</a> . This structure is used to specify signing key descriptor validation error information.
DNSSRV_TYPEID_TRUST_POINT_LIST 0x00000031	A pointer to a structure of type <a href="#">DNS RPC TRUST POINT LIST (section 2.2.6.2.5)</a> . This structure is used to enumerate <b>trust points</b> .
DNSSRV_TYPEID_TRUST_ANCHOR_LIST 0x00000032	A pointer to a structure of type <a href="#">DNS RPC TRUST ANCHOR LIST (section 2.2.6.2.7)</a> . This structure is used to enumerate <b>trust anchors</b> .
DNSSRV_TYPEID_ZONE_SIGNING_SETTINGS 0x00000033	A pointer to a structure of type <a href="#">DNS RPC ZONE DNSSEC SETTINGS (section 2.2.6.2.9)</a> . This structure is used to specify the DNSSEC settings for file-backed zones.
DNSSRV_TYPEID_ZONE_SCOPE_ENUM 0x00000034	A pointer to a structure of type <a href="#">DNS RPC ENUM ZONE SCOPE LIST (section 2.2.13.2.1)</a> . This structure is used to enumerate <b>zone scopes</b> .<5>

Constant/value	Description
DNS_SRV_TYPEID_ZONE_STATS 0x00000035	A pointer to a structure of type <a href="#">DNS_RPC_ZONE_STATS_V1 (section 2.2.12.2.5)</a> . This structure is used to enumerate the zone statistics.
DNS_SRV_TYPEID_ZONE_SCOPE_CREATE 0x00000036	A pointer to a structure of type <a href="#">DNS_RPC_ZONE_SCOPE_CREATE_INFO_V1 (section 2.2.13.2.2.1)</a> . This structure is used to create the zone scope.
DNS_SRV_TYPEID_ZONE_SCOPE_INFO 0x00000037	A pointer to a structure of type <a href="#">DNS_RPC_ZONE_SCOPE_INFO_V1 (section 2.2.13.2.3.1)</a> . This structure is used to get the zone scope information.

Clients and servers of the DNS Server Management Protocol SHOULD [<6>](#) support all values above.

### 2.2.1.1.2 DNS\_RPC\_PROTOCOLS

DNS\_RPC\_PROTOCOLS is a DWORD value that specifies types of RPC protocols supported by the DNS server. For more details about this type, see section 2.1 in [\[MS-RPCE\]](#).

Constant/value	Description
DNS_RPC_USE_TCPIP 0x00000001	The server allows clients to connect using RPC over TCP/IP.
DNS_RPC_USE_NAMED_PIPE 0x00000002	The server allows clients to connect using RPC over named pipes.
DNS_RPC_USE_LPC 0x00000004	An implementation-specific mode of communicating with a client on the same machine. <a href="#">&lt;7&gt;</a>
DNS_RPC_USE_ALL_PROTOCOLS 0xFFFFFFFF	The server allows clients to connect using any of the above RPC mechanisms.

### 2.2.1.1.3 TRUSTPOINT\_STATE

The TRUSTPOINT\_STATE enumeration identifies the current state of the life cycle of a DNSSEC trust point. The state of the trust point is determined by the states of its trust anchors and is given for informational purposes only; it does not affect the DNS or LDAP protocol.

Constant/value	Description
TRUSTPOINT_STATE_INITIALIZED 0x00000000	Indicates a trust point that has no trust anchors but has child nodes that do have trust anchors.
TRUSTPOINT_STATE_DSPENDING 0x00000001	Indicates a trust point that has only DS trust anchors and is therefore ineffective for DNSSEC operations.
TRUSTPOINT_STATE_ACTIVE 0x00000002	Indicates a trust point that has one or more DNSKEY trust anchors. Queries covered by this trust point will be validated using the DNSKEY trust anchors.

Constant/value	Description
TRUSTPOINT_STATE_DELETE_PENDING 0x00000003	Indicates a trust point containing only revoked DNSKEY trust anchors. When the <a href="#">[RFC5011]</a> remove-hold-down time is exceeded for all revoked trust anchors, this trust point will be deleted.

#### 2.2.1.1.4 TRUSTANCHOR\_STATE

The TRUSTANCHOR\_STATE enumeration identifies the current state of the life cycle of a DNSSEC trust anchor. All states other than TRUSTANCHOR\_STATE\_DSPENDING and TRUSTANCHOR\_STATE\_DSINVALID correspond to the states given in [\[RFC5011\]](#) section 4.2. Only the TRUSTANCHOR\_STATE\_VALID and TRUSTANCHOR\_STATE\_MISSING trust anchors affect the DNS protocol (by acting as a DNSSEC trust anchor). None of the states affect the LDAP protocol [\[RFC4511\]](#).

Constant/value	Description
TRUSTANCHOR_STATE_DSPENDING 0x00000001	Indicates a DS trust anchor that can be replaced by the DNSKEY trust anchor that matches the hash of the DS trust anchor after the next <a href="#">[RFC5011]</a> active refresh.
TRUSTANCHOR_STATE_DSINVALID 0x00000002	Indicates a DS trust anchor that matches a DNSKEY record that is not valid as a trust anchor because the SEP flag is not set or because the algorithm is unsupported.
TRUSTANCHOR_STATE_ADDPEND 0x00000003	Indicates a DNSKEY trust anchor that has not exceeded the <a href="#">[RFC5011]</a> add-hold-down time. This trust anchor may become a valid trust anchor after the hold-down time has been exceeded.
TRUSTANCHOR_STATE_VALID 0x00000004	Indicates a DNSKEY trust anchor that has exceeded the <a href="#">[RFC5011]</a> add-hold-down time, has been matched to (and has replaced) a DS trust anchor, or has been added by the administrator. This trust anchor is trusted for DNSSEC operations, and a copy exists in the TrustAnchors zone.
TRUSTANCHOR_STATE_MISSING 0x00000005	Indicates a DNSKEY trust anchor that is valid but was not seen in the last <a href="#">[RFC5011]</a> active refresh. This trust anchor is trusted for DNSSEC operations, and a copy exists in the TrustAnchors zone.
TRUSTANCHOR_STATE_REVOKED 0x00000006	Indicates a DNSKEY trust anchor that has been revoked by the authoritative zone administrator according to <a href="#">[RFC5011]</a> . This trust anchor will be deleted when the <a href="#">[RFC5011]</a> remove-hold-down time is exceeded.

#### 2.2.1.2 Structures

##### 2.2.1.2.1 DNS\_RPC\_CURRENT\_CLIENT\_VER

This structure specifies version number information of the DNS RPC client. This version number is used by RPC clients to identify the requested RPC structures' version to the server so that the server can treat the request appropriately. This structure is defined as (in host 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
DNS RPC Client Major Version																DNS RPC Client Minor Version															

**DNS RPC Client Major Version (2 bytes):** The major version number for the DNS RPC client. This MUST have a value from the following set:

Value	Meaning
0x0000	Client requests RPC structures that do not require specific version numbers.
0x0006	Client requests RPC structures associated with version 6.
0x0007	Client requests RPC structures associated with version 7.

**DNS RPC Client Minor Version (2 bytes):** The minor version number for the DNS RPC client. Senders MUST set this to zero and receivers MUST ignore it.

### 2.2.1.2.2 DNS\_RPC\_BUFFER

DNS\_RPC\_BUFFER defines a structure that contains a set of a specific type of structures. The DNS Server Management Protocol uses this structure to return information from the server, while processing R\_DnssrvComplexOperation2 (section [3.1.4.8](#)) method calls with operation type "Statistics".

```
typedef struct _DnssrvRpcBuffer {
    DWORD dwLength;
    [size_is(dwLength)] BYTE Buffer[];
} DNS_RPC_BUFFER,
*PDNS_RPC_BUFFER;
```

**dwLength:** The length, in bytes, of the data stored in Buffer.

**Buffer:** A variable length array of bytes of length specified by dwLength. The buffer can contain one or more DNS\_RPC\_NODE structures (section [2.2.2.3](#)). Each DNS\_RPC\_NODE contains the length of that node such that the DNS\_RPC\_BUFFER dwLength may be larger to indicate multiple DNS\_RPC\_NODE structures.

### 2.2.1.2.3 DNS\_RPC\_UTF8\_STRING\_LIST

DNS\_RPC\_UTF8\_STRING\_LIST defines a structure that contains a list of null-terminated UTF-8 strings. This structure is used by the DNS server management protocol while processing R\_DnssrvOperations2 (section [3.1.4.6](#)) and R\_DnssrvQuery2 (section [3.1.4.7](#)) method calls, with operations type "GlobalQueryBlockList".

```
typedef struct _DnsRpcUtf8StringList {
    [range(0,10000)] DWORD dwCount;
    [size_is(dwCount), string] char* pszStrings[];
} DNS_RPC_UTF8_STRING_LIST,
*PDNS_RPC_UTF8_STRING_LIST;
```

**dwCount:** The number of strings present in the pszStrings array.

**pszStrings:** A variable length array of pointers to null-terminated UTF-8 strings.

#### 2.2.1.2.4 DNS\_RPC\_UNICODE\_STRING\_LIST

**DNS\_RPC\_UNICODE\_STRING\_LIST** defines a structure that contains a list of null-terminated **Unicode strings**. This structure is used by the DNS server management protocol while processing the [R\\_DnssrvComplexOperation2 \(section 3.1.4.8\)](#) method call, with operations of type "EnumerateKeyStorageProviders".

```
typedef struct _DnsRpcUnicodeStringList {
    [range(0,10000)] DWORD dwCount;
    [size_is(dwCount), string] char* ppszStrings[];
} DNS_RPC_UNICODE_STRING_LIST,
*PDNS_RPC_UNICODE_STRING_LIST;
```

**dwCount:** The number of strings present in the ppszStrings array.

**ppszStrings:** A variable-length array of pointers to null-terminated Unicode strings.

#### 2.2.1.2.5 DNS\_RPC\_NAME\_AND\_PARAM

**DNS\_RPC\_NAME\_AND\_PARAM** defines the structure that contains information about a simple server property that takes a DWORD value. The DNS Server Management Protocol uses this structure to exchange information about various properties that take an integer value, while processing the [R\\_DnssrvOperation2 \(section 3.1.4.6\)](#) method call with operation types "ResetDwordProperty", "DeleteNode" and "DeleteRecordSet".

```
typedef struct _DnssrvRpcNameAndParam {
    DWORD dwParam;
    [string] char* pszNodeName;
} DNS_RPC_NAME_AND_PARAM,
*PDNS_RPC_NAME_AND_PARAM;
```

**dwParam:** The requested new value for the server property specified by pszNodeName.

**pszNodeName:** Pointer to a null-terminated UTF-8 string that specifies the name of the server property.

#### 2.2.1.2.6 DNSSRV\_RPC\_UNION

**DNSSRV\_RPC\_UNION** specifies a collection of all possible messages structures that can be exchanged between a client and server communicating using the DNS Server Management Protocol. This is used by the [R\\_DnssrvOperation2 \(section 3.1.4.6\)](#), [R\\_DnssrvQuery2 \(section 3.1.4.7\)](#) and [R\\_DnssrvComplexOperation2 \(section 3.1.4.8\)](#) method calls. The exact message format inside **DNSSRV\_RPC\_UNION** is identified by an accompanying **DNS\_RPC\_TYPEID (section 2.2.1.1.1)** value. Clients and servers of the DNS Server Management Protocol SHOULD support all members of **DNSSRV\_RPC\_UNION**.<8>



```

typedef
[switch_type(DWORD)]
union _DnssrvSrvRpcUnion {
[case(DNSSRV_TYPEID_NULL)]
    PBYTE Null;
[case(DNSSRV_TYPEID_DWORD)]
    DWORD Dword;
[case(DNSSRV_TYPEID_LPSTR)]
    [string] char* String;
[case(DNSSRV_TYPEID_LPWSTR)]
    [string] wchar_t* WideString;
[case(DNSSRV_TYPEID_IPARRAY)]
    PIP4_ARRAY IpArray;
[case(DNSSRV_TYPEID_BUFFER)]
    PDNS_RPC_BUFFER Buffer;
[case(DNSSRV_TYPEID_SERVER_INFO_W2K)]
    PDNS_RPC_SERVER_INFO_W2K ServerInfoW2K;
[case(DNSSRV_TYPEID_STATS)]
    PDNSSRV_STATS Stats;
[case(DNSSRV_TYPEID_FORWARDERS_W2K)]
    PDNS_RPC_FORWARDERS_W2K ForwardersW2K;
[case(DNSSRV_TYPEID_ZONE_W2K)]
    PDNS_RPC_ZONE_W2K ZoneW2K;
[case(DNSSRV_TYPEID_ZONE_INFO_W2K)]
    PDNS_RPC_ZONE_INFO_W2K ZoneInfoW2K;
[case(DNSSRV_TYPEID_ZONE_SECONDARIES_W2K)]
    PDNS_RPC_ZONE_SECONDARIES_W2K SecondariesW2K;
[case(DNSSRV_TYPEID_ZONE_DATABASE_W2K)]
    PDNS_RPC_ZONE_DATABASE_W2K DatabaseW2K;
[case(DNSSRV_TYPEID_ZONE_CREATE_W2K)]
    PDNS_RPC_ZONE_CREATE_INFO_W2K ZoneCreateW2K;
[case(DNSSRV_TYPEID_NAME_AND_PARAM)]
    PDNS_RPC_NAME_AND_PARAM NameAndParam;
[case(DNSSRV_TYPEID_ZONE_LIST_W2K)]
    PDNS_RPC_ZONE_LIST_W2K ZoneListW2K;
[case(DNSSRV_TYPEID_SERVER_INFO_DOTNET)]
    PDNS_RPC_SERVER_INFO_DOTNET ServerInfoDotNet;
[case(DNSSRV_TYPEID_FORWARDERS_DOTNET)]
    PDNS_RPC_FORWARDERS_DOTNET ForwardersDotNet;
[case(DNSSRV_TYPEID_ZONE)]
    PDNS_RPC_ZONE Zone;
[case(DNSSRV_TYPEID_ZONE_INFO_DOTNET)]
    PDNS_RPC_ZONE_INFO_DOTNET ZoneInfoDotNet;
[case(DNSSRV_TYPEID_ZONE_SECONDARIES_DOTNET)]
    PDNS_RPC_ZONE_SECONDARIES_DOTNET SecondariesDotNet;
[case(DNSSRV_TYPEID_ZONE_DATABASE)]
    PDNS_RPC_ZONE_DATABASE Database;
[case(DNSSRV_TYPEID_ZONE_CREATE_DOTNET)]
    PDNS_RPC_ZONE_CREATE_INFO_DOTNET ZoneCreateDotNet;
[case(DNSSRV_TYPEID_ZONE_LIST)]
    PDNS_RPC_ZONE_LIST ZoneList;
[case(DNSSRV_TYPEID_ZONE_EXPORT)]
    PDNS_RPC_ZONE_EXPORT_INFO ZoneExport;
[case(DNSSRV_TYPEID_DP_INFO)]
    PDNS_RPC_DP_INFO DirectoryPartition;
[case(DNSSRV_TYPEID_DP_ENUM)]
    PDNS_RPC_DP_ENUM DirectoryPartitionEnum;
[case(DNSSRV_TYPEID_DP_LIST)]
    PDNS_RPC_DP_LIST DirectoryPartitionList;

```

```

[case(DNSSRV_TYPEID_ENLIST_DP)]
    PDNS_RPC_ENLIST_DP EnlistDirectoryPartition;
[case(DNSSRV_TYPEID_ZONE_CHANGE_DP)]
    PDNS_RPC_ZONE_CHANGE_DP ZoneChangeDirectoryPartition;
[case(DNSSRV_TYPEID_ENUM_ZONES_FILTER)]
    PDNS_RPC_ENUM_ZONES_FILTER EnumZonesFilter;
[case(DNSSRV_TYPEID_ADDRARRAY)]
    PDNS_ADDR_ARRAY AddrArray;
[case(DNSSRV_TYPEID_SERVER_INFO)]
    PDNS_RPC_SERVER_INFO ServerInfo;
[case(DNSSRV_TYPEID_ZONE_CREATE)]
    PDNS_RPC_ZONE_CREATE_INFO ZoneCreate;
[case(DNSSRV_TYPEID_FORWARDERS)]
    PDNS_RPC_FORWARDERS Forwarders;
[case(DNSSRV_TYPEID_ZONE_SECONDARIES)]
    PDNS_RPC_ZONE_SECONDARIES Secondaries;
[case(DNSSRV_TYPEID_IP_VALIDATE)]
    PDNS_RPC_IP_VALIDATE IpValidate;
[case(DNSSRV_TYPEID_ZONE_INFO)]
    PDNS_RPC_ZONE_INFO ZoneInfo;
[case(DNSSRV_TYPEID_AUTOCONFIGURE)]
    PDNS_RPC_AUTOCONFIGURE AutoConfigure;
[case(DNSSRV_TYPEID_UTF8_STRING_LIST)]
    PDNS_RPC_UTF8_STRING_LIST Utf8StringList;
[case(DNSSRV_TYPEID_UNICODE_STRING_LIST)]
    PDNS_RPC_UNICODE_STRING_LIST UnicodeStringList;
[case(DNSSRV_TYPEID_SKD)]
    PDNS_RPC_SKD Skd;
[case(DNSSRV_TYPEID_SKD_LIST)]
    PDNS_RPC_SKD_LIST SkdList;
[case(DNSSRV_TYPEID_SKD_STATE)]
    PDNS_RPC_SKD_STATE SkdState;
[case(DNSSRV_TYPEID_SIGNING_VALIDATION_ERROR)]
    PDNS_RPC_SIGNING_VALIDATION_ERROR SigningValidationError;
[case(DNSSRV_TYPEID_TRUST_POINT_LIST)]
    PDNS_RPC_TRUST_POINT_LIST TrustPointList;
[case(DNSSRV_TYPEID_TRUST_ANCHOR_LIST)]
    PDNS_RPC_TRUST_ANCHOR_LIST TrustAnchorList;
[case(DNSSRV_TYPEID_ZONE_SIGNING_SETTINGS)]
    PDNS_RPC_ZONE_DNSSEC_SETTINGS ZoneDnsSecSettings;
[case(DNSSRV_TYPEID_ZONE_SCOPE_ENUM)]
    PDNS_RPC_ENUM_ZONE_SCOPE_LIST ZoneScopeList;
[case(DNSSRV_TYPEID_ZONE_STATS)]
    PDNS_RPC_ZONE_STATS ZoneStats;
[case(DNSSRV_TYPEID_ZONE_SCOPE_CREATE)]
    PDNS_RPC_ZONE_SCOPE_CREATE_INFO ScopeCreate;
[case(DNSSRV_TYPEID_ZONE_SCOPE_INFO)]
    PDNS_RPC_ZONE_SCOPE_INFO ScopeInfo;
} DNSSRV_RPC_UNION;

```

**Null:** No data is provided.

**Dword:** Data is a DWORD value.

**String:** A pointer to a null-terminated UTF-8 string or a NULL pointer.

**WideString:** A pointer to a null-terminated Unicode string or a NULL pointer.

**IpArray:** An array of IPv4 addresses in IP4\_ARRAY (section [2.2.3.2.1](#)) format.

**Buffer:** A pointer to a DNS\_RPC\_BUFFER (section [2.2.1.2.2](#)).

**ServerInfoW2K:** A pointer to a structure of type DNS\_RPC\_SERVER\_INFO\_W2K (section [2.2.4.2.2](#)). This structure is used to specify general DNS server state and configuration.

**Stats:** A pointer to a structure of type DNSSRV\_STAT (section [2.2.10.2.2](#)). The structure exposes internal statistics and counters.

**ForwardersW2K:** A pointer to a structure of type DNS\_RPC\_FORWARDERS\_W2K (section [2.2.5.2.10.1](#)). This structure specifies the set of DNS servers this DNS server will forward unresolved queries to.

**ZoneW2K:** A pointer to a structure of type DNS\_RPC\_ZONE\_W2K (section [2.2.5.2.1](#)). This structure is used to specify basic information about a DNS zone.

**ZoneInfoW2K:** A pointer to a structure of type DNS\_RPC\_INFO\_W2K (section [2.2.5.2.4](#)). This structure is used to specify detailed DNS zone information.

**SecondariesW2K:** A pointer to a structure of type DNS\_RPC\_ZONE\_SECONDARIES\_W2K (section [2.2.5.2.5](#)). This structure is used to specify information about the secondary servers for a primary DNS zone.

**DatabaseW2K:** A pointer to a structure of type DNS\_RPC\_ZONE\_DATABASE\_W2K (section [2.2.5.2.6](#)). This structure specifies how a DNS zone is stored in persistent storage.

**ZoneCreateW2K:** A pointer to a structure of type DNS\_RPC\_ZONE\_CREATE\_INFO\_W2K (section [2.2.5.2.9](#)). This structure is used to specify parameters required when creating a new DNS zone.

**NameAndParam:** A pointer to a structure of type DNS\_RPC\_NAME\_AND\_PARAM (section [2.2.1.2.5](#)). This is a general purpose structure used to associate a parameter name with a DWORD value.

**ZoneListW2K:** A pointer to a structure of type DNS\_RPC\_ZONE\_LIST\_W2K (section [2.2.1.2.5](#)). This structure is used to enumerate zones.

**ServerInfoDotNet:** A pointer to a structure of type DNS\_RPC\_SERVER\_INFO\_DOTNET (section [2.2.4.2.2](#)). This structure is used to specify general DNS server state and configuration.

**ForwardersDotNet:** A pointer to a structure of type DNS\_RPC\_FORWARDERS\_DOTNET (section [2.2.5.2.10.2](#)). This structure specifies the set of DNS servers this DNS server will forward unresolved queries to.

**Zone:** A pointer to a structure of type DNS\_RPC\_ZONE (section [2.2.5.2.1](#)). This structure is used to specify basic information about a DNS zone.

**ZoneInfoDotNet:** A pointer to a structure of type DNS\_RPC\_ZONE\_INFO\_DOTNET (section [2.2.5.2.4](#)). This structure is used to specify detailed DNS zone information.

**SecondariesDotNet:** A pointer to a structure of type DNS\_RPC\_ZONE\_SECONDARIES\_DOTNET (section [2.2.5.2.5](#)). This structure is used to specify information about the secondary servers for a primary DNS zone.

**Database:** A pointer to a structure of type DNS\_RPC\_ZONE\_DATABASE (section [2.2.5.2.6](#)). This structure specifies how a DNS zone is stored in persistent storage.

**ZoneCreateDotNet:** A pointer to a structure of type `DNS_RPC_ZONE_CREATE_INFO_DOTNET` (section [2.2.5.2.9](#)). This structure is used to specify parameters required when creating a new DNS zone.

**ZoneList:** A pointer to a structure of type `DNS_RPC_ZONE_LIST` (section [2.2.5.2.3](#)). This structure is used to enumerate zones.

**ZoneExport:** A pointer to a structure of type `DNS_RPC_ZONE_EXPORT_INFO` (section [2.2.5.2.8](#)). This structure is used to specify how a zone should be exported to file.

**DirectoryPartition:** A pointer to a structure of type `DNS_RPC_DP_INFO` (section [2.2.7.2.1](#)). This structure specifies detailed information about a single application directory partition.

**DirectoryPartitionEnum:** A pointer to a structure of type `DNS_RPC_DP_ENUM` (section [2.2.7.2.3](#)). This structure is used to specify basic information about an application directory partition.

**DirectoryPartitionList:** A pointer to a structure of type `DNS_RPC_DP_LIST` (section [2.2.7.2.4](#)). This structure is used to enumerate the Application Directory Partition Table.

**EnlistDirectoryPartition:** A pointer to a structure of type `DNS_RPC_ENLIST_DP` (section [2.2.7.2.5](#)). This structure is used to request enlistment changes for an application directory partition.

**ZoneChangeDirectoryPartition:** A pointer to a structure of type `DNS_RPC_ZONE_CHANGE_DP` (section [2.2.7.2.6](#)). This structure is used to request that a DNS zone be moved from one application directory partition to another.

**EnumZonesFilter:** A pointer to a structure of type `DNS_RPC_ENUM_ZONES_FILTER` (section [2.2.5.2.9](#)). This structure is used to filter DNS zones during enumeration.

**AddrArray:** A pointer to a structure of type `DNS_ADDR_ARRAY` (section [2.2.3.2.3](#)). This structure is used to specify a list of IPv4 or IPv6 addresses.

**ServerInfo:** A pointer to a structure of type `DNS_RPC_SERVER_INFO` (section [2.2.4.2.2](#)). This structure is used to specify general DNS server state and configuration.

**ZoneCreate:** A pointer to a structure of type `DNS_RPC_ZONE_CREATE_INFO` (section [2.2.5.2.9](#)). This structure is used to specify parameters required when creating a new DNS zone.

**Forwarders:** A pointer to a structure of type `DNS_RPC_FORWARDERS` (section [2.2.5.2.10](#)). This structure specifies the set of DNS servers this DNS server will forward unresolved queries to.

**Secondaries:** A pointer to a structure of type `DNS_RPC_ZONE_SECONDARIES` (section [2.2.5.2.5](#)). This structure is used to specify information about the secondary servers for a primary DNS zone.

**IpValidate:** A pointer to a structure of type [DNS\\_RPC\\_IP\\_VALIDATE \(section 2.2.3.2.4\)](#). This structure is used to request IP validation and to return the results of IP validation.

**ZoneInfo:** A pointer to a structure of type `DNS_RPC_ZONE_INFO` (section [2.2.5.2.4](#)). This structure is used to specify detailed DNS zone information.

**AutoConfigure:** A pointer to a structure of type `DNS_AUTOCONFIGURE` (section [2.2.8.2.1](#)). This structure is used to request DNS server autoconfiguration.

**Utf8StringList:** A pointer to a structure of type `DNS_RPC_UTF8_STRING_LIST` (section [2.2.1.2.3](#)). This structure is used to represent a list of UTF-8 [\[RFC3629\]](#) strings.

**UnicodeStringList:** A pointer to a structure of type [DNS\\_RPC\\_UNICODE\\_STRING\\_LIST](#) (section [2.2.1.2.4](#)). This structure is used to represent a list of Unicode strings.

**Skd:** A pointer to a structure of type [DNS\\_RPC\\_SKD](#) (section [2.2.6.2.1](#)). This structure is used to specify detailed signing key descriptor information.

**SkdList:** A pointer to a structure of type [DNS\\_RPC\\_SKD\\_LIST](#) (section [2.2.6.2.2](#)). This structure is used to enumerate signing key descriptors.

**SkdState:** A pointer to a structure of type [DNS\\_RPC\\_SKD\\_STATE](#) (section [2.2.6.2.3](#)). This structure is used to specify detailed signing key descriptor state information.

**SigningValidationError:** A pointer to a structure of type [DNS\\_RPC\\_SIGNING\\_VALIDATION\\_ERROR](#) (section [2.2.6.2.8](#)). This structure is used to specify signing key descriptor validation error information.

**TrustPointList:** A pointer to a structure of type [DNS\\_RPC\\_TRUST\\_POINT\\_LIST](#) (section [2.2.6.2.5](#)). This structure is used to enumerate trust points.

**TrustAnchorList:** A pointer to a structure of type [DNS\\_RPC\\_TRUST\\_ANCHOR\\_LIST](#) (section [2.2.6.2.7](#)). This structure is used to enumerate trust anchors.

**ZoneDnsSecSettings:** A pointer to a structure of type [DNS\\_RPC\\_ZONE\\_DNSSEC\\_SETTINGS](#) (section [2.2.6.2.9](#)). This structure is used to list the DNSSEC settings of a zone.

**ZoneScopeList:** A pointer to a structure of type [DNS\\_RPC\\_ENUM\\_ZONE\\_SCOPE\\_LIST](#) (section [2.2.13.2.1](#)). This structure is used to enumerate zone scopes in a specified zone.

**ZoneStats:** A pointer to a structure of type [DNS\\_RPC\\_ZONE\\_STATS\\_V1](#) (section [2.2.12.2.5](#)). This structure is used to enumerate the zone statistics.

**ScopeCreate:** A pointer to a structure of type [DNS\\_RPC\\_ZONE\\_SCOPE\\_CREATE\\_INFO\\_V1](#) (section [2.2.13.2.2.1](#)). This structure is used to create a zone scope.

**ScopeInfo:** A pointer to a structure of type [DNS\\_RPC\\_ZONE\\_SCOPE\\_INFO\\_V1](#) (section [2.2.13.2.3.1](#)). This structure is used to specify detailed DNS zone scope information.

## 2.2.2 Resource Record Messages

### 2.2.2.1 Enumerations and Constants

#### 2.2.2.1.1 DNS\_RECORD\_TYPE

The `DNS_RECORD_TYPE` is a 16-bit integer value that specifies DNS record types that can be enumerated by the DNS server.

Constant/value	Description
<code>DNS_TYPE_ZERO</code> 0x0000	An empty record type ( <a href="#">[RFC1034]</a> section 3.6 and <a href="#">[RFC1035]</a> section 3.2.2).
<code>DNS_TYPE_A</code> 0x0001	An A record type, used for storing an IP address ( <a href="#">[RFC1035]</a> section 3.2.2).

Constant/value	Description
DNS_TYPE_NS 0x0002	An <b>authoritative</b> name-server record type ( <a href="#">[RFC1034]</a> section 3.6 and <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_MD 0x0003	A mail-destination record type ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_MF 0x0004	A mail forwarder record type ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_CNAME 0x0005	A record type that contains the canonical name of a DNS alias ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_SOA 0x0006	A <b>Start of Authority (SOA)</b> record type ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_MB 0x0007	A mailbox record type ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_MG 0x0008	A mail group member record type ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_MR 0x0009	A mail-rename record type ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_NULL 0x000A	A record type for completion queries ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_WKS 0x000B	A record type for a well-known service ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_PTR 0x000C	A record type containing <b>FQDN</b> pointer ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_HINFO 0x000D	A host information record type ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_MINFO 0x000E	A mailbox or mailing list information record type ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_MX 0x000F	A mail-exchanger record type ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_TXT 0x0010	A record type containing a text string ( <a href="#">[RFC1035]</a> section 3.2.2).
DNS_TYPE_RP 0x0011	A responsible-person record type <a href="#">[RFC1183]</a> .
DNS_TYPE_AFSDB 0x0012	A record type containing AFS database location <a href="#">[RFC1183]</a> .
DNS_TYPE_X25	An X25 PSDN address record type <a href="#">[RFC1183]</a> .

Constant/value	Description
0x0013	
DNS_TYPE_ISDN 0x0014	An ISDN address record type <a href="#">[RFC1183]</a> .
DNS_TYPE_RT 0x0015	A route through record type <a href="#">[RFC1183]</a> .
DNS_TYPE_SIG 0x0018	A cryptographic public key signature record type <a href="#">[RFC2931]</a> .
DNS_TYPE_KEY 0x0019	A record type containing public key used in DNSSEC <a href="#">[RFC2535]</a> .
DNS_TYPE_AAAA 0x001C	An IPv6 address record type <a href="#">[RFC3596]</a> .
DNS_TYPE_LOC 0x001D	A location information record type <a href="#">[RFC1876]</a> .
DNS_TYPE_NXT 0x001E	A next-domain record type <a href="#">[RFC2065]</a> .
DNS_TYPE_SRV 0x0021	A server selection record type <a href="#">[RFC2782]</a> .
DNS_TYPE_ATMA 0x0022	An Asynchronous Transfer Mode (ATM) address record type <a href="#">[ATMA]</a> .
DNS_TYPE_NAPTR 0x0023	An NAPTR record type <a href="#">[RFC2915]</a> .
DNS_TYPE_DNAME 0x0027	A DNAME record type <a href="#">[RFC2672]</a> .
DNS_TYPE_DS 0x002B	A DS record type <a href="#">[RFC4034]</a> .
DNS_TYPE_RRSIG 0x002E	An RRSIG record type <a href="#">[RFC4034]</a> .
DNS_TYPE_NSEC 0x002F	An NSEC record type <a href="#">[RFC4034]</a> .
DNS_TYPE_DNSKEY 0x0030	A DNSKEY record type <a href="#">[RFC4034]</a> .
DNS_TYPE_DHCID 0x0031	A DHCID record type <a href="#">[RFC4701]</a> .
DNS_TYPE_NSEC3 0x0032	An NSEC3 record type <a href="#">[RFC5155]</a> .

Constant/value	Description
DNS_TYPE_NSEC3PARAM 0x0033	An NSEC3PARAM record type <a href="#">[RFC5155]</a> .
DNS_TYPE_ALL 0x00FF	A query-only type requesting all records <a href="#">[RFC1035]</a> .
DNS_TYPE_WINS 0xFF01	A record type containing <b>WINS</b> forward lookup data <a href="#">[MS-WINSRA]</a> .
DNS_TYPE_WINSR 0xFF02	A record type containing WINS reverse lookup data <a href="#">[MS-WINSRA]</a> .

An implementation SHOULD [<9>](#) support all the preceding record types.

Other type values that are not explicitly defined in the preceding table MUST be enumerable, including values defined by [\[IANA-DNS\]](#).

### 2.2.2.1.2 DNS\_RPC\_NODE\_FLAGS

DNS\_RPC\_NODE\_FLAGS enumerates the possible property values for the DNS\_RPC\_NODE and DNS\_RPC\_RECORD structures, which MUST have dwFlags field set to any combination of following possible values. These flags are used to indicate special properties of DNS records and to request special handling of DNS records during enumeration and modification operations.

Constant/value	Description
DNS_RPC_FLAG_CACHE_DATA 0x80000000	Data is from the DNS <b>cache</b> .
DNS_RPC_FLAG_ZONE_ROOT 0x40000000	Data is from enumeration performed at a zone-root. Applicable for dwFlags in DNS_RPC_NODE (section <a href="#">2.2.2.2.3</a> ).
DNS_RPC_FLAG_AUTH_ZONE_ROOT 0x20000000	Data is from enumeration performed at an authoritative zone-root. Applicable for dwFlags in DNS_RPC_NODE (section <a href="#">2.2.2.2.3</a> ).
DNS_RPC_FLAG_ZONE_DELEGATION 0x10000000	Data is from enumeration performed at a <b>node</b> that is represents in a delegated sub-zone. Applicable for dwFlags in DNS_RPC_NODE (section <a href="#">2.2.2.2.3</a> ).
DNS_RPC_FLAG_RECORD_DEFAULT_TTL 0x08000000	Record should use zone default <b>TTL</b> value. Applicable for dwFlags in DNS_RPC_RECORD (section <a href="#">2.2.2.2.5</a> ).
DNS_RPC_FLAG_RECORD_TTL_CHANGE 0x04000000	This flag is set in case of update record to update TTL value for the record. Applicable for dwFlags in DNS_RPC_RECORD (section <a href="#">2.2.2.2.5</a> ).
DNS_RPC_FLAG_RECORD_CREATE_PTR 0x02000000	This flag is set when adding or deleting a record. Applicable for dwFlags in DNS_RPC_RECORD (section <a href="#">2.2.2.2.5</a> ).
DNS_RPC_FLAG_NODE_STICKY 0x01000000	This flag is set when enumerating a node that is at the domain root. Applicable for dwFlags in DNS_RPC_NODE (section <a href="#">2.2.2.2.3</a> ).



Constant/value	Description
DNS_RPC_FLAG_NODE_COMPLETE 0x00800000	This flag is set when requested enumeration is completed with the buffer being returned. Applicable for dwFlags in DNS_RPC_NODE (section <a href="#">2.2.2.2.3</a> ).
DNS_RPC_FLAG_SUPPRESS_NOTIFY 0x00010000	This flag is set when updated record to disable zone-update notifications for a zone. Applicable for dwFlags in DNS_RPC_RECORD (section <a href="#">2.2.2.2.5</a> ).
DNS_RPC_FLAG_AGING_ON 0x00020000	This flag is set when updating a record to enable or disable <b>aging</b> for a record. Applicable for dwFlags in DNS_RPC_RECORD (section <a href="#">2.2.2.2.5</a> ).
DNS_RPC_FLAG_OPEN_ACL 0x00040000	This flag is set when updating a record to disable access control for a record. Applicable for dwFlags in DNS_RPC_RECORD (section <a href="#">2.2.2.2.5</a> ).

An implementation SHOULD support all flags above. <10>

## 2.2.2.2 Structures

### 2.2.2.2.1 DNS\_RPC\_NAME

The DNS\_RPC\_NAME structure is used to specify an FQDN, a DNS label, or another string in an RPC buffer by the DNS server. See section [3.1.6.3](#) for the handling of this structure in the directory server.

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					
cchNameLength										dnsName (variable)																										
...																																				

**cchNameLength (1 byte):** The length, in bytes, of the string stored in the dnsName member. To represent an empty string, cchNameLength MUST be zero and dnsName MUST be empty. The length of this structure will always be 4-byte aligned so there may be 0-3 bytes of padding at the end of this field. The pad bytes are not included in the cchNameLength count.

**dnsName (variable):** A UTF-8 string with length given by cchNameLength. The string MUST NOT be null-terminated. This string can represent a fully qualified domain name or any other string.

### 2.2.2.2.2 DNS\_COUNT\_NAME

The DNS\_COUNT\_NAME structure is used to specify an FQDN in an LDAP message.

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		
Length										LabelCount										RawName (variable)													

...

**Length (1 byte):** The length, in bytes, of the string stored in the RawName member, including null termination. To represent an empty string, Length MUST be zero, LabelCount MUST be zero, and RawName MUST be empty.

**LabelCount (1 byte):** The count of DNS labels in the RawName member.

**RawName (variable):** A string containing an FQDN in which a 1-byte label length count for the subsequent label has been inserted before the first label and in place of each "." delimiter. The string MUST be null-terminated. The maximum length of the string, including the null terminator, is 256 bytes.

### 2.2.2.2.3 DNS\_RPC\_NODE

DNS\_RPC\_NODE defines a structure that is used as a header for a list of DNS\_RPC\_RECORD structures (section [2.2.2.2.5](#)) returned by the DNS server inside a DNS\_RPC\_BUFFER (section [2.2.1.2.2](#)) while processing the R\_DnsrvcEnumRecords2 (section [3.1.4.9](#)).

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
wLength																wRecordCount															
dwFlags																															
dwChildCount																															
dnsNodeName (variable)																															
...																															

**wLength (2 bytes):** The length of this structure, in bytes, including the fixed size elements plus the length of the dnsNodeName element. The length of this structure will always be 4-byte aligned so there may be 0-3 bytes of padding at the end of the structure. The pad bytes are included in the wLength count.

**wRecordCount (2 bytes):** The number of DNS\_RPC\_RECORD structures that follow this node structure.

**dwFlags (4 bytes):** The properties of the DNS\_RPC\_NODE structure.

**dwChildCount (4 bytes):** The total number of children nodes below this node in the DNS record database.

**dnsNodeName (variable):** The name of this node in DNS\_RPC\_NAME (section [2.2.2.2.1](#)) format.

### 2.2.2.2.4 DNS\_RPC\_RECORD\_DATA

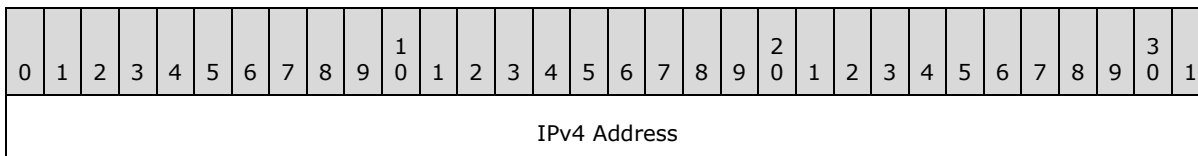
DNS\_RPC\_RECORD\_DATA defines a collection of possible resource record structures that are available on the DNS server. This is used by the DNS server while responding to

R\_DnsrrvEnumRecords2 (section [3.1.4.9](#)) method call to return resource record information inside a DNS\_RPC\_RECORD (2.2.2.2.5) structure that in turn is encapsulated inside a DNS\_RPC\_BUFFER (section [2.2.1.2.2](#)) structure. It is similarly used as input to the R\_DnsrrvUpdateRecord (section [3.1.4.5](#)) and R\_DnsrrvUpdateRecord2 (section [3.1.4.10](#)) method calls.

The DNS\_RPC\_RECORD\_DATA MUST be specified in one of the type-specific formats defined in a section that follows, where the type is indicated by the **wType** value in the DNS\_RPC\_RECORD (section [2.2.2.2.5](#)) structure. Further, for each DNS\_RECORD\_TYPE (section [2.2.2.1.1](#)) that the server supports, the server MUST support the corresponding DNS\_RPC\_RECORD\_DATA subtype.

#### 2.2.2.2.4.1 DNS\_RPC\_RECORD\_A

The DNS\_RPC\_RECORD\_A structure contains an IPv4 address. This record MUST be formatted as follows:



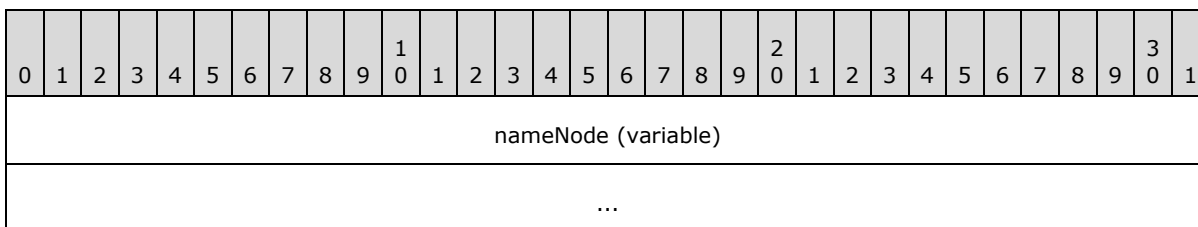
**IPv4 Address (4 bytes):** An IPv4 address in network byte order.

#### 2.2.2.2.4.2 DNS\_RPC\_RECORD\_NODE\_NAME

The DNS\_RPC\_RECORD\_NODE\_NAME structure contains information about a DNS record of any of the following types: [<11>](#)

- DNS\_TYPE\_PTR
- DNS\_TYPE\_NS
- DNS\_TYPE\_CNAME
- DNS\_TYPE\_DNAME
- DNS\_TYPE\_MB
- DNS\_TYPE\_MR
- DNS\_TYPE\_MG
- DNS\_TYPE\_MD
- DNS\_TYPE\_MF

This record MUST be formatted as follows:



**nameNode (variable):** The FQDN of this node in [DNS\\_RPC\\_NAME \(section 2.2.2.2.1\)](#) format.

### 2.2.2.2.4.3 DNS\_RPC\_RECORD\_SOA

The DNS\_RPC\_RECORD\_SOA structure contains information about an SOA record (section 3.3.13 in [RFC1035](#)). This record MUST be formatted as follows:

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
dwSerialNo																															
dwRefresh																															
dwRetry																															
dwExpire																															
dwMinimumTtl																															
namePrimaryServer (variable)																															
...																															
Zone Administrator Email (variable)																															
...																															

**dwSerialNo (4 bytes):** The **serial number** of the SOA record.

**dwRefresh (4 bytes):** The interval, in seconds, at which a **secondary DNS server** attempts to contact the **primary DNS server** for getting an update.

**dwRetry (4 bytes):** The interval, in seconds, at which a secondary DNS server retries to check with the primary DNS server in case of failure.

**dwExpire (4 bytes):** The duration, in seconds, that a secondary DNS server continues attempts to get updates from the primary DNS server and if still unsuccessful assumes that the primary DNS server is unreachable.

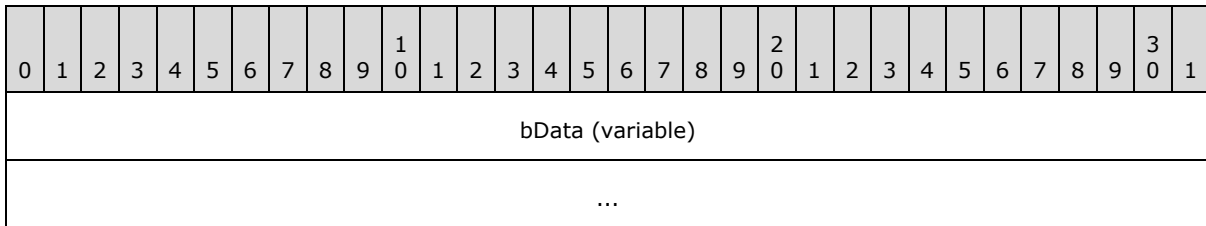
**dwMinimumTtl (4 bytes):** The minimum duration, in seconds, for which record data in the zone is valid.

**namePrimaryServer (variable):** The FQDN of the primary DNS server for this zone in DNS\_RPC\_NAME (section [2.2.2.2.1](#)) format.

**Zone Administrator Email (variable):** The contact email address for the zone administrators in a structure of type DNS\_RPC\_NAME (section [2.2.2.2.1](#)).

#### 2.2.2.2.4.4 DNS\_RPC\_RECORD\_NULL

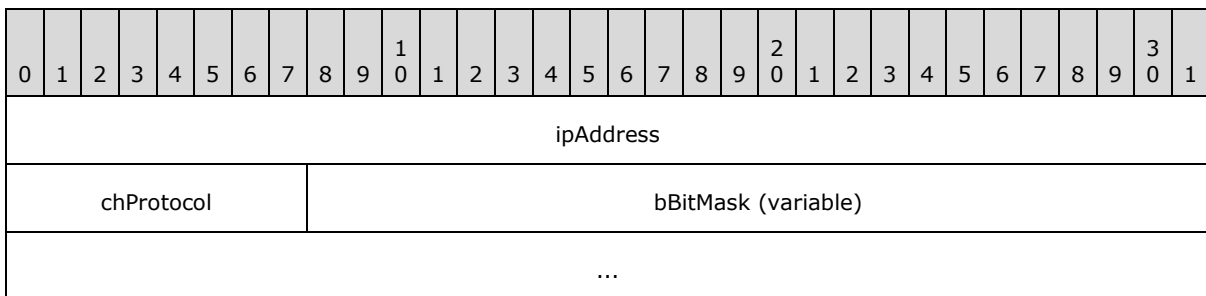
The DNS\_RPC\_RECORD\_NULL structure contains information for any record for which there is no more specific DNS\_RPC\_RECORD structure. This record MUST be formatted as follows:



**bData (variable):** An array of data. The sender can provide any data in this.

#### 2.2.2.2.4.5 DNS\_RPC\_RECORD\_WKS

The DNS\_RPC\_RECORD\_WKS structure contains the information for the well known services supported by a host, as defined in section 3.4.2 [\[RFC1035\]](#). This record MUST be formatted as follows:



**ipAddress (4 bytes):** The IPv4 address of the server that provides the service.

**chProtocol (1 byte):** The IP protocol number as specified in [\[IANA-PROTO-NUM\]](#).

**bBitMask (variable):** A list of service names (specified as "keywords" in the "WELL KNOWN PORT NUMBERS" section of [\[IANAPORT\]](#)) or port number if service name is unknown as an **ASCII** character string in DNS\_RPC\_NAME (section [2.2.2.2.1](#)) format. If more than one port is listed for a single combination of service name and IP protocol number, then only the first such port number is indicated when that service name is used. Each service name or port MUST be separated by a single space character, and the string MUST be terminated by a single null character. Each port number specified MUST be less than or equal to 1024. The terminating null character MUST be included in the **cchNameLength** field of the DNS\_RPC\_NAME (section [2.2.2.2.1](#)) structure.

#### 2.2.2.2.4.6 DNS\_RPC\_RECORD\_STRING

The DNS\_RPC\_RECORD\_STRING structure contains information about a DNS record of any of the following types:

- DNS\_TYPE\_HINFO
- DNS\_TYPE\_ISDN
- DNS\_TYPE\_TXT

- DNS\_TYPE\_X25
- DNS\_TYPE\_LOC

This packet contains one or more instances of stringData, depending upon the type listed above. This record MUST be formatted as follows:

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
stringData (variable)																															
...																															

**stringData (variable):** Each stringData member contains the host name value for a node in DNS\_RPC\_NAME (section [2.2.2.2.1](#)) structure.

#### 2.2.2.2.4.7 DNS\_RPC\_RECORD\_MAIL\_ERROR

The DNS\_RPC\_RECORD\_MAIL\_ERROR structure contains information about a DNS record of any of the following types:

- DNS\_TYPE\_MINFO
- DNS\_TYPE\_RP

This record MUST be formatted as follows:

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
nameMailBox (variable)																															
...																															
ErrorMailBox (variable)																															
...																															

**nameMailBox (variable):** A structure of type DNS\_RPC\_NAME (section [2.2.2.2.1](#)) containing the RMAILBX value specified in section 3.3.7 of [\[RFC1035\]](#) for an MINFO record, or the mbox-dname value specified in section 2.2 of [\[RFC1183\]](#) for an RP record.

**ErrorMailBox (variable):** A structure of type DNS\_RPC\_NAME (section [2.2.2.2.1](#)) containing the EMAILBX value specified in section 3.3.7 of [\[RFC1035\]](#) for an MINFO record, or the txt-dname value specified in section 2.2 of [\[RFC1183\]](#) for an RP record.

#### 2.2.2.2.4.8 DNS\_RPC\_RECORD\_NAME\_PREFERENCE

The DNS\_RPC\_RECORD\_NAME\_PREFERENCE structure specifies information about a DNS record of any of the following types:

- DNS\_TYPE\_MX
- DNS\_TYPE\_AFSDB
- DNS\_TYPE\_RT

This record MUST be formatted as follows:

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
wPreference																nameExchange (variable)															
...																															

**wPreference (2 bytes):** The preference value for the DNS server that holds the record.

**nameExchange (variable):** The FQDN of the server hosting the mail-exchange and specified in DNS\_RPC\_NAME (section [2.2.2.2.1](#)) format.

#### 2.2.2.2.4.9 DNS\_RPC\_RECORD\_SIG

The DNS\_RPC\_RECORD\_SIG structure contains information about cryptographic public key signatures as specified in section 4 of [\[RFC2535\].<12>](#) This record MUST be formatted as follows:

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
wTypeCovered																chAlgorithm								chLabelCount							
dwOriginalTtl																															
dwSigExpiration																															
dwSigInception																															
wKeyTag																nameSigner (variable)															
...																															
SignatureInfo (variable)																															
...																															

**wTypeCovered (2 bytes):** The type covered value for SIG RR as specified in section 4.1 of [\[RFC2535\]](#).

**chAlgorithm (1 byte):** The algorithm value for SIG RR as specified in section 4.1 of [\[RFC2535\]](#).

**chLabelCount (1 byte):** The total number of labels present in the name of the record signed by the SIG RR as specified in section 4.1 of [\[RFC2535\]](#).

**dwOriginalTtl (4 bytes):** The original TTL value of the record signed by the SIG RR as specified in section 4.1 of [\[RFC2535\]](#).

**dwSigExpiration (4 bytes):** The signature expiration time as specified in section 4.1 of [\[RFC2535\]](#).

**dwSigInception (4 bytes):** The signature inception time as specified in section 4.1 of [\[RFC2535\]](#).

**wKeyTag (2 bytes):** The key tag value for SIG RR as specified in section 4.1 of [\[RFC2535\]](#).

**nameSigner (variable):** Pointer to a structure of type [DNS\\_RPC\\_NAME](#) (section [2.2.2.2.1](#)) containing the FQDN of the originating host for this record.

**SignatureInfo (variable):** Binary signature information as specified in section 4.1 of [\[RFC2535\]](#).

#### 2.2.2.2.4.10 DNS\_RPC\_RECORD\_RRSIG

The DNS\_RPC\_RECORD\_RRSIG structure contains information about cryptographic public key signatures as specified in section 3 of [\[RFC4034\]](#).<sup><13></sup> This record MUST be formatted as follows.

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
wTypeCovered											chAlgorithm										chLabelCount										
dwOriginalTtl																															
dwSigExpiration																															
dwSigInception																															
wKeyTag											nameSigner (variable)																				
...																															
SignatureInfo (variable)																															
...																															

**wTypeCovered (2 bytes):** The type covered value for RRSIG RR as specified in section 3.1 of [\[RFC4034\]](#).

**chAlgorithm (1 byte):** The algorithm value for RRSIG RR as specified in section 3.1 of [\[RFC4034\]](#).

**chLabelCount (1 byte):** The total number of labels present in the name of the record signed by the RRSIG RR as specified in section 3.1 of [\[RFC4034\]](#).



**dwOriginalTtl (4 bytes):** The original TTL value of the record signed by the RRSIG RR as specified in section 3.1 of [\[RFC4034\]](#).

**dwSigExpiration (4 bytes):** The signature expiration time as specified in section 3.1 of [\[RFC4034\]](#).

**dwSigInception (4 bytes):** The signature inception time as specified in section 3.1 of [\[RFC4034\]](#).

**wKeyTag (2 bytes):** The tag value for RRSIG RR as specified in section 3.1 of [\[RFC4034\]](#).

**nameSigner (variable):** A structure of type [DNS\\_RPC\\_NAME](#) (section [2.2.2.2.1](#)) containing the FQDN of the originating host for this record.

**SignatureInfo (variable):** Binary signature information as specified in section 3.1 of [\[RFC4034\]](#).

#### 2.2.2.2.4.11 DNS\_RPC\_RECORD\_NSEC

The DNS\_RPC\_RECORD\_NSEC structure contains the next FQDN in the zone as specified in section 4 of [\[RFC4034\]](#).<14> This record MUST be formatted as follows.

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
nameSigner (variable)																															
...																															
NSECBitmap (variable)																															
...																															

**nameSigner (variable):** A structure of type [DNS\\_RPC\\_NAME](#) (section [2.2.2.2.1](#)) containing the FQDN of the originating host for this record.

**NSECBitmap (variable):** Bitmap of types present at this node as specified in section 4 of [\[RFC4034\]](#).

#### 2.2.2.2.4.12 DNS\_RPC\_RECORD\_DS

The DNS\_RPC\_RECORD\_DS structure contains a public key associated with an FQDN as specified in section 5 of [\[RFC4034\]](#).<15> This record MUST be formatted as follows.

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
wKeyTag										chAlgorithm										chDigestType											
bDigest (variable)																															
...																															

**wKeyTag (2 bytes):** The key tag of the DNSKEY record referred to by this DS record, as specified in section 5 of [\[RFC4034\]](#).

**chAlgorithm (1 byte):** Algorithm value of the DNSKEY record referred to by this DS record, as specified in section 5 of [\[RFC4034\]](#).

**chDigestType (1 byte):** The digest algorithm that was used to construct this DS record, as specified in section 5 of [\[RFC4034\]](#).

**bDigest (variable):** An array of bytes containing the digest value as specified of section 5 in [\[RFC4034\]](#).

#### 2.2.2.2.4.13 DNS\_RPC\_RECORD\_KEY

The DNS\_RPC\_RECORD\_KEY structure contains a public key associated with an FQDN as specified in section 3 of [\[RFC2535\]](#).<16> This record MUST be formatted as follows:

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
wFlags																chProtocol						chAlgorithm									
bKey (variable)																															
...																															

**wFlags (2 bytes):** Flags value for the key RR as specified in section 3.1 of [\[RFC2535\]](#).

**chProtocol (1 byte):** Protocol value for the key RR as specified in section 3.1 of [\[RFC2535\]](#).

**chAlgorithm (1 byte):** Algorithm value for the key RR as specified in section 3.1 of [\[RFC2535\]](#).

**bKey (variable):** An array of bytes containing the key value as specified of section 3.1 in [\[RFC2535\]](#).

#### 2.2.2.2.4.14 DNS\_RPC\_RECORD\_DHCID

The DNS\_RPC\_RECORD\_DHCID structure contains a public key associated with an FQDN as specified in section 3 of [\[RFC2535\]](#).<17> This record MUST be formatted as follows.

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
bDHCID (variable)																															
...																															

**bDHCID (variable):** An opaque DHCID record as specified in section 3 in [\[RFC4701\]](#).

#### 2.2.2.2.4.15 DNS\_RPC\_RECORD\_DNSKEY

The DNS\_RPC\_RECORD\_DNSKEY structure contains a public key associated with an FQDN as specified in section 2 of [\[RFC4034\]](#).<18> This record MUST be formatted as follows.

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
wFlags																chProtocol						chAlgorithm									
bKey (variable)																															
...																															

**wFlags (2 bytes):** Flags value for the key RR as specified in section 2.1 of [\[RFC4034\]](#).

**chProtocol (1 byte):** Protocol value for the key RR as specified in section 2.1 of [\[RFC4034\]](#).

**chAlgorithm (1 byte):** Algorithm value for the key RR as specified in section 2.1 of [\[RFC4034\]](#).

**bKey (variable):** An array of bytes containing the key value as specified of section 2.1 in [\[RFC4034\]](#).

#### 2.2.2.2.4.16 DNS\_RPC\_RECORD\_AAAA

The DNS\_RPC\_RECORD\_AAAA structure contains IPv6 address information. This record MUST be formatted as follows:

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
ipv6Address																															
...																															
...																															
...																															

**ipv6Address (16 bytes):** An IPv6 address member holds an IPv6 address, in network byte order.

#### 2.2.2.2.4.17 DNS\_RPC\_RECORD\_NXT

The DNS\_RPC\_RECORD\_NXT specifies a NXT resource record as specified in section 5.1 of [\[RFC2535\].<19>](#) This record MUST be formatted as follows:

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
wNumRecordTypes																wTypeWords (variable)															
...																															

nextName (variable)
...

**wNumRecordTypes (2 bytes):** The number of 16-bit unsigned integers in the variable sized wTypeWords array. This value MUST be greater than 1.

**wTypeWords (variable):** An array for of 16-bit unsigned integers in little-endian byte order for that contains a variable sized bit-mask value for types present in this record, as specified in section 5.2 of [\[RFC2535\]](#). The most significant bit of the first integer corresponds to type zero and MUST be zero. If there is a second 16-bit unsigned integer present in the array, the most significant bit of the second integer corresponds to type 16, and so on.

**nextName (variable):** A DNS\_RPC\_NAME (section [2.2.2.2.1](#)) containing next name information, as specified in section 5.2 of [\[RFC2535\]](#).

#### 2.2.2.2.4.18 DNS\_RPC\_RECORD\_SRV

The DNS\_RPC\_RECORD\_SRV specifies an SRV resource record as specified in [\[RFC2782\]](#). This record MUST be formatted as follows:

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
wPriority																wWeight															
wPort																nameTarget (variable)															
...																															

**wPriority (2 bytes):** The priority of the target host as specified in [\[RFC2782\]](#).

**wWeight (2 bytes):** The relative weight for the target host as specified in [\[RFC2782\]](#).

**wPort (2 bytes):** The port number for the service on the target host as specified in [\[RFC2782\]](#).

**nameTarget (variable):** The FDQN of the server that hosts this service in DNS\_RPC\_NAME (section [2.2.2.2.1](#)) format, as specified in [\[RFC2782\]](#).

#### 2.2.2.2.4.19 DNS\_RPC\_RECORD\_ATMA

The DNS\_RPC\_RECORD\_ATMA specifies a resource record that contains ATM address information as specified in [\[ATMA\]](#). This record MUST be formatted as follows:

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
chFormat										bAddress (variable)																					
...																															

**chFormat (1 byte):** The format of the address as specified in section 5.2 of [\[ATMA\]](#).

**bAddress (variable):** The ATM address of the node to which this resource record pertains (see section 5.2 of [\[ATMA\]](#)).

#### 2.2.2.2.4.20 DNS\_RPC\_RECORD\_NAPTR

The DNS\_RPC\_RECORD\_NAPTR specifies a NAPTR resource record as specified in section 4 of [\[RFC3403\]](#).<20> This record MUST be formatted as follows:

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
wOrder											wPreference																				
nameFlags (variable)																															
...																															
nameService (variable)																															
...																															
nameSubstitution (variable)																															
...																															
nameReplacement (variable)																															
...																															

**wOrder (2 bytes):** A value that specifies the order in which the NAPTR record should be processed, as specified in section 4.1 in [\[RFC3403\]](#).

**wPreference (2 bytes):** The preference given to this NAPTR record, as specified in section 4.1 in [\[RFC3403\]](#).

**nameFlags (variable):** Pointer to a structure of type DNS\_RPC\_NAME (section [2.2.2.2.1](#)) containing the string flags value as specified in section 4.1 in [\[RFC3403\]](#).

**nameService (variable):** Pointer to a structure of type DNS\_RPC\_NAME (section [2.2.2.2.1](#)) containing service parameters value for NAPTR to control the rewriting and interpretation of the field in the record, as specified in section 4.1 in [\[RFC3403\]](#).

**nameSubstitution (variable):** Pointer to a structure of type DNS\_RPC\_NAME (section [2.2.2.2.1](#)) containing a substitution expression value for the NAPTR record, as specified in section 4.1 in [\[RFC3403\]](#).

**nameReplacement (variable):** Pointer to a structure of type DNS\_RPC\_NAME (section [2.2.2.2.1](#)) containing a replacement expression value for the NAPTR record, as specified in section 4.1 in [\[RFC3403\]](#).

### 2.2.2.2.4.21 DNS\_RPC\_RECORD\_WINS

The DNS\_RPC\_RECORD\_WINS specifies a WINS resource record. This record MUST be formatted as follows:

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
dwMappingFlag																															
dwLookupTimeout																															
dwCacheTimeout																															
cWinsServerCount																															
aipWinsServers (variable)																															
...																															

**dwMappingFlag (4 bytes):** The scope of the WINS record lookups. This value MUST be set to 0x00000000 or any combination of the following:

Value	Meaning
DNS_WINS_FLAG_SCOPE 0x80000000	Server forwards lookup requests to remote WINS servers.
DNS_WINS_FLAG_LOCAL 0x00010000	Server performs WINS lookups locally.

**dwLookupTimeout (4 bytes):** The duration, in seconds, for which the server waits to receive a response from a remote DNS server.

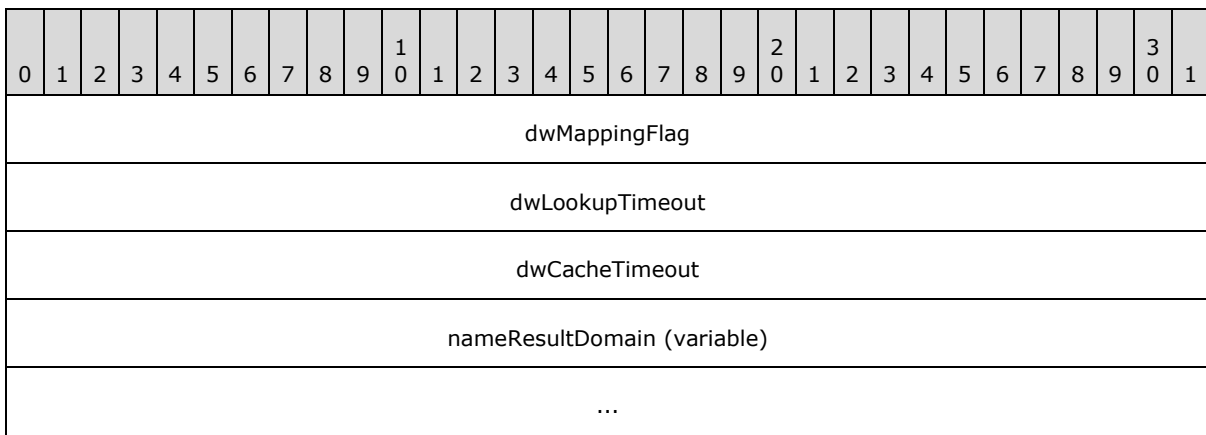
**dwCacheTimeout (4 bytes):** The duration, in seconds, for which the server keeps this record in its cache before considering it stale.

**cWinsServerCount (4 bytes):** The number of WINS server addresses in this record. The value of this field MUST be at least one.

**aipWinsServers (variable):** An array of IPv4 addresses in network byte order with length given by cWinsServerCount.

### 2.2.2.2.4.22 DNS\_RPC\_RECORD\_WINSR

The DNS\_RPC\_RECORD\_WINSR specifies a **Windows Internet Name Service Reverse Lookup (WINS-R)** resource record. This record MUST be formatted as follows:



**dwMappingFlag (4 bytes):** The scope of the WINS-R record lookups. This value MUST be set to zero or any combination of the following:

Value	Meaning
DNS_WINS_FLAG_SCOPE 0x80000000	Server forwards lookup requests to remote WINS servers.
DNS_WINS_FLAG_LOCAL 0x00010000	Server performs WINS lookups locally.

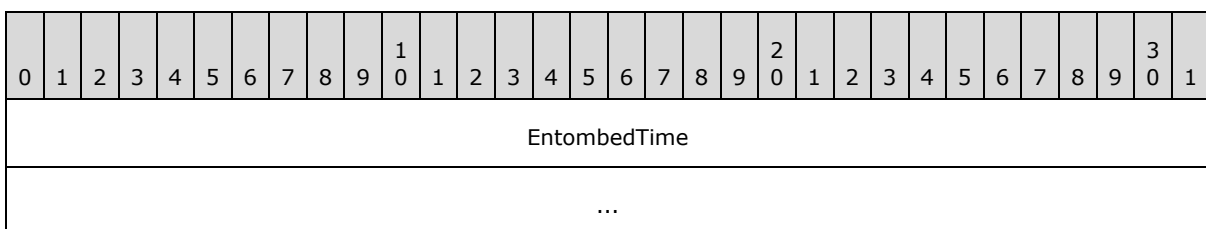
**dwLookupTimeout (4 bytes):** The duration, in seconds, for which server waits to receive a response from a remote DNS server.

**dwCacheTimeout (4 bytes):** The duration, in seconds, for which server keeps this record in its cache before considering it stale.

**nameResultDomain (variable):** Pointer to a structure of type DNS\_RPC\_NAME (section [2.2.2.2.1](#)) containing a **domain name (3)** suffix that will be appended to a **single-label name** obtained from a WINS-R lookup.

#### 2.2.2.2.4.23 DNS\_RPC\_RECORD\_TS

The DNS\_RPC\_RECORD\_TS specifies information for a node that has been **tombstoned**.



**EntombedTime (8 bytes):** The time at which this node was tombstoned, represented as the number of 100-nanosecond intervals since midnight (00:00:00), January 1, 1601 UTC.

#### 2.2.2.2.4.24 DNS\_RPC\_RECORD\_NSEC3

The DNS\_RPC\_RECORD\_NSEC3 structure specifies an NSEC3 resource record as specified in [\[RFC5155\]](#) section 3.<21> This record MUST be formatted as follows.

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
chAlgorithm										bFlags										wIterations											
bSaltLength										bHashLength										salt (variable)											
...																															
nextHashedOwnerName (variable)																															
...																															
bitmaps (variable)																															
...																															

**chAlgorithm (1 byte):** A value that specifies the cryptographic hash algorithm used to construct the hash value, as specified in [\[RFC5155\]](#) section 3.1.

**bFlags (1 byte):** A value that specifies NSEC3 processing flags, as specified in [\[RFC5155\]](#) section 3.1.

**wIterations (2 bytes):** A value that specifies the number of additional times the hash function has been performed, as specified in [\[RFC5155\]](#) section 3.1.

**bSaltLength (1 byte):** A value that specifies the length of the salt field, in octets.

**bHashLength (1 byte):** A value that specifies the length of the nextHashedOwnerName field, in octets.

**salt (variable):** A value that specifies the salt to be appended to the original owner name before hashing, as specified in [\[RFC5155\]](#) section 3.1.

**nextHashedOwnerName (variable):** A value that specifies the next hashed owner name in hash order, as specified in [\[RFC5155\]](#) section 3.1.

**bitmaps (variable):** A value that specifies the DNS types that exist at the original owner name of the NSEC3 record, as specified in [\[RFC5155\]](#) section 3.1.

#### 2.2.2.2.4.25 DNS\_RPC\_RECORD\_NSEC3PARAM

The DNS\_RPC\_RECORD\_NSEC3PARAM structure specifies an NSEC3PARAM resource record as specified in [\[RFC5155\]](#) section 3.<22> This record MUST be formatted as follows.



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
chAlgorithm										bFlags										wIterations											
bSaltLength										salt (variable)																					
...																															

**chAlgorithm (1 byte):** A value that specifies the cryptographic hash algorithm used to construct the hash value, as specified in [\[RFC5155\]](#) section 4.1.

**bFlags (1 byte):** A value that specifies NSEC3 processing flags, as specified in [\[RFC5155\]](#) section 3.1.

**wIterations (2 bytes):** A value that specifies the number of additional times the hash function has been performed, as specified in [\[RFC5155\]](#) section 4.1.

**bSaltLength (1 byte):** A value that specifies the length of the salt field, in octets.

**salt (variable):** A value that specifies the salt to be appended to the original owner name before hashing, as specified in [\[RFC5155\]](#).

#### 2.2.2.2.5 DNS\_RPC\_RECORD

The DNS\_RPC\_RECORD structure is used to specify a single DNS record's parameters and data. This structure is returned by the DNS server in response to an R\_DnsrvcEnumRecords2 (section [3.1.4.9](#)) method call.

```
typedef struct _DnsrvcRpcRecord {
    WORD wDataLength;
    WORD wType;
    DWORD dwFlags;
    DWORD dwSerial;
    DWORD dwTtlSeconds;
    DWORD dwTimeStamp;
    DWORD dwReserved;
    [size_is(wDataLength)] BYTE Buffer[];
} DNS_RPC_RECORD,
*PDNS_RPC_RECORD,
DNS_FLAT_RECORD,
*PDNS_FLAT_RECORD;
```

**wDataLength:** The total size of the variable buffer, in bytes. Note that the DNS\_RPC\_RECORD structure is always 4-byte aligned, which means there may be 0-3 bytes of padding at the end of the structure. The pad bytes are not included in the wDataLength count.

**wType:** The type of the resource record, as specified in section [2.2.2.1.1](#) DNS\_RECORD\_TYPE.

**dwFlags:** Resource record properties. This field may contain one of the RANK\* flags in the low-order bits and one of the DNS\_RPC\_FLAGS\* in the high-order bits.

Value	Meaning
RANK_CACHE_BIT 0x00000001	The record came from the cache.
RANK_ROOT_HINT 0x00000008	The record is a preconfigured root hint.
RANK_OUTSIDE_GLUE 0x00000020	This value is not used.
RANK_CACHE_NA_ADDITIONAL 0x00000031	The record was cached from the additional section of a non-authoritative response.
RANK_CACHE_NA_AUTHORITY 0x00000041	The record was cached from the authority section of a non-authoritative response.
RANK_CACHE_A_ADDITIONAL 0x00000051	The record was cached from the additional section of an authoritative response.
RANK_CACHE_NA_ANSWER 0x00000061	The record was cached from the answer section of a non-authoritative response.
RANK_CACHE_A_AUTHORITY 0x00000071	The record was cached from the authority section of an authoritative response.
RANK_GLUE 0x00000080	The record is a <b>glue record</b> in an authoritative zone.
RANK_NS_GLUE 0x00000082	The record is a <b>delegation</b> (type NS) record in an authoritative zone.
RANK_CACHE_A_ANSWER 0x000000c1	The record was cached from the answer section of an authoritative response.
RANK_ZONE 0x000000f0	The record comes from an authoritative zone.
DNS_RPC_FLAG_ZONE_ROOT 0x40000000	The record is at the root of a zone (not necessarily a zone hosted by this server; the record could have come from the cache).
DNS_RPC_FLAG_AUTH_ZONE_ROOT 0x20000000	The record is at the root of a zone that is locally hosted on this server.
DNS_RPC_FLAG_CACHE_DATA 0x80000000	The record came from the cache.

**dwSerial:** This MUST be set to 0x00000000 when sent by the client or server, and ignored on receipt by the server or client.

**dwTtlSeconds:** The duration, in seconds, after which this record will expire.

**dwTimeStamp:** The **time-stamp**, in hours, for the record when it received the last update.

**dwReserved:** This value MUST be set to 0x00000000 when sent by the client and ignored on receipt by the server.

**Buffer:** Record data in DNS\_RPC\_RECORD\_DATA (section [2.2.2.2.4](#)) format where type is specified by the value **wType**.<23>

Value	Meaning
DNS_TYPE_ZERO 0x0000	DNS_RPC_RECORD_TS
DNS_TYPE_A 0x0001	DNS_RPC_RECORD_A
DNS_TYPE_NS 0x0002	DNS_RPC_RECORD_NODE_NAME
DNS_TYPE_MD 0x0003	DNS_RPC_RECORD_NODE_NAME
DNS_TYPE_MF 0x0004	DNS_RPC_RECORD_NODE_NAME
DNS_TYPE_CNAME 0x0005	DNS_RPC_RECORD_NODE_NAME
DNS_TYPE_SOA 0x0006	DNS_RPC_RECORD_SOA
DNS_TYPE_MB 0x0007	DNS_RPC_RECORD_NODE_NAME
DNS_TYPE_MG 0x0008	DNS_RPC_RECORD_NODE_NAME
DNS_TYPE_MR 0x0009	DNS_RPC_RECORD_NODE_NAME
DNS_TYPE_NULL 0x000A	DNS_RPC_RECORD_NULL
DNS_TYPE_WKS 0x000B	DNS_RPC_RECORD_WKS
DNS_TYPE_PTR 0x000C	DNS_RPC_RECORD_NODE_NAME
DNS_TYPE_HINFO 0x000D	DNS_RPC_RECORD_STRING
DNS_TYPE_MINFO 0x000E	DNS_RPC_RECORD_MAIL_ERROR
DNS_TYPE_MX 0x000F	DNS_RPC_RECORD_NAME_PREFERENCE
DNS_TYPE_TXT 0x0010	DNS_RPC_RECORD_STRING
DNS_TYPE_RP 0x0011	DNS_RPC_RECORD_MAIL_ERROR

<b>Value</b>	<b>Meaning</b>
DNS_TYPE_AFSDB 0x0012	DNS_RPC_RECORD_NAME_PREFERENCE
DNS_TYPE_X25 0x0013	DNS_RPC_RECORD_STRING
DNS_TYPE_ISDN 0x0014	DNS_RPC_RECORD_STRING
DNS_TYPE_RT 0x0015	DNS_RPC_RECORD_NAME_PREFERENCE
DNS_TYPE_SIG 0x0018	DNS_RPC_RECORD_SIG
DNS_TYPE_KEY 0x0019	DNS_RPC_RECORD_KEY
DNS_TYPE_AAAA 0x001C	DNS_RPC_RECORD_AAAA
DNS_TYPE_NXT 0x001E	DNS_RPC_RECORD_NXT
DNS_TYPE_SRV 0x0021	DNS_RPC_RECORD_SRV
DNS_TYPE_ATMA 0x0022	DNS_RPC_RECORD_ATMA
DNS_TYPE_NAPTR 0x0023	DNS_RPC_RECORD_NAPTR
DNS_TYPE_DNAME 0x0027	DNS_RPC_RECORD_NODE_NAME
DNS_TYPE_DS 0x002B	DNS_RPC_RECORD_DS
DNS_TYPE_RRSIG 0x002E	DNS_RPC_RECORD_RRSIG
DNS_TYPE_NSEC 0x002F	DNS_RPC_RECORD_NSEC
DNS_TYPE_DNSKEY 0x0030	DNS_RPC_RECORD_DNSKEY
DNS_TYPE_DHCID 0x0031	DNS_RPC_RECORD_DHCID
DNS_TYPE_NSEC3 0x0032	DNS_RPC_RECORD_NSEC3
DNS_TYPE_NSEC3PARAM 0x0033	DNS_RPC_RECORD_NSEC3PARAM

Value	Meaning
DNS_TYPE_WINS 0xFF01	DNS_RPC_RECORD_WINS
DNS_TYPE_WINSR 0xFF02	DNS_RPC_RECORD_WINSR

Other type values that are not explicitly defined in the preceding table MUST be enumerable, including values defined by [\[IANA-DNS\]](#), and they MUST use the DNS\_RPC\_RECORD\_NULL (section [2.2.2.2.4.4](#)) structure.

## 2.2.3 Address Information Messages

### 2.2.3.1 Enumerations and Constants

#### 2.2.3.1.1 DNS\_IPVAL\_CONTEXT

DNS\_IPVAL\_CONTEXT is a DWORD value that specifies possible context values for IP validation. This is used to populate the dwContext field in DNS\_RPC\_IP\_VALIDATE (section [2.2.3.2.4](#)). The DNS server MUST use this value to determine the type of validation that SHOULD be performed for IP address specified in DNS\_RPC\_IP\_VALIDATE (section [2.2.3.2.4](#))<24>.

Constant/value	Description
DNS_IPVAL_DNS_SERVERS 0x00000000	Validate that IP addresses are reachable and operational by the DNS servers.
DNS_IPVAL_DNS_ROOTHINTS 0x00000001	Validate that IP addresses are suitable as root-hints.
DNS_IPVAL_DNS_FORWARDERS 0x00000002	Validate that IP addresses are server level forwarders.
DNS_IPVAL_DNS_ZONE_MASTERS 0x00000003	Validate that IP addresses are remote DNS servers hosting a zone, named as pointed to by pszContextName in the DNS_RPC_IP_VALIDATE (section <a href="#">2.2.3.2.4</a> ).
DNS_IPVAL_DNS_DELEGATIONS 0x00000004	Validate that IP addresses are remote DNS servers are name-server for the delegated zone, named as pointed to by <b>pszContextName</b> in the DNS_RPC_IP_VALIDATE (section <a href="#">2.2.3.2.4</a> ).

#### 2.2.3.1.2 DNS\_IP\_VALIDATE\_RETURN\_FLAGS

DNS\_IP\_VALIDATE\_RETURN\_FLAGS is a DWORD value that specifies the results of IP validation performed by the DNS server. This value will be used by the DNS server to populate the Flags field within each DNS\_ADDR structure (section [2.2.3.2.2](#)) present in the DNS\_ADDR\_ARRAY (section [2.2.3.2.3](#)) structure which in turn is present inside the returned DNS\_RPC\_IP\_VALIDATE structure (section [2.2.3.2.4](#))<25>.

Constant/value	Description
ERROR_SUCCESS 0x00000000	The remote IP address is valid.

Constant/value	Description
DNS_IPVAL_INVALID_ADDR 0x00000001	Remote IP address is not a valid IP address.
DNS_IPVAL_UNREACHABLE 0x00000002	Remote IP address is not reachable.
DNS_IPVAL_NO_RESPONSE 0x00000003	Remote IP address does not appear to be hosting a DNS server.
DNS_IPVAL_NOT_AUTH_FOR_ZONE 0x00000004	Remote IP address is not authoritative for the required zone, specified by pszContextName in the DNS_RPC_IP_VALIDATE (section <a href="#">2.2.3.2.4</a> ).
DNS_IPVAL_UNKNOWN_ERROR 0x000000FF	The DNS server encountered an unknown error occurred while validating IP address.
DNS_IPVAL_NO_TCP 0x80000000	Indicated that remote IP address, responds to UDP DNS messages, but does not respond to TCP DNS messages.

## 2.2.3.2 Structures

### 2.2.3.2.1 IP4\_ARRAY

The IP4\_ARRAY structure is used to represent an array of IPv4 addresses. This structure cannot represent IPv6 addresses.

```
typedef struct _IP4_ARRAY {
    DWORD AddrCount;
    [size_is(AddrCount)] DWORD AddrArray[];
} IP4_ARRAY,
*PIP4_ARRAY;
```

**AddrCount:** The number of IPv4 addresses present in the AddrArray member.

**AddrArray:** An array of IPv4 addresses. An IPv4 address is represented as a 32-bit unsigned integer in network byte order.

An empty IP4\_ARRAY is represented by **AddrCount** set to zero and **AddrArray** unused. Senders of an empty IP4\_ARRAY MUST set **AddrArray** to a single entry containing binary zeros, and receivers MUST ignore it.

### 2.2.3.2.2 DNS\_ADDR

This DNS\_ADDR structure is used to represent an IP address. The IP address may be either IPv4 or IPv6.

```
typedef struct _DnsAddr {
    CHAR MaxSa[32];
    DWORD DnsAddrUserDword[8];
} DNS_ADDR,
*PDNS_ADDR;
```

**MaxSa:** This field MUST be constructed as specified in [DNS\\_ADDR](#).

**DnsAddrUserDword:** This field MUST be constructed as specified in [DNS\\_ADD\\_USER](#).

Any field not specified above MUST be set to zero by the sender and ignored by the receiver.

### 2.2.3.2.2.1 DNS\_ADDR

An IPv4 or IPv6 addressed used by [DNS\\_ADDR](#) **MaxSa** field.

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
Address Family																Port Number															
IPv4 Address																															
IPv6 Address																															
...																															
...																															
...																															
Padding																															
...																															

**Address Family (2 bytes):** This MUST be set to 0x0002 if this is an IPv4 address or 0x0017 if this is an IPv6 address.

0x0002  
0x0017

**Port Number (2 bytes):** Senders MUST set this to zero, and receivers MUST ignore it.

**IPv4 Address (4 bytes):** An IPv4 address in network byte order value for the host pointed to by **DNS\_ADDR** structure.

**IPv6 Address (16 bytes):** An IPv6 address in network byte order value for the host pointed to by **DNS\_ADDR** structure.

**Padding (8 bytes):** Senders MUST set this to zero, and receivers MUST ignore it.

### 2.2.3.2.2.2 DNS\_ADD\_USER

Used by [DNS\\_ADDR](#) **DnsAddrUserDword** field.

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
SockaddrLength																															
SubnetLength																															
Flags																															
Padding																															
...																															
...																															
...																															
...																															

**SockaddrLength (4 bytes):** The length of valid data in the socket address structure present above this field.

**SubnetLength (4 bytes):** Senders MUST set this to 0x00000000 and receivers MUST ignore this value.

**Flags (4 bytes):** Clients MUST set to 0x00000000. Senders will use this field to indicate the results of IP validation for this IP address, where it has a format as follows (in host 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
T	Zero							RTT												validationStatus											

**T (1 bit):** If set, DNS over UDP is available from the remote DNS server but DNS over TCP is not available.

**Zero (7 bits):** Reserved. Senders MUST set these bits to zero and receivers MUST ignore them.

**RTT (12 bits):** Round trip time to the remote DNS server for a UDP query, measured in units of 10 milliseconds.

**validationStatus (12 bits):** The result of the DNS UDP validation attempt. This field MUST be set to one of the DNS\_IP\_VALIDATE\_RETURN\_FLAGS (section [2.2.3.1.2](#)).

**Padding (20 bytes):** This can be any value and MUST be ignored.



### 2.2.3.2.3 DNS\_ADDR\_ARRAY

The DNS\_ADDR\_ARRAY structure is used to represent an array of DNS\_ADDR (section [2.2.3.2.2](#)) structures. The DNS server management protocol uses this structure to exchange lists of mixed IPv4 and IPv6 addresses between client and server.

```
typedef struct _DnsAddrArray {
    DWORD MaxCount;
    DWORD AddrCount;
    DWORD Tag;
    WORD Family;
    WORD WordReserved;
    DWORD Flags;
    DWORD MatchFlag;
    DWORD Reserved1;
    DWORD Reserved2;
    [size_is(AddrCount)] DNS_ADDR AddrArray[];
} DNS_ADDR_ARRAY,
*PDNS_ADDR_ARRAY;
```

**MaxCount:** The actual number of IP addresses that are present in the AddrArray member.

**AddrCount:** Must be set to the same value as MaxCount.

**Tag:** This field is unused. Senders MUST set the value to zero and receivers MUST ignore it.

**Family:** The family of addresses present in the array, such as AF\_INET or AF\_INET6. If this field is not specified, then addresses with all families may be present.

**WordReserved:** This field is unused. Senders MUST set the value to zero and receivers MUST ignore it.

**Flags:** This field is unused. Senders MUST set the value to zero and receivers MUST ignore it.

**MatchFlag:** This field is unused. Senders MUST set the value to zero and receivers MUST ignore it.

**Reserved1:** This field is unused. Senders MUST set the value to zero and receivers MUST ignore it.

**Reserved2:** This field is unused. Senders MUST set the value to zero and receivers MUST ignore it.

**AddrArray:** An array of DNS\_ADDR (section [2.2.3.2.2](#)) structures. The number of elements in this array is specified by the AddrCount member.

An empty DNS\_ADDR\_ARRAY is represented by **AddrCount** set to zero. Senders of an empty DNS\_ADDR\_ARRAY MUST set the other fields' values to zero (including a single entry in **AddrArray**, which is set to binary zeros), and receivers MUST ignore them.

### 2.2.3.2.4 DNS\_RPC\_IP\_VALIDATE

The DNS\_RPC\_IP\_VALIDATE structure is used to request that the DNS server validate a number of IP addresses. This can be used by clients to determine if an IP address is suitable for use as a DNS server in the context specified by the dwContext member (see below). This structure is to request IP

validation while processing the [R\\_DnssrvComplexOperation2 \(section 3.1.4.8\)](#) method call with operation type IpValidate.

```
typedef struct _DnsRpcIPValidate {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    DWORD dwContext;
    DWORD dwReserved1;
    [string] char* pszContextName;
    PDNS_ADDR_ARRAY aipValidateAddrs;
} DNS_RPC_IP_VALIDATE,
*PDNS_RPC_IP_VALIDATE;
```

**dwRpcStructureVersion:** The DNS management structure version number. This value MUST be set to 0x00000001.

**dwReserved0:** This field is unused. The client MUST set this value to zero and the server MUST ignore it.

**dwContext:** The context or purpose for which addresses present in aipValidateAddrs MUST be validated by the DNS server. This field MUST be set to one of the following values:

Value	Meaning
DNS_IPVAL_DNS_SERVERS 0x00000000	Validate that IP addresses are reachable and operational by the DNS servers.
DNS_IPVAL_DNS_ROOTHINTS 0x00000001	Validate that IP addresses are suitable as <b>root hints</b> .
DNS_IPVAL_DNS_FORWARDERS 0x00000002	Validate that IP addresses are server level forwarders.
DNS_IPVAL_DNS_ZONE_MASTERS 0x00000003	Validate that IP addresses are remote DNS servers hosting a zone, named as pointed to by pszContextName.
DNS_IPVAL_DNS_DELEGATIONS 0x00000004	Validate that IP addresses are remote DNS servers are name-server for the delegated zone, named as pointed to by <b>pszContextName</b> .

**dwReserved1:** This field is unused. The client MUST set this to zero and the server MUST ignore it.

**pszContextName:** A pointer to a null-terminated ASCII character string that specifies a zone name. The use of this zone name is specified by the dwContext member.

**aipValidateAddrs:** A pointer to a [DNS\\_ADDR\\_ARRAY structure \(section 2.2.3.2.3\)](#) contains a list of IP addresses to be validated by the DNS server.

## 2.2.4 Server Messages

### 2.2.4.1 Enumerations and Constants

#### 2.2.4.1.1 DNS\_BOOT\_METHODS

The DNS\_BOOT\_METHODS is an 8-bit integer value that specifies the sources of information from which the DNS server obtains information at boot time. Following are possible values for this:

Constant/value	Description
BOOT_METHOD_UNINITIALIZED 0x00	Server obtains the boot information, the list of zones to load, and populates its database in the following order, until successful: from a file-based persistent storage, or from the directory server, or from the persistent copy of the DNS Zone Table. <a href="#">&lt;26&gt;</a>
BOOT_METHOD_FILE 0x01	Server obtains boot information, the list of zones to load, and populates its database from a file based persistent storage.
BOOT_METHOD_REGISTRY 0x02	Server obtains boot information, the list of zones to load, and populates its database from the local persistent copy of the DNS Zone Table.
BOOT_METHOD_DIRECTORY 0x03	Server obtains boot information, the list of zones to load, and populates its database from the local persistent copy of the DNS Zone Table for zones located in local persistent storage, and from the directory server for zones persistently stored in the directory server.

#### 2.2.4.1.2 DNS\_NAME\_CHECK\_FLAGS

The DNS server enforces different levels of syntax checking for FQDNs. DNS\_NAME\_CHECK\_FLAGS is a DWORD value that specifies the configured level of syntax checking for FQDNs. DNS [\[RFC1034\]](#) and [\[RFC2181\]](#) requires that all FQDNs meet the following basic criteria:

- Total length no longer than 255 characters.
- Each label is less than 63 characters.
- No two consecutive "." characters.

The name check value MUST be set to one of the following allowed values to modify basic FQDN validity checking:

Constant/value	Description
DNS_ALLOW_RFC_NAMES_ONLY 0x00000000	The DNS server will accept FQDNs that only contain the ASCII characters "a-z", "A-Z", and "0-9". Names that begin with "." or contain two consecutive "." characters will be rejected. The name may contain zero or one "*" but this character must appear as the first character in the name if it is present.
DNS_ALLOW_NONRFC_NAMES 0x00000001	The DNS server will accept any printable ASCII character in an FQDN.
DNS_ALLOW_MULTIBYTE_NAMES 0x00000002	The DNS server will accept all characters specified above and also UTF-8 <a href="#">[RFC3629]</a> characters in FQDNs.

Constant/value	Description
DNS_ALLOW_ALL_NAMES 0x00000003	The DNS server will not restrict the set of characters that may appear in FQDNs.

## 2.2.4.2 Structures

### 2.2.4.2.1 DNSSRV\_VERSION

The DNSSRV\_VERSION is used to store detailed version information of the operating system running on the DNS server. This structure is used by the DNS server to populate the **dwVersion** field in the DNS\_RPC\_SERVER\_INFO structure (section [2.2.4.2.2](#)).

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
Service Pack Version																OS Minor Version					OS Major Version										

**Service Pack Version (2 bytes):** The implementation-specific revision number of the DNS server's operating system. [<27>](#)

**OS Minor Version (1 byte):** The minor OS version number for the DNS server.

**OS Major Version (1 byte):** The major OS version number for the DNS server.

### 2.2.4.2.2 DNS\_RPC\_SERVER\_INFO

The DNS\_RPC\_SERVER\_INFO structure contains information about the DNS server's configuration and state. There are different versions of the DNS\_RPC\_SERVER\_INFO structure. The DNS server MUST use one of the structures corresponding to the value of the **dwClientVersion** field in DNS Server Management Protocol method calls (section [3.1.4](#)) as shown in the following table. If the method call does not specify the value of **dwClientVersion**, the DNS\_RPC\_SERVER\_INFO\_W2K version of the structure MUST be used.

Value	Structure
0x00000000	DNS_RPC_SERVER_INFO_W2K (section <a href="#">2.2.4.2.2.1</a> )
0x00060000	DNS_RPC_SERVER_INFO_DOTNET (section <a href="#">2.2.4.2.2.2</a> )
0x00070000	DNS_RPC_SERVER_INFO_LONGHORN (section <a href="#">2.2.4.2.2.3</a> )

#### 2.2.4.2.2.1 DNS\_RPC\_SERVER\_INFO\_W2K

```
typedef struct _DnsRpcServerInfoW2K {
    DWORD dwVersion;
    UCHAR fBootMethod;
    BOOLEAN fAdminConfigured;
    BOOLEAN fAllowUpdate;
    BOOLEAN fDsAvailable;
    [string] char* pszServerName;
    [string] wchar_t* pszDsContainer;
    PIP4_ARRAY aipServerAddrs;
}
```

```

PIP4_ARRAY aipListenAddrs;
PIP4_ARRAY aipForwarders;
PDWORD pExtension1;
PDWORD pExtension2;
PDWORD pExtension3;
PDWORD pExtension4;
PDWORD pExtension5;
DWORD dwLogLevel;
DWORD dwDebugLevel;
DWORD dwForwardTimeout;
DWORD dwRpcProtocol;
DWORD dwNameCheckFlag;
DWORD cAddressAnswerLimit;
DWORD dwRecursionRetry;
DWORD dwRecursionTimeout;
DWORD dwMaxCacheTtl;
DWORD dwDsPollingInterval;
DWORD dwScavengingInterval;
DWORD dwDefaultRefreshInterval;
DWORD dwDefaultNoRefreshInterval;
DWORD dwReserveArray[10];
BOOLEAN fAutoReverseZones;
BOOLEAN fAutoCacheUpdate;
BOOLEAN fRecurseAfterForwarding;
BOOLEAN fForwardDelegations;
BOOLEAN fNoRecursion;
BOOLEAN fSecureResponses;
BOOLEAN fRoundRobin;
BOOLEAN fLocalNetPriority;
BOOLEAN fBindSecondaries;
BOOLEAN fWriteAuthorityNs;
BOOLEAN fStrictFileParsing;
BOOLEAN fLooseWildcarding;
BOOLEAN fDefaultAgingState;
BOOLEAN fReserveArray[15];
} DNS_RPC_SERVER_INFO_W2K,
*PDNS_RPC_SERVER_INFO_W2K;

```

**dwVersion:** The operating system version of the DNS server in DNSSRV\_VERSION (section [2.2.4.2.1](#)).

**fBootMethod:** The method by which the DNS server obtains information at the start time. This MUST be set to one of the possible values as specified in DNS\_BOOT\_METHODS (section [2.2.4.1.1](#)).

**fAdminConfigured:** A Boolean field that specifies whether the DNS server has been configured by an administrator. On a fresh installed server this value MUST be set to FALSE. This value MUST be set to TRUE whenever a zone is created, or a record is modified, or an **Active Directory domain controller promotion (DCPROMO)** configures the DNS server.

**fAllowUpdate:** A Boolean field that indicates whether the DNS server allows dynamic DNS updates. This field MUST be set to FALSE if the server does not allow dynamic zone-updates, otherwise set to TRUE.

**fDsAvailable:** A Boolean field that specifies whether a directory server is available to the DNS server. It MUST be set to FALSE, if the server does not have access to a directory server.

**pszServerName:** A pointer to a null-terminated UTF-8 string that contains the fully qualified domain name of the DNS server.

**pszDsContainer:** A pointer to a null-terminated Unicode string that points to the DNS server's container path as a **distinguished name** in the directory server. If no directory server is configured, this value MUST be set to NULL. This value is synthesized by the server by concatenating a constant container **relative distinguished name** and the result of an **LDAP** search operation to retrieve the "defaultNamingContext" of the **Active Directory** server's rootDSE. <28>

**aipServerAddrs:** The list of IP addresses that are available on the server.

**aipListenAddrs:** The list of IP addresses that are explicitly configured by the administrator on the DNS server on which server should listen for the DNS requests. If this value is set to NULL then the server listens to all available IP addresses.

**aipForwarders:** The list of remote DNS servers to which this DNS server will forward unresolved DNS requests.

**pExtension1:** Reserved for future use and MUST be ignored by receiver.

**pExtension2:** Reserved for future use and MUST be ignored by receiver.

**pExtension3:** Reserved for future use and MUST be ignored by receiver.

**pExtension4:** Reserved for future use and MUST be ignored by receiver.

**pExtension5:** Reserved for future use and MUST be ignored by receiver.

**dwLogLevel:** This indicates which DNS packets will be logged and how they will be logged. This field MUST be set to either zero or a combination (by bitwise OR) of the possible values as specified under DNS\_LOG\_LEVELS (section [2.2.9.1.1](#)). If this value is set to zero, then no logging will be performed for DNS packets.

**dwDebugLevel:** Unused. Receivers MUST ignore.

**dwForwardTimeout:** The time interval, in seconds, for which the DNS server waits for a response from each server in the forwarders list.

**dwRpcProtocol:** This value indicates what RPC protocols this DNS server will accept connections on. This value MUST be set to any combination of values specified in DNS\_RPC\_PROTOCOLS (section [2.2.1.1.2](#)).

**dwNameCheckFlag:** The level of domain name checking and validation enforced by the DNS server. This value MUST be set one of the allowed values that are specified in DNS\_NAME\_CHECK\_FLAGS (section [2.2.4.1.2](#)).

**cAddressAnswerLimit:** The configured value for the maximum number of type A IP address resource records that the DNS server can insert in the answer section of a response to a UDP query of type A. If this value is set to 0x00000000 then the DNS server MUST NOT enforce any artificial limit on number of records in a response and if response becomes larger than the DNS UDP packet size then the truncation bit MUST be set [\[RFC1035\]](#). If this property value is not 0x00000000 and the DNS server is unable to add the specified number of records to the response message due to message size limitations, it MUST return as many records as will fit in the message and the truncation bit MUST NOT be set. The DNS server MUST NOT enforce this limit if the query is not of type A. If the value of this property is not 0x00000000 the DNS server MUST enforce this limit for UDP queries and MUST NOT enforce this limit for TCP

queries. If the **LocalNetPriority** property value is set to TRUE then the DNS server must first order the address records as per the **LocalNetPriority** property and then MUST select the first **cAddressAnswerLimit** type A records in this sorted list for inclusion in the response. The value MUST be either zero or between 0x00000005 and 0x0000001C inclusive.

**dwRecursionRetry:** The time-interval, in seconds, for which the DNS server waits before it retries a recursive query to the remote DNS server from which it did not receive a response. The values MUST be between 1 and 15 seconds inclusive.

**dwRecursionTimeout:** The time-interval, in seconds, for which the DNS server waits for a recursive query-response from a remote DNS server. The values MUST be between 1 and 15 seconds inclusive.

**dwMaxCacheTtl:** The maximum time duration, in seconds, for which the DNS server will cache a resource record obtained from a remote server in a successful query response. The values for this MUST be between 0 to 30 days (but specified in seconds) inclusive.

**dwDsPollingInterval:** The interval, in seconds, at which the DNS server will poll a directory server to obtain updated information for any changes that may have occurred to zones loaded in the server. The values MUST be between 30 and 3600 seconds inclusive.

**dwScavengingInterval:** The **scavenging** interval, in hours, on the DNS server. This is the interval at which the server will execute the cleanup of stale DNS records. The value MUST be between 0 and 8760 hours (1 year). If this value is zero then scavenging is disabled.

**dwDefaultRefreshInterval:** The default value of the **Refresh interval**, in hours, for new zones created on the DNS server. For any **primary zone** created on the server by default this value is used as the Refresh interval.

**dwDefaultNoRefreshInterval:** The default value of the **NoRefresh interval**, in hours, for new zones created on the DNS server. For any primary zone created on the server by default this value is used as the NoRefresh interval.

**dwReserveArray:** This value is reserved for future use and MUST be ignored by the receiver. Senders MUST set this to zero and receivers MUST ignore it.

**fAutoReverseZones:** A Boolean value that indicates whether the DNS server is configured to automatically create standard reverse lookup zones at boot time.

**fAutoCacheUpdate:** A Boolean value that indicates whether the DNS server is configured to automatically write-back cached root hints and delegation data to persistent storage.

**fRecurseAfterForwarding:** A Boolean value that indicates whether the DNS server is configured to use recursion in addition to forwarding. If this value is TRUE (0x01) then if the DNS server does not have any forwarders configured or if forwarders are unreachable then it MUST return failure, otherwise it MUST perform normal recursive processing for this query as specified in section 4.3.1 [\[RFC1034\]](#).

**fForwardDelegations:** A Boolean value indicates whether or not the DNS server will forward queries about delegated subzones to servers outside of its authoritative zone. If this value is set to TRUE, then the DNS server forwards all name queries about delegated subzones to forwarding servers in other zones; otherwise it will send such queries within its authoritative zone to the corresponding subzone only.

**fNoRecursion:** A Boolean value that indicates whether the DNS server will perform recursive resolution for queries. If this value is TRUE then recursion MUST NOT be performed even if the

Recursion Desired (RD) bit (section 4.1.1 of [RFC1035](#)) is set in the DNS query packet header. If this value is FALSE then recursion will be performed as per [RFC1035](#).

**fSecureResponses:** A Boolean value that indicates if the DNS server SHOULD screen DNS records received in remote query responses against the zone of authority for the remote server to prevent cache pollution. If it is set to TRUE, then the DNS server will cache only those records that are in zone of authority for the remote server that was queried. otherwise when set to FALSE it will save all records in the cache.

**fRoundRobin:** A Boolean value that indicates if the DNS server is configured to rotate the order of DNS records it returns for a given name. If this value is set to FALSE no round robin will be performed and DNS records will be returned in static, arbitrary order.

**fLocalNetPriority:** A Boolean value that indicates if the DNS server is configured to prioritize address records in a response based on the IP address of the DNS client that submitted the query. If this is set to TRUE the DNS server will return address records in the order of their closeness to the querying client's IP address by applying the **net mask** pointed to by LocalNetPriorityNetMask. If this value is set to 0 then the DNS server returns records in the order in which they are obtained from the database.

**fBindSecondaries:** A Boolean value that indicates if the DNS server allows zone transfers to secondary DNS servers running older non-Microsoft software. If this value is set to TRUE the DNS server sends zone transfer to secondaries via a slower mechanism, with one resource record in each message.

**fWriteAuthorityNs:** A Boolean value that indicates if the DNS server is enabled to write NS records in the authority section of all successful authoritative responses. If this value is TRUE then NS records will be included in the authority section of responses, otherwise NS records will only be included in referral responses.

**fStrictFileParsing:** A Boolean value that indicates if the DNS server is configured to perform strict file parsing. When this value is set to TRUE and a record parsing error is detected server will quit after indicating error. If this value is FALSE parsing errors will cause that specific record to be ignored and the server will continue to load the rest of the database.

**fLooseWildcarding:** A Boolean value that indicates if the DNS server is configured to perform loose wildcarding [RFC1035](#), otherwise it returns FALSE. When a server does not find a resource record that matches the name and type specified in the query in the authoritative zone, then it searches for related wildcard records, (section 4.3.3 [RFC1034](#)), if configured to perform loose wildcarding will return the first node it finds that has matching resource-record type, whereas if it is not then it will return the first node that has any resource record.

**fDefaultAgingState:** A Boolean value that indicates if the default value of ageing state for new primary zones created on the DNS server. For any primary zone created on the server this value is used as its default aging state. If this is FALSE then timestamps of records in the zone will not be tracked whereas when this value is TRUE then the timestamps of records in the zone will be tracked.

**fReserveArray:** Reserved for future use. These values MUST be ignored by receiver.

#### 2.2.4.2.2.2 DNS\_RPC\_SERVER\_INFO\_DOTNET

All fields have same definition as specified in section DNS\_RPC\_SERVER\_INFO\_W2K (section [2.2.4.2.2.1](#)), with the following exceptions: [<29>](#)

```
typedef struct _DnsRpcServerInfoDotNet {
```



```

DWORD dwRpcStructureVersion;
DWORD dwReserved0;
DWORD dwVersion;
UCHAR fBootMethod;
BOOLEAN fAdminConfigured;
BOOLEAN fAllowUpdate;
BOOLEAN fDsAvailable;
[string] char* pszServerName;
[string] wchar_t* pszDsContainer;
PIP4_ARRAY aipServerAddr;
PIP4_ARRAY aipListenAddr;
PIP4_ARRAY aipForwarders;
PIP4_ARRAY aipLogFilter;
[string] wchar_t* pwszLogFilePath;
[string] char* pszDomainName;
[string] char* pszForestName;
[string] char* pszDomainDirectoryPartition;
[string] char* pszForestDirectoryPartition;
[string] char* pExtensions[6];
DWORD dwLogLevel;
DWORD dwDebugLevel;
DWORD dwForwardTimeout;
DWORD dwRpcProtocol;
DWORD dwNameCheckFlag;
DWORD cAddressAnswerLimit;
DWORD dwRecursionRetry;
DWORD dwRecursionTimeout;
DWORD dwMaxCacheTtl;
DWORD dwDsPollingInterval;
DWORD dwLocalNetPriorityNetMask;
DWORD dwScavengingInterval;
DWORD dwDefaultRefreshInterval;
DWORD dwDefaultNoRefreshInterval;
DWORD dwLastScavengeTime;
DWORD dwEventLogLevel;
DWORD dwLogFileMaxSize;
DWORD dwDsForestVersion;
DWORD dwDsDomainVersion;
DWORD dwDsDsaVersion;
DWORD dwReserveArray[4];
BOOLEAN fAutoReverseZones;
BOOLEAN fAutoCacheUpdate;
BOOLEAN fRecurseAfterForwarding;
BOOLEAN fForwardDelegations;
BOOLEAN fNoRecursion;
BOOLEAN fSecureResponses;
BOOLEAN fRoundRobin;
BOOLEAN fLocalNetPriority;
BOOLEAN fBindSecondaries;
BOOLEAN fWriteAuthorityNs;
BOOLEAN fStrictFileParsing;
BOOLEAN fLooseWildcarding;
BOOLEAN fDefaultAgingState;
BOOLEAN fReserveArray[15];
} DNS_RPC_SERVER_INFO_DOTNET,
*PDNS_RPC_SERVER_INFO_DOTNET;

```

**dwRpcStructureVersion:** The DNS management structure version number. The following are possible values:

Value	Meaning
0x00000001	Structure is of type DNS_SERVER_INFO_DOTNET.
0x00000002	Structure is of type DNS_SERVER_INFO_LONGHORN.

**dwReserved0:** This field is reserved for future use and it MUST be ignored by receiver.

**aipLogFilter:** The list of IP addresses used for debug log filtering. The DNS server will log DNS traffic that is sent to or received from for these IP addresses and will not log DNS traffic to or from other IP addresses. If this value is set to NULL then the DNS server will not perform IP filtering when logging DNS traffic.

**pszLogFilePath:** A pointer to a null-terminated Unicode string that contains an absolute pathname or relative pathname or filename for the operational log file on the DNS server. If this value is set to NULL then the log SHOULD be logged to an implementation specific log file.

**pszDomainName:** A pointer to a null-terminated UTF-8 string that contains the name of the directory server domain to which the DNS server belongs if directory server is available. This value will be NULL if no directory server is available.

**pszForestName:** A pointer to a null-terminated UTF-8 string that contains the name of the directory server **forest** to which the DNS server belongs if directory server is available. This value will be NULL if no directory server is available.

**pszDomainDirectoryPartition:** A pointer to a null-terminated UTF-8 string that contains the base name for the domain wide DNS application directory partition.

**pszForestDirectoryPartition:** A pointer to a null-terminated UTF-8 string that contains the base name for the forest wide DNS application directory partition.

**pExtensions:** Reserved for future use and MUST be ignored by receiver.

**dwLocalNetPriorityNetMask:** The net mask used by the DNS server to prioritize address records in responses when the server is configured to enforce LocalNetPriority as mentioned above.

**dwLastScavengeTime:** The timestamp at which the last scavenging cycle was executed on the DNS server. If this value is set to 0 then no scavenging cycle has been run since the server was last started.

**dwEventLogLevel:** This value indicates what level of events will be logged by the DNS server. This value MUST be set to one of the combination of the possible values for this defined in DNS\_EVENTLOG\_TYPES (section [2.2.9.1.2](#)).

**dwLogFileMaxSize:** The maximum allowed size, in bytes, for the log file.

**dwDsForestVersion:** This value indicates the directory server forest version being used by the DNS server, stored in the ForceForestBehaviorVersion property.

**dwDsDomainVersion:** This value indicates the directory server domain version being used by the **DNS** server, stored in the ForceDomainBehaviorVersion property.

**dwDsDsaVersion:** This value indicates the directory server local server version being used by the DNS server, stored in the ForceDsaBehaviorVersion property.

### 2.2.4.2.2.3 DNS\_RPC\_SERVER\_INFO\_LONGHORN

All fields have same definition as specified in section DNS\_RPC\_SERVER\_INFO\_DOTNET (section [2.2.4.2.2.2](#) ), with the following exceptions: <30>

```
typedef struct _DnsRpcServerInfoLonghorn {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    DWORD dwVersion;
    UCHAR fBootMethod;
    BOOLEAN fAdminConfigured;
    BOOLEAN fAllowUpdate;
    BOOLEAN fDsAvailable;
    [string] char* pszServerName;
    [string] wchar_t* pszDsContainer;
    PDNS_ADDR_ARRAY aipServerAddrs;
    PDNS_ADDR_ARRAY aipListenAddrs;
    PDNS_ADDR_ARRAY aipForwarders;
    PDNS_ADDR_ARRAY aipLogFilter;
    [string] wchar_t* pwszLogFilePath;
    [string] char* pszDomainName;
    [string] char* pszForestName;
    [string] char* pszDomainDirectoryPartition;
    [string] char* pszForestDirectoryPartition;
    [string] char* pExtensions[6];
    DWORD dwLogLevel;
    DWORD dwDebugLevel;
    DWORD dwForwardTimeout;
    DWORD dwRpcProtocol;
    DWORD dwNameCheckFlag;
    DWORD cAddressAnswerLimit;
    DWORD dwRecursionRetry;
    DWORD dwRecursionTimeout;
    DWORD dwMaxCacheTtl;
    DWORD dwDsPollingInterval;
    DWORD dwLocalNetPriorityNetMask;
    DWORD dwScavengingInterval;
    DWORD dwDefaultRefreshInterval;
    DWORD dwDefaultNoRefreshInterval;
    DWORD dwLastScavengeTime;
    DWORD dwEventLogLevel;
    DWORD dwLogFileMaxSize;
    DWORD dwDsForestVersion;
    DWORD dwDsDomainVersion;
    DWORD dwDsDsaVersion;
    BOOLEAN fReadOnlyDC;
    DWORD dwReserveArray[3];
    BOOLEAN fAutoReverseZones;
    BOOLEAN fAutoCacheUpdate;
    BOOLEAN fRecurseAfterForwarding;
    BOOLEAN fForwardDelegations;
    BOOLEAN fNoRecursion;
    BOOLEAN fSecureResponses;
    BOOLEAN fRoundRobin;
    BOOLEAN fLocalNetPriority;
```

```

BOOLEAN fBindSecondaries;
BOOLEAN fWriteAuthorityNs;
BOOLEAN fStrictFileParsing;
BOOLEAN fLooseWildcarding;
BOOLEAN fDefaultAgingState;
BOOLEAN fReserveArray[15];
} DNS_RPC_SERVER_INFO_LONGHORN,
*PDNS_RPC_SERVER_INFO_LONGHORN,
DNS_RPC_SERVER_INFO,
*PDNS_RPC_SERVER_INFO;

```

**fReadOnlyDC:** A [Boolean](#) value that indicates whether the DNS server has access to a directory server that is running in read-only mode, that is, whether the server does not accept directory server write operations. The DNS server detects whether this is the case by reading the **supportedCapabilities** attribute of the server's "rootDse" object, looking for LDAP\_CAP\_ACTIVE\_DIRECTORY\_PARTIAL\_SECRETS\_OID. (See [\[MS-ADTS\]](#), section [3.1.1.3.2.10](#).)

## 2.2.5 Zone Messages

### 2.2.5.1 Enumerations and Constants

#### 2.2.5.1.1 DNS\_ZONE\_TYPE

DNS\_ZONE\_TYPE is an 8-bit integer value that specifies the type of a zone. An implementation SHOULD support all values below. [<31>](#)

Constant/value	Description
DNS_ZONE_TYPE_CACHE 0x00	This zone is used to store all cached DNS records received from remote DNS servers during normal query processing.
DNS_ZONE_TYPE_PRIMARY 0x01	The DNS server is a primary DNS server for this zone.
DNS_ZONE_TYPE_SECONDARY 0x02	The DNS server is acting as a secondary DNS server for this zone.
DNS_ZONE_TYPE_STUB 0x03	Zone is a <b>stub zone</b> , that is, it contains only those resource records that are necessary to identify authoritative DNS servers for that zone.
DNS_ZONE_TYPE_FORWARDER 0x04	The DNS server is a forwarder for this zone, that is, the server does not have authoritative information for resource records in this zone.
DNS_ZONE_TYPE_SECONDARY_CACHE 0x05	This zone is used to hold cached records for some implementation specific purpose.

#### 2.2.5.1.2 DNS\_ZONE\_SECONDARY\_SECURITY

DNS\_ZONE\_SECONDARY\_SECURITY is a 32-bit integer value that enumerates the types of security settings that are enforced by the master DNS server to honor zone transfer requests for this zone.

Constant/value	Description
ZONE_SECSECURE_NO_SECURITY 0x00000000	No security enforcement for secondaries, that is, any zone transfer request will be honored.
ZONE_SECSECURE_NS_ONLY 0x00000001	Zone transfer request will be honored from the remote servers, which are in the list of name servers for this zone.
ZONE_SECSECURE_LIST_ONLY 0x00000002	Zone transfer request will be honored from the remote servers, which are explicitly configured by IP addresses in the <b>aiPSecondaries</b> field in the DNS_RPC_ZONE_INFO structure (section <a href="#">2.2.5.2.4</a> ).
ZONE_SECSECURE_NO_XFER 0x00000003	No zone transfer requests will be honored.

### 2.2.5.1.3 DNS\_ZONE\_NOTIFY\_LEVEL

DNS\_ZONE\_NOTIFY\_LEVEL is a DWORD value that enumerates the levels of notification settings that can be configured on a master DNS server to send out notifications to secondaries about any changes to this zone, so that they can initiate a zone transfer to get updated zone information.

Constant/value	Description
ZONE_NOTIFY_OFF 0x00000000	The Master DNS server does not send any zone notifications.
ZONE_NOTIFY_ALL_SECONDARIES 0x00000001	The Master DNS server sends zone notifications to all secondary servers for this zone, either they are listed as name-servers for this zone or they are present explicitly zone <b>notify</b> list for this zone.
ZONE_NOTIFY_LIST_ONLY 0x00000002	The Master DNS server sends zone notifications only to those remote servers which are explicitly configured by IP addresses in the <b>aiPNotify</b> field in the DNS_RPC_ZONE_INFO structure (section <a href="#">2.2.5.2.4</a> ).

### 2.2.5.1.4 ZONE\_REQUEST\_FILTERS

ZONE\_REQUEST\_FILTERS is a 32-bit integer value that specifies possible selection filter types for zone selection. An implementation SHOULD support all values. [<32>](#)

Constant/value	Description
ZONE_REQUEST_PRIMARY 0x00000001	Consider primary zones.
ZONE_REQUEST_SECONDARY 0x00000002	Consider <b>secondary zones</b> .
ZONE_REQUEST_CACHE 0x00000004	Consider cache zones.
ZONE_REQUEST_AUTO 0x00000008	Consider the <b>autocreated zones</b> . These are zones with the AutoCreated flag (section <a href="#">2.2.5.2.2</a> ) turned on.

Constant/value	Description
ZONE_REQUEST_FORWARD 0x00000010	Consider zones whose names do not end with the "arpa" label.
ZONE_REQUEST_REVERSE 0x00000020	Consider zones whose names end with the "arpa" label.
ZONE_REQUEST_FORWARDER 0x00000040	Consider forwarder zones.
ZONE_REQUEST_STUB 0x00000080	Consider stub zones.
ZONE_REQUEST_DS 0x00000100	Consider zones that are directory server integrated.
ZONE_REQUEST_NON_DS 0x00000200	Consider zones that are not directory server integrated (that is, zones that are either persistently stored in local storage or are zones of type DNS_ZONE_TYPE_CACHE or DNS_ZONE_TYPE_SECONDARY_CACHE (section <a href="#">2.2.5.1.1</a> )).
ZONE_REQUEST_DOMAIN_DP 0x00000400	Consider zones that are stored in the domain application directory partition.
ZONE_REQUEST_FOREST_DP 0x00000800	Consider zones that are stored in the forest application directory partition.
ZONE_REQUEST_CUSTOM_DP 0x00001000	Consider zones that are stored in a custom application directory partition, where the application directory partition name is specified by <b>pszPartitionFqdn</b> in the DNS_RPC_ENUM_ZONES_FILTER (section <a href="#">2.2.5.2.9</a> ) structure.
ZONE_REQUEST_LEGACY_DP 0x00002000	Consider zones that are stored in the default application directory partition.

### 2.2.5.1.5 ZONE\_SKD\_ROLLOVER\_TYPE

ZONE\_SKD\_ROLLOVER\_TYPE is a 32-bit integer value that specifies possible **key rollover** types for a signing key descriptor. An implementation SHOULD support all values. [<33>](#)

Constant/value	Description
DNS_ROLLOVER_TYPE_PREPUBLISH 0x00000000	A method of key rollover in which the new key is published in the zone before it will be used. See <a href="#">[RFC4641]</a> section 4.2.
DNS_ROLLOVER_TYPE_DOUBLE_SIGNATURE 0x00000001	A method of key rollover in which data is signed by both old and new keys simultaneously for a period of time. See <a href="#">[RFC4641]</a> section 4.2.

### 2.2.5.1.6 ZONE\_SKD\_ROLLOVER\_ACTION

ZONE\_SKD\_ROLLOVER\_ACTION is a 32-bit integer value that specifies possible key rollover actions for a signing key descriptor. An implementation SHOULD support all values. [<34>](#)

Constant/value	Description
DNS_ROLLOVER_ACTION_DEFAULT 0x00000000	The server MUST never send this value. The client MUST send this value to indicate that no change to the current key rollover behavior is desired.
DNS_ROLLOVER_ACTION_NORMAL 0x00000001	The server will perform a normal key rollover the next time the keys for this signing key descriptor are rolled over.
DNS_ROLLOVER_ACTION_REVOKE_STANDBY 0x00000002	The server will revoke the standby key for this signing key descriptor as defined by <a href="#">[RFC5011]</a> the next time the keys for this signing key descriptor are rolled over. This rollover action applies only to signing key descriptors representing <b>key signing keys</b> (those whose "fIsKSK" property as described in section <a href="#">2.2.6.2.1</a> is set to 0x00000001).
DNS_ROLLOVER_ACTION_RETIRE 0x00000003	The server will retire this signing key descriptor and remove all signatures associated with it the next time the keys for this signing key descriptor are rolled over.

## 2.2.5.2 Structures

### 2.2.5.2.1 DNS\_RPC\_ZONE

The DNS\_RPC\_ZONE structure contains basic information about a zone present on the DNS server. There are different versions of the DNS\_RPC\_ZONE structure. The DNS server MAY [<35>](#) decide to use one of these structures depending upon the value of the **dwClientVersion** field in DNS Server Management Protocol method calls (section [3.1.4](#)) as follows in the table provided. If the method call does not specify the value of **dwClientVersion**, the DNS\_RPC\_ZONE\_W2K version of the structure MUST be used.

Value	Structure
0x00000000	DNS_RPC_ZONE_W2K (section <a href="#">2.2.5.2.1.1</a> )
0x00060000 or 0x00070000	DNS_RPC_ZONE_DOTNET (section <a href="#">2.2.5.2.1.2</a> )

#### 2.2.5.2.1.1 DNS\_RPC\_ZONE\_W2K

```
typedef struct _DnsSrvRpcZoneW2K {
    [string] wchar_t* pszZoneName;
    DNS_RPC_ZONE_FLAGS Flags;
    UCHAR ZoneType;
    UCHAR Version;
} DNS_RPC_ZONE_W2K,
*PDNS_RPC_ZONE_W2K;
```

**pszZoneName:** A pointer to a null-terminated Unicode string that contains zone-name.

**Flags:** Zone flags as specified in section [2.2.5.2.2](#).

**ZoneType:** The zone type. This MUST be set to one of the allowed DNS\_ZONE\_TYPE section [2.2.5.1.1](#).

**Version:** The RPC protocol version. It MUST be set to 0x32.

### 2.2.5.2.1.2 DNS\_RPC\_ZONE\_DOTNET

All fields have same definition as specified in section DNS\_RPC\_ZONE\_W2K (section [2.2.5.2.1.1](#)), with the following exceptions:

```
typedef struct _DnssrvRpcZoneDotNet {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [string] wchar_t* pszZoneName;
    DNS_RPC_ZONE_FLAGS Flags;
    UCHAR ZoneType;
    UCHAR Version;
    DWORD dwDpFlags;
    [string] char* pszDpFqdn;
} DNS_RPC_ZONE_DOTNET,
*PDNS_RPC_ZONE_DOTNET,
DNS_RPC_ZONE,
*PDNS_RPC_ZONE;
```

**dwRpcStructureVersion:** The DNS management structure version number. This value MUST be set to 0x00000001.

**dwReserved0:** This field is reserved for future use. Senders set this to an arbitrary value and receivers MUST ignore it.

**dwDpFlags:** Application directory partition flags for this zone. This MUST be set to one of the combination of the DNS\_RPC\_DP\_FLAGS (section [2.2.7.1.1](#)). If this zone is not stored in the directory server this value MUST be 0x00000000.

**pszDpFqdn:** A pointer to a null-terminated UTF-8 string that specifies the fully qualified domain name of the application directory partition in which this zone is stored in the directory server. If this zone is not stored in the directory server this value MUST be NULL.

If the DNS RPC client sends an older version of DNS\_RPC\_ZONE structure, that is, DNS\_RPC\_ZONE\_W2K (section [2.2.5.2.1.1](#)), then the DNS RPC server MUST construct a current version of DNS\_RPC\_ZONE structure, that is, DNS\_RPC\_ZONE\_DOTNET, using the following steps:

1. Copy the same value for fields that are common to input and the current version of the DNS\_RPC\_ZONE structures.
2. The dwRpcStructureVersion field MUST be set to "1".
3. All other fields that are defined only in DNS\_RPC\_ZONE\_DOTNET and are not defined in DNS\_RPC\_ZONE (section [2.2.5.2.1](#)), MUST be set to "0".

### 2.2.5.2.2 DNS\_RPC\_ZONE\_FLAGS

DNS\_RPC\_ZONE\_FLAGS is a bit-mask value used by the DNS\_RPC\_ZONE structure (section [2.2.5.2.1](#)) to indicate zone properties and state.



This type is declared as follows:

```
typedef DWORD DNS_RPC_ZONE_FLAGS, *PDNS_RPC_ZONE_FLAGS;
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	B	C	D	E	F	G	H	I																							

**Figure 5: DNS\_RPC\_ZONE\_FLAGS Bit-Mask**

**A (Paused):** If set to 1, indicates that this zone is currently administratively paused. The DNS server will not use this zone to respond to queries, will not accept updates in this zone, and will suspend all other functions related to this zone. The default value for this flag is 0.

**B (Shutdown):** If set to 1, indicates that an error occurred which caused the DNS server to be unable to load a complete copy of the zone into memory. For primary zones, the DNS server MUST set this flag to 1 if an error occurred while loading the zone or its records from persistent storage. If the zone is directory server-integrated, the DNS server MUST retry loading the zone at an interval specified by the "DsPollingInterval" property (section [3.1.1.1](#)) and set this flag to zero if a subsequent loading attempt succeeds. If the server is not directory server-integrated the DNS server MUST NOT automatically retry loading the zone. For secondary and stub zones, the DNS server MUST set this flag to 1 if zone transfer was refused by all of the master servers or if zone transfer could not be successfully completed. The DNS server MUST retry zone transfer as specified by [\[RFC1034\]](#) and set this flag to zero if a subsequent zone transfer attempt succeeds. The default value of this field MUST be 1 and the value MUST be set to zero if the zone is successfully loaded during initialization (section [3.1.3](#)).

**C (Reverse):** If set to 1, indicates that this is a reverse lookup zone.

**D (AutoCreated):** If set to 1, indicates that zone was auto-created. A DNS server MAY automatically create certain zones at boot time which are flagged as "AutoCreated". Such zones are never written to persistent storage, and the DNS Server MUST NOT perform any DNS Server Management Protocol RPC operations on such zones. [<36>](#)

**E (DsIntegrated):** If set to 1, indicates that zone is stored in a directory server.

**F (Aging):** If set to 1, indicates that zone has aging enabled for resource records.

**G (Update):** The type of updates that are supported on this zone. This value MUST be set to one of the possible value from DNS\_ZONE\_UPDATE (section [2.2.6.1.1](#)).

**H (ReadOnly):** If set to 1, indicates that this zone is operating in read-only mode. The DNS server SHOULD support read-only zones. [<37>](#) If the DNS server supports read-only zones, it MUST set this flag to 1 if the zone is loaded from a directory server that is read-only, and in all other cases it MUST set the flag to 0.

**I (Unused):** MUST be set to zero when sent and MUST be ignored on receipt.

### 2.2.5.2.3 DNS\_RPC\_ZONE\_LIST

The DNS\_RPC\_ZONE\_LIST structure contains the information about a list of zones present on the DNS server. There are different versions of the DNS\_RPC\_ZONE\_LIST structure. The DNS server SHOULD [<38>](#) use one of these structures depending upon the passed-in value for the **dwClientVersion** field in DNS Server Management Protocol method calls (section [3.1.4](#)) as shown in the following table. If the method call does not specify the value of **dwClientVersion**, the DNS\_RPC\_ZONE\_LIST\_W2K version of the structure MUST be used.

Value	Structure
0x00000000	DNS_RPC_ZONE_LIST_W2K (section <a href="#">2.2.5.2.3.1</a> )
0x00060000	DNS_RPC_ZONE_LIST_DOTNET (section <a href="#">2.2.5.2.3.2</a> )
0x00070000	DNS_RPC_ZONE_LIST_DOTNET (section <a href="#">2.2.5.2.3.2</a> )

#### 2.2.5.2.3.1 DNS\_RPC\_ZONE\_LIST\_W2K

```
typedef struct _DnssrvRpcZoneListW2K {
    [range(0,500000)] DWORD dwZoneCount;
    [size_is(dwZoneCount)] PDNS_RPC_ZONE_W2K ZoneArray[];
} DNS_RPC_ZONE_LIST_W2K,
*PDNS_RPC_ZONE_LIST_W2K;
```

**dwZoneCount:** The number of zones present in the array of zones pointed to by ZoneArray.

**ZoneArray:** An array of structures of type DNS\_RPC\_ZONE (section [2.2.5.2.1.1](#)). Each element of the array represents one zone.

#### 2.2.5.2.3.2 DNS\_RPC\_ZONE\_LIST\_DOTNET

All fields have same definition as specified in section DNS\_RPC\_ZONE\_LIST\_W2K (section [2.2.5.2.3.1](#)), with the following exceptions:

```
typedef struct _DnssrvRpcZoneListDotNet {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [range(0,500000)] DWORD dwZoneCount;
    [size_is(dwZoneCount)] PDNS_RPC_ZONE_DOTNET ZoneArray[];
} DNS_RPC_ZONE_LIST_DOTNET,
*PDNS_RPC_ZONE_LIST_DOTNET,
DNS_RPC_ZONE_LIST,
*PDNS_RPC_ZONE_LIST;
```

**dwRpcStructureVersion:** The DNS management structure version number. This MUST be set to 0x00000001.

**dwReserved0:** This field is reserved for future use. Senders can send an arbitrary value and receivers MUST ignore it.

If the DNS RPC client sends an older version of DNS\_RPC\_ZONE\_LIST structure, that is, DNS\_RPC\_ZONE\_LIST\_W2K (section [2.2.5.2.3.1](#)), then the DNS RPC server MUST construct a

current version of DNS\_RPC\_ZONE\_LIST structure, that is, DNS\_RPC\_ZONE\_LIST\_DOTNET, using the following steps:

1. Copy the same value for the fields that are common to input and the current version of DNS\_RPC\_ZONE\_LIST structures.
2. The dwRpcStructureVersion field MUST be set to "1".
3. The value for the ZoneArray field MUST be obtained from the input structure as DNS\_RPC\_ZONE\_W2K (section [2.2.5.2.1.1](#)) array elements and each MUST be converted using the steps specified in section DNS\_RPC\_ZONE\_DOTNET (section [2.2.5.2.1.2](#)), and then assigned to the ZoneArray field in the DNS\_RPC\_ZONE\_LIST\_DOTNET structure.
4. All other fields that are defined only in DNS\_RPC\_ZONE\_LIST\_DOTNET and are not defined in DNS\_RPC\_ZONE\_LIST\_W2K (section [2.2.5.2.3.1](#)), MUST be set to "0".

## 2.2.5.2.4 DNS\_RPC\_ZONE\_INFO

The DNS\_RPC\_ZONE\_INFO structure contains the detailed information about a zone present on the DNS server. The DNS server uses this structure to return information about a zone while responding to R\_DnsrvcQuery2 (section [3.1.4.7](#)) method calls with operation type "ZoneInfo". There are different versions of DNS\_RPC\_ZONE\_INFO. The DNS server MUST use the structures corresponding to the passed-in value for the **dwClientVersion** field in DNS Server Management Protocol method calls (section [3.1.4](#)) in the following table, or if the method call does not specify the value of **dwClientVersion**, the DNS\_RPC\_ZONE\_INFO\_W2K version of the structure MUST be used.

Value	Structure
0x00000000	DNS_RPC_ZONE_INFO_W2K (section <a href="#">2.2.5.2.4.1</a> )
0x00060000	DNS_RPC_ZONE_INFO_DOTNET (section <a href="#">2.2.5.2.4.2</a> )
0x00070000	DNS_RPC_ZONE_INFO_LONGHORN (section <a href="#">2.2.5.2.4.3</a> )

### 2.2.5.2.4.1 DNS\_RPC\_ZONE\_INFO\_W2K

```
typedef struct _DnsRpcZoneInfoW2K {
    [string] char* pszZoneName;
    DWORD dwZoneType;
    DWORD fReverse;
    DWORD fAllowUpdate;
    DWORD fPaused;
    DWORD fShutdown;
    DWORD fAutoCreated;
    DWORD fUseDatabase;
    [string] char* pszDataFile;
    PIP4_ARRAY aipMasters;
    DWORD fSecureSecondaries;
    DWORD fNotifyLevel;
    PIP4_ARRAY aipSecondaries;
    PIP4_ARRAY aipNotify;
    DWORD fUseWins;
    DWORD fUseNbstat;
    DWORD fAging;
    DWORD dwNoRefreshInterval;
    DWORD dwRefreshInterval;
    DWORD dwAvailForScavengeTime;
}
```

```

PIP4_ARRAY aipScavengeServers;
DWORD pvReserved1;
DWORD pvReserved2;
DWORD pvReserved3;
DWORD pvReserved4;
} DNS_RPC_ZONE_INFO_W2K,
*PDNS_RPC_ZONE_INFO_W2K;

```

**pszZoneName:** A pointer to a null-terminated Unicode string that contains a zone name.

**dwZoneType:** The zone type. This MUST be set to one of the allowed values as specified in DNS\_ZONE\_TYPE (section [2.2.5.1.1](#)).

**fReverse:** A Boolean value where TRUE (0x00000001) indicates this is a reverse lookup zone and FALSE (0x00000000) indicates this is a forward lookup zone.

**fAllowUpdate:** A value that indicates what kind **dynamic updates** are allowed for this zone. This must be set to one of the following values:

Constant/value	Description
ZONE_UPDATE_OFF 0x00000000	No updates are allowed for the zone.
ZONE_UPDATE_UNSECURE0x00000001	All updates (secure and unsecure) are allowed for the zone.
ZONE_UPDATE_SECURE0x00000002	The zone only allows secure updates, that is, DNS packet MUST have a TSIG <a href="#">RFC2845</a> present in the additional section.

**fPaused:** A Boolean value indicates whether zone operations are currently paused. TRUE indicates that the DNS server does not use this zone to answer queries or accept updates. FALSE indicates that the zone is handled normally.

**fShutdown:** A Boolean value that indicates whether this zone is currently shut down.

**fAutoCreated:** A Boolean value that indicates whether this zone was autocreated by the DNS server at boot time.

**fUseDatabase:** A Boolean value that indicates whether this zone is stored in a directory server.

**pszDataFile:** A pointer to a null-terminated UTF-8 character string that specifies the name (with no path) of the zone file for a file-based zone or NULL if this zone is not stored in a file.

**aipMasters:** A pointer to a structure of type IP4\_ARRAY (section [2.2.3.2.1](#)) that specifies a list of IPv4 addresses of the remote DNS servers that can be sources of information for this zone to perform zone transfers by a secondary. This value is applicable for secondary, stub and forwarder zones only and MUST be NULL for all other zone types.

**fSecureSecondaries:** The secondary security settings configured for a zone on the master DNS server. The DNS server MUST respond to zone transfer requests from a secondary server according to the behavior description corresponding to this flag's value as specified in DNS\_ZONE\_SECONDARY\_SECURITY (section [2.2.5.1.2](#)). This value MUST be set to one of the allowed values as specified in DNS\_ZONE\_SECONDARY\_SECURITY (section [2.2.5.1.2](#)).

**fNotifyLevel:** A value that specifies the settings for sending zone notifications for a zone from the master DNS server. This value MUST be set to one of the allowed values as specified in DNS\_ZONE\_NOTIFY\_LEVEL (section [2.2.5.1.3](#)).

**aipSecondaries:** A pointer to a structure of type IP4\_ARRAY (section [2.2.3.2.1](#)) that specifies a list of IPv4 addresses of the remote DNS servers that are secondary DNS servers for this zone, or NULL if there are no secondary DNS servers. If fSecureSecondaries is set to ZONE\_SECSECURE\_LIST\_ONLY then only zone transfers from IP addresses in this list will be honored.

**aipNotify:** A pointer to a structure of type IP4\_ARRAY (section [2.2.3.2.1](#)) that specifies a list of IPv4 addresses of the remote DNS servers that are secondaries for this zone, for which this DNS server is acting as master and the DNS server will send zone notifications to these secondary servers, as directed by the value of fNotifyLevel above.

**fUseWins:** A Boolean value that indicates whether WINS resource record lookup is enabled for this forward lookup zones.

**fUseNbstat:** A Boolean value that indicates whether WINS-R resource record lookup is enabled for this reverse lookup zones.

**fAging:** A Boolean value where TRUE (0x00000001) indicates that aging is enabled for resource records in this zone, so the time-stamps of records in the zone will be updated when server receives dynamic update request for that record; whereas FALSE(0x00000000) indicates, that the time-stamps of the records in the zone will not be updated.

**dwNoRefreshInterval:** The time interval, in hours, that is configured as No Refresh interval value for this zone. This value determines the time interval between the last update of a record's time-stamp and the earliest instance when that time-stamp can be refreshed.

**dwRefreshInterval:** The time interval, in hours, that is configured as the Refresh interval value for this zone. Records that have not been refreshed by the expiration of this interval are eligible to be removed during the next scavenging cycle performed by the DNS server.

**dwAvailForScavengeTime:** The time interval, in hours, that is available before the scheduled next scavenging cycle for this zone.

**aipScavengeServers:** A pointer to a structure of type IP4\_ARRAY (section [2.2.3.2.1](#)) that specifies a list of IPv4 addresses of the DNS servers that will perform scavenging for this zone. This value is applicable for zones of type DNS\_ZONE\_TYPE\_PRIMARY (section DNS\_ZONE\_TYPE) only. If this value is NULL then there are no restrictions on which DNS server may perform scavenging for this zone.

**pvReserved1:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**pvReserved2:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**pvReserved3:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**pvReserved4:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

## 2.2.5.2.4.2 DNS\_RPC\_ZONE\_INFO\_DOTNET

All fields have same definition as specified in section DNS\_RPC\_ZONE\_INFO\_W2K (section [2.2.5.2.4.1](#)), with the following exceptions:

```
typedef struct _DnsRpcZoneInfoDotNet {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [string] char* pszZoneName;
    DWORD dwZoneType;
    DWORD fReverse;
    DWORD fAllowUpdate;
    DWORD fPaused;
    DWORD fShutdown;
    DWORD fAutoCreated;
    DWORD fUseDatabase;
    [string] char* pszDataFile;
    PIP4_ARRAY aipMasters;
    DWORD fSecureSecondaries;
    DWORD fNotifyLevel;
    PIP4_ARRAY aipSecondaries;
    PIP4_ARRAY aipNotify;
    DWORD fUseWins;
    DWORD fUseNstat;
    DWORD fAging;
    DWORD dwNoRefreshInterval;
    DWORD dwRefreshInterval;
    DWORD dwAvailForScavengeTime;
    PIP4_ARRAY aipScavengeServers;
    DWORD dwForwarderTimeout;
    DWORD fForwarderSlave;
    PIP4_ARRAY aipLocalMasters;
    DWORD dwDpFlags;
    [string] char* pszDpFqdn;
    [string] wchar_t* pwszZoneDn;
    DWORD dwLastSuccessfulSoaCheck;
    DWORD dwLastSuccessfulXfr;
    DWORD dwReserved1;
    DWORD dwReserved2;
    DWORD dwReserved3;
    DWORD dwReserved4;
    DWORD dwReserved5;
    [string] char* pReserved1;
    [string] char* pReserved2;
    [string] char* pReserved3;
    [string] char* pReserved4;
} DNS_RPC_ZONE_INFO_DOTNET,
*PDNS_RPC_ZONE_INFO_DOTNET;
```

**dwRpcStructureVersion:** The DNS server management structure version number. This value SHOULD be set to 0x00000001. [<39>](#)

**dwReserved0:** Reserved for future use. Server will set to zero and receiver MUST ignore this value.

**dwForwarderTimeout:** (4 bytes): The forwarder timeout value for a zone, in seconds. This is the number of seconds the DNS server should wait for response from a remote DNS server for

a forwarded query. This value is applicable for zones of type `DNS_ZONE_TYPE_FORWARDER` (section [2.2.5.1.1](#)). For all other zone types, senders MUST set this value to zero and receivers MUST ignore it.

**fForwarderSlave:** A Boolean value indicating whether the DNS server is not allowed to perform recursion while resolving names for this zone. The DNS server MUST fail queries after getting failure from all forwarded servers, if the value is `TRUE(0x00000001)`. Otherwise normal recursion MUST be performed. This value is applicable for zones of type `DNS_ZONE_TYPE_FORWARDER` (section [2.2.5.1.1](#)). For all other zone types, senders MUST set this value to zero and receivers MUST ignore it.

**aipLocalMasters:** A pointer to a structure of type `IP4_ARRAY` (section [2.2.3.2.1](#)) that specifies a list of IPv4 addresses of primary DNS servers for this zone. If this value is not `NULL` then it overrides the master servers list configured in the directory server.

**dwDpFlags:** Flag value that specifies information about the application directory partition in which this zone is stored. This MUST be set to any combination of the `DNS_RPC_DP_FLAGS` (section [2.2.7.1.1](#)) or zero if this zone is not stored in a directory server.

**pszDpFqdn:** A pointer to a null-terminated UTF-8 string that specifies the FQDN of the application directory partition in which this zone is stored. If the zone is not stored in an application directory partition this value MUST be `NULL`.

**pwszZoneDn:** Pointer to a null-terminated Unicode string that specifies the distinguished name for the zone if the zone is stored the directory server. This field is applicable for directory server integrated zones only. The value MUST be `NULL` if this zone is not stored in the directory server.

**dwLastSuccessfulSoaCheck:** The time-stamp at which last SOA record was received successfully from the primary DNS server for this zone. This field is applicable only for zones which are secondary or non-authoritative. For all other zones this value MUST be set to zero by senders and MUST be ignored by receivers.

**dwLastSuccessfulXfr:** The time-stamp at which last zone transfer was completed successfully from the primary DNS server for this zone. This field is applicable only for zones which are secondary or non-authoritative. For all other zones this value MUST be set to zero by senders and MUST be ignored by receivers.

**dwReserved1:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**dwReserved2:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**dwReserved3:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**dwReserved4:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**dwReserved5:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**pReserved1:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**pReserved2:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**pReserved3:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**pReserved4:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

#### 2.2.5.2.4.3 DNS\_RPC\_ZONE\_INFO\_LONGHORN

All fields have same definition as specified in section DNS\_RPC\_ZONE\_INFO\_DOTNET (section [2.2.5.2.4.2](#)), with the following exceptions:

```
typedef struct _DnsRpcZoneInfoLonghorn {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [string] char* pszZoneName;
    DWORD dwZoneType;
    DWORD fReverse;
    DWORD fAllowUpdate;
    DWORD fPaused;
    DWORD fShutdown;
    DWORD fAutoCreated;
    DWORD fUseDatabase;
    [string] char* pszDataFile;
    PDNS_ADDR_ARRAY aipMasters;
    DWORD fSecureSecondaries;
    DWORD fNotifyLevel;
    PDNS_ADDR_ARRAY aipSecondaries;
    PDNS_ADDR_ARRAY aipNotify;
    DWORD fUseWins;
    DWORD fUseNstat;
    DWORD fAging;
    DWORD dwNoRefreshInterval;
    DWORD dwRefreshInterval;
    DWORD dwAvailForScavengeTime;
    PDNS_ADDR_ARRAY aipScavengeServers;
    DWORD dwForwarderTimeout;
    DWORD fForwarderSlave;
    PDNS_ADDR_ARRAY aipLocalMasters;
    DWORD dwDpFlags;
    [string] char* pszDpFqdn;
    [string] wchar_t* pwszZoneDn;
    DWORD dwLastSuccessfulSoaCheck;
    DWORD dwLastSuccessfulXfr;
    DWORD fQueuedForBackgroundLoad;
    DWORD fBackgroundLoadInProgress;
    BOOL fReadOnlyZone;
    DWORD dwLastXfrAttempt;
    DWORD dwLastXfrResult;
} DNS_RPC_ZONE_INFO_LONGHORN,
*PDNS_RPC_ZONE_INFO_LONGHORN,
DNS_RPC_ZONE_INFO,
*PDNS_RPC_ZONE_INFO;
```



**dwRpcStructureVersion:** The DNS server management structure version number. It SHOULD be set to 0x00000002. <40>

**aipMasters:** A pointer to a structure of type DNS\_ADDR\_ARRAY (section 2.2.3.2.3) that specifies a list of IP addresses of the remote DNS servers that can be sources of information for this zone on which to perform zone transfers by a secondary. This value is applicable for secondary, stub and forwarder zones only and MUST be NULL for all other zone types.

**aipSecondaries:** A pointer to a structure of type DNS\_ADDR\_ARRAY (section 2.2.3.2.3) that specifies a list of IP addresses of the remote DNS servers that are secondary DNS servers for this zone, or NULL if there are no secondary DNS servers. If **fSecureSecondaries** is set to ZONE\_SECSECURE\_LIST\_ONLY, then only zone transfers from IP addresses in this list will be honored.

**aipNotify:** A pointer to a structure of type DNS\_ADDR\_ARRAY (section 2.2.3.2.3) that specifies a list of IP addresses of the remote DNS servers that are secondaries for this zone, for which this DNS server is acting as master and the DNS server will send zone notifications to these secondary servers, as directed by the value of **fNotifyLevel** above.

**aipScavengeServers:** A pointer to a structure of type DNS\_ADDR\_ARRAY (section 2.2.3.2.3) that specifies a list of IP addresses of the DNS servers that will perform scavenging for this zone. This value is applicable for zones of type DNS\_ZONE\_TYPE\_PRIMARY (section 2.2.5.1.1) only. If this value is NULL then there are no restrictions on which DNS server may perform scavenging for this zone.

**aipLocalMasters:** A pointer to a structure of type DNS\_ADDR\_ARRAY (section 2.2.3.2.3) that specifies a list of IP addresses of primary DNS servers for this zone. If this value is not NULL then it overrides the masters servers list configured in the directory server.

**fQueuedForBackgroundLoad:** This MUST be set to 0x00000001 if this zone is has not yet been loaded from persistent storage.

**fBackgroundLoadInProgress:** This MUST be set to 0x00000001 if this zone is currently being loaded from persistent storage, or 0x00000000 if it has been loaded.

**fReadOnlyZone:** This MUST be set to 0x00000001 if this zone is backed by a read-only store that will not accept write operations, or 0x00000000 if not.

**dwLastXfrAttempt:** The time-stamp at which last zone transfer was attempted by a DNS server. This field is applicable only for zones which are secondary or not-authoritative. For all other zones senders MUST set this value to zero and receivers MUST ignore it.

**dwLastXfrResult:** The result of the last zone transfer attempted by server. This field is applicable only for zones which are secondary or not-authoritative, and in this case it MUST be either a Win32 error code, or 0xFFFFFFFF to indicate that a zone transfer is currently in progress. For all other zones senders MUST set this value to zero and receivers MUST ignore it.

### 2.2.5.2.5 DNS\_RPC\_ZONE\_SECONDARIES

The DNS\_RPC\_ZONE\_SECONDARIES structure contains the information about the secondary DNS servers for a zone. There are different versions of the DNS\_RPC\_ZONE\_SECONDARIES structure. The DNS server MUST use the structure corresponding to the value of the *dwClientVersion* in DNS Server Management Protocol method calls (section 3.1.4) as in the following table, or if the method call does not specify the value of *dwClientVersion*, the DNS\_RPC\_ZONE\_SECONDARIES\_W2K version of the structure MUST be used.

Value	Structure
0x00000000	DNS_RPC_ZONE_SECONDARIES_W2K (section <a href="#">2.2.5.2.5.1</a> )
0x00060000	DNS_RPC_ZONE_SECONDARIES_DOTNET (section <a href="#">2.2.5.2.5.2</a> )
0x00070000	DNS_RPC_ZONE_SECONDARIES_LONGHORN (section <a href="#">2.2.5.2.5.3</a> )

### 2.2.5.2.5.1 DNS\_RPC\_ZONE\_SECONDARIES\_W2K

```
typedef struct _DnssrvRpcZoneSecondariesW2K {
    DWORD fSecureSecondaries;
    DWORD fNotifyLevel;
    PIP4_ARRAY aipSecondaries;
    PIP4_ARRAY aipNotify;
} DNS_RPC_ZONE_SECONDARIES_W2K,
*PDNS_RPC_ZONE_SECONDARIES_W2K;
```

**fSecureSecondaries:** The secondary security settings configured for this zone. The DNS server MUST respond to zone transfer requests from a secondary server according to the behavior corresponding to the value of the flag, as described in DNS\_ZONE\_SECONDARY\_SECURITY (section [2.2.5.1.2](#)). This value MUST be set to one of the allowed values as specified in DNS\_ZONE\_SECONDARY\_SECURITY (section [2.2.5.1.2](#)).

**fNotifyLevel:** The settings for sending zone notifications for this zone. The DNS server MUST send notify messages to secondary servers as specified by the entry corresponding to the value of this flag, as shown in the table in section [2.2.5.1.3](#). This flag's value MUST be set to one of the allowed values as specified in DNS\_ZONE\_NOTIFY\_LEVEL (section [2.2.5.1.3](#)).

**aipSecondaries:** The list of IPv4 addresses of remote DNS servers that are permitted to perform zone transfers for this zone. The DNS server will honor zone transfer requests from these secondary servers, as specified by fSecureSecondaries above.

**aipNotify:** The list of IPv4 addresses of the remote DNS servers that will be sent notification messages when records in this zone change, as directed by fNotifyLevel above.

### 2.2.5.2.5.2 DNS\_RPC\_ZONE\_SECONDARIES\_DOTNET

All fields have same definition as specified in section DNS\_RPC\_ZONE\_SECONDARIES\_W2K (section [2.2.5.2.5.1](#)), with the following exceptions:

```
typedef struct _DnssrvRpcZoneSecondariesDotNet {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    DWORD fSecureSecondaries;
    DWORD fNotifyLevel;
    PIP4_ARRAY aipSecondaries;
    PIP4_ARRAY aipNotify;
} DNS_RPC_ZONE_SECONDARIES_DOTNET,
*PDNS_RPC_ZONE_SECONDARIES_DOTNET;
```

**dwRpcStructureVersion:** The DNS server management structure version number. It MUST be set to 0x00000001

**dwReserved0:** Reserved for future use. Sender MUST set to zero and receiver MUST ignore this value.

### 2.2.5.2.5.3 DNS\_RPC\_ZONE\_SECONDARIES\_LONGHORN

All fields have same definition as specified in section [DNS\\_RPC\\_ZONE\\_SECONDARIES\\_DOTNET](#) (section [2.2.5.2.5.2](#)), with the following exceptions:

```
typedef struct _DnsSrvRpcZoneSecondariesLonghorn {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    DWORD fSecureSecondaries;
    DWORD fNotifyLevel;
    PDNS_ADDR_ARRAY aipSecondaries;
    PDNS_ADDR_ARRAY aipNotify;
} DNS_RPC_ZONE_SECONDARIES_LONGHORN,
*PDNS_RPC_ZONE_SECONDARIES_LONGHORN,
DNS_RPC_ZONE_SECONDARIES,
*PDNS_RPC_ZONE_SECONDARIES;
```

**dwRpcStructureVersion:** The DNS server management structure version number. It MUST be set to 0x00000002.

**aipSecondaries:** A pointer to a structure of type [DNS\\_ADDR\\_ARRAY](#) (section [2.2.3.2.3](#)) that specifies a list of IP addresses of remote DNS servers that are permitted to perform zone transfers for this zone. The DNS server will honor zone transfer requests from these secondary servers, as directed by the value of `fSecureSecondaries` above.

**aipNotify:** A pointer to a structure of type [DNS\\_ADDR\\_ARRAY](#) (section [2.2.3.2.3](#)) that specifies a list of IP addresses of the remote DNS servers that be sent notification messages when records in this zone change, as directed by `fNotifyLevel` above.

### 2.2.5.2.6 DNS\_RPC\_ZONE\_DATABASE

The `DNS_RPC_ZONE_DATABASE` structure contains information about the persistent data store for a zone on the DNS server. There are different versions of the `DNS_RPC_ZONE_DATABASE` structure. The DNS server MUST use the structure corresponding to the value of `dwClientVersion` in DNS Server Management Protocol method calls (section [3.1.4](#)) as shown in the following table, or if the method call does not specify the value of `dwClientVersion`, the `DNS_RPC_ZONE_DATABASE_W2K` (section [2.2.5.2.6.1](#)) version of the structure MUST be used.

Value	Structure
0x00000000	<code>DNS_RPC_ZONE_DATABASE_W2K</code> (section <a href="#">2.2.5.2.6.1</a> )
0x00060000	<code>DNS_RPC_ZONE_DATABASE_DOTNET</code> (section <a href="#">2.2.5.2.6.2</a> )
0x00070000	<code>DNS_RPC_ZONE_DATABASE_DOTNET</code> (section <a href="#">2.2.5.2.6.2</a> )

#### 2.2.5.2.6.1 DNS\_RPC\_ZONE\_DATABASE\_W2K

```
typedef struct _DnsSrvRpcZoneDatabaseW2K {
    DWORD fDsIntegrated;
    [string] char* pszFileName;
```

```

} DNS_RPC_ZONE_DATABASE_W2K,
*PDNS_RPC_ZONE_DATABASE_W2K;

```

**fDsIntegrated:** This MUST be set TRUE (0x00000001) if the zone is stored in a directory server, or FALSE (0x00000000) if not.

```

0x00000001
0x00000000

```

**pszFileName:** A pointer to a null-terminated UTF-8 string that specifies the name of the file in which this zone is stored, or NULL if this zone is to be stored in a directory server or in a file with the default file name for the zone.

### 2.2.5.2.6.2 DNS\_RPC\_ZONE\_DATABASE\_DOTNET

All fields have same definition as specified in section DNS\_RPC\_ZONE\_DATABASE\_W2K (section [2.2.5.2.6.1](#)), with the following exceptions:

```

typedef struct _DnssrvRpcZoneDatabaseDotNet {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    DWORD fDsIntegrated;
    [string] char* pszFileName;
} DNS_RPC_ZONE_DATABASE_DOTNET,
*PDNS_RPC_ZONE_DATABASE_DOTNET,
DNS_RPC_ZONE_DATABASE,
*PDNS_RPC_ZONE_DATABASE;

```

**dwRpcStructureVersion:** The DNS management structure version number. This MUST be set to 0x00000001.

**dwReserved0:** Reserved for future use. The sender MUST set this value to 0x00000000 and it MUST be ignored by the receiver.

### 2.2.5.2.7 DNS\_RPC\_ZONE\_CREATE\_INFO

The DNS\_RPC\_ZONE\_CREATE\_INFO structure contains information required to create a zone or reset a zone's information on the DNS server. There are different versions of the DNS\_RPC\_ZONE\_CREATE\_INFO structure. The DNS server MUST use the structure corresponding to the value of *dwClientVersion* in DNS Server Management Protocol method calls (section [3.1.4](#)) as shown in the following table, or if the method call does not specify the value of *dwClientVersion*, the DNS\_RPC\_ZONE\_CREATE\_INFO\_W2K version of the structure MUST be used.

Value	Structure
0x00000000	DNS_RPC_ZONE_CREATE_INFO_W2K (section <a href="#">2.2.5.2.7.1</a> )
0x00060000	DNS_RPC_ZONE_CREATE_INFO_DOTNET (section <a href="#">2.2.5.2.7.2</a> )
0x00070000	DNS_RPC_ZONE_CREATE_INFO_LONGHORN (section <a href="#">2.2.5.2.7.3</a> )

## 2.2.5.2.7.1 DNS\_RPC\_ZONE\_CREATE\_INFO\_W2K

```
typedef struct _DnsRpcZoneCreateInfo {
    [string] char* pszZoneName;
    DWORD dwZoneType;
    DWORD fAllowUpdate;
    DWORD fAging;
    DWORD dwFlags;
    [string] char* pszDataFile;
    DWORD fDsIntegrated;
    DWORD fLoadExisting;
    [string] char* pszAdmin;
    PIP4_ARRAY aipMasters;
    PIP4_ARRAY aipSecondaries;
    DWORD fSecureSecondaries;
    DWORD fNotifyLevel;
    [string] char* pvReserved1;
    [string] char* pvReserved2;
    [string] char* pvReserved3;
    [string] char* pvReserved4;
    [string] char* pvReserved5;
    [string] char* pvReserved6;
    [string] char* pvReserved7;
    [string] char* pvReserved8;
    DWORD dwReserved1;
    DWORD dwReserved2;
    DWORD dwReserved3;
    DWORD dwReserved4;
    DWORD dwReserved5;
    DWORD dwReserved6;
    DWORD dwReserved7;
    DWORD dwReserved8;
} DNS_RPC_ZONE_CREATE_INFO_W2K,
*PDNS_RPC_ZONE_CREATE_INFO_W2K;
```

**pszZoneName:** As specified in section [2.2.5.2.4.1](#).

**dwZoneType:** The zone type. This MUST be set to one of the allowed values specified in DNS\_ZONE\_TYPE (section [2.2.5.1.1](#)), and it MUST NOT be either DNS\_ZONE\_TYPE\_CACHE or DNS\_ZONE\_TYPE\_SECONDARY\_CACHE.

**fAllowUpdate:** As specified in section [2.2.5.2.4.1](#).

**fAging:** As specified in section [2.2.5.2.4.1](#).

**dwFlags:** The zone creation behavior that the DNS server is to follow while creating the zone. This field is only used when the operation is **ZoneCreate**. The DNS server MUST ignore the value of this field when the operation is **ZoneTypeReset**. This field MUST be set to any combination of the following values:

Value	Meaning
DNS_ZONE_LOAD_OVERWRITE_MEMORY 0x00000010	If <b>dwZoneType</b> is not set to the value DNS_ZONE_TYPE_PRIMARY (section <a href="#">2.2.5.1.1</a> ), then this flag MUST be ignored. Otherwise, the DNS server MUST attempt to find and load the

Value	Meaning
	zone database from persistent storage instead of creating a new empty zone database. If the value of <code>fDsIntegrated</code> is <code>0x00000001</code> then the DNS server MUST search for a pre-existing zone database in the directory server, otherwise the DNS server MUST search for a pre-existing zone database in a file. If a pre-existing zone database is not found then it SHOULD continue with zone creation, however if a pre-existing zone database was found but could not be loaded then the DNS server MUST fail the operation and return an error.
<code>DNS_ZONE_CREATE_FOR_DCPROMO</code> <code>0x00001000</code>	If this flag is set, the DNS server MUST create the zone such that it is directory server-integrated and stored in the DNS domain partition.
<code>DNS_ZONE_CREATE_FOR_DCPROMO_FOREST</code> <code>0x00004000</code>	If this flag is set, the DNS server MUST create the zone such that it is directory server-integrated and stored in the DNS forest partition.

**pszDataFile:** As specified in section [2.2.5.2.4.1](#).

**fDsIntegrated:** A value of `0x00000001` indicates that the zone is to be created to use the directory server for persistent storage, and `0x00000000` indicates it is not. If this is set to `0x00000001` the caller MUST specify the application directory partition information in `pszDpFqdn` (section [2.2.5.2.7.2](#)); in this case the DNS server MUST ignore the value of `pszDataFile`.

**fLoadExisting:** If the operation is **ZoneCreate** this field is interpreted as a **Boolean** value. If set to TRUE this has the same effect as specifying `DNS_ZONE_LOAD_OVERWRITE_MEMORY` in the **dwFlags** field above. If the operation is **ZoneTypeReset** this field may be set to one of the following values but the DNS server MUST ignore the value of this field if **fDsIntegrated** is not TRUE or **dwZoneType** is not `DNS_ZONE_TYPE_PRIMARY` (section [2.2.5.1.1](#)).

Value	Meaning
<code>DNS_ZONE_LOAD_OVERWRITE_MEMORY</code> <code>0x00000010</code>	The server MUST attempt to find and load the zone database from persistent storage instead of retaining the existing in-memory zone database by searching for a pre-existing zone database in the directory server. If a pre-existing zone database is not found, then the server MUST fail the operation and return an error.
<code>DNS_ZONE_LOAD_OVERWRITE_DS</code> <code>0x00000020</code>	If this flag is set and if the zone already exists in the database, then the server MUST overwrite the existing zone database with current in-memory zone database.

**pszAdmin:** A pointer to a null-terminated UTF-8 string containing the administrator's email name (in the format specified in [RFC1035](#) section 8) or NULL to cause the DNS server to use a default value "hostmaster", followed by the name of the zone. This value MUST be used to populate the zone administrator email field in the SOA record in the new zone.

**aipMasters:** As specified in section [2.2.5.2.4.1](#).

**aipSecondaries:** As specified in section [2.2.5.2.4.1](#).

**fSecureSecondaries:** As specified in section [2.2.5.2.4.1](#).

**fNotifyLevel:** As specified in section [2.2.5.2.4.1](#).

**pvReserved1:** MUST be set to zero when sent and MUST be ignored on receipt.

**pvReserved2:** MUST be set to zero when sent and MUST be ignored on receipt.

**pvReserved3:** MUST be set to zero when sent and MUST be ignored on receipt.

**pvReserved4:** MUST be set to zero when sent and MUST be ignored on receipt.

**pvReserved5:** MUST be set to zero when sent and MUST be ignored on receipt.

**pvReserved6:** MUST be set to zero when sent and MUST be ignored on receipt.

**pvReserved7:** MUST be set to zero when sent and MUST be ignored on receipt.

**pvReserved8:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwReserved1:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwReserved2:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwReserved3:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwReserved4:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwReserved5:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwReserved6:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwReserved7:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwReserved8:** MUST be set to zero when sent and MUST be ignored on receipt.

## 2.2.5.2.7.2 DNS\_RPC\_ZONE\_CREATE\_INFO\_DOTNET

All fields have same definition as specified in section DNS\_RPC\_ZONE\_CREATE\_INFO\_W2K (section [2.2.5.2.7.1](#)), with the following exceptions:

```
typedef struct _DnsRpcZoneCreateInfoDotNet {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [string] char* pszZoneName;
    DWORD dwZoneType;
    DWORD fAllowUpdate;
    DWORD fAging;
    DWORD dwFlags;
    [string] char* pszDataFile;
    DWORD fDsIntegrated;
    DWORD fLoadExisting;
    [string] char* pszAdmin;
    PIP4_ARRAY aipMasters;
    PIP4_ARRAY aipSecondaries;
    DWORD fSecureSecondaries;
    DWORD fNotifyLevel;
}
```

```

    DWORD dwTimeout;
    DWORD fRecurseAfterForwarding;
    DWORD dwDpFlags;
    [string] char* pszDpFqdn;
    DWORD dwReserved[32];
} DNS_RPC_ZONE_CREATE_INFO_DOTNET,
*PDNS_RPC_ZONE_CREATE_INFO_DOTNET;

```

**dwRpcStructureVersion:** As specified in section [2.2.5.2.4.2](#).

**dwReserved0:** As specified in section [2.2.5.2.4.2](#).

**dwTimeout:** Equivalent to dwForwarderTimeout specified in section [2.2.5.2.4.2](#).

**fRecurseAfterForwarding:** Equivalent to fForwarderSlave specified in section [2.2.5.2.4.2](#).

**dwDpFlags:** As specified in section [2.2.5.2.4.2](#). However, only the following values may be used and of these values more than one MUST NOT be specified: "DNS\_DP\_LEGACY", "DNS\_DP\_DOMAIN\_DEFAULT", and "DNS\_DP\_FOREST\_DEFAULT". This field SHOULD be set to zero if the zone is not to be created to use the directory server for persistent storage. The DNS server will return an error if any value not listed above is specified or if more than one of the allowable values are specified.

**pszDpFqdn:** As specified in section [2.2.5.2.4.2](#).

**dwReserved:** MUST be set to zero when sent and MUST be ignored on receipt.

If the DNS RPC client sends an older version of DNS\_RPC\_ZONE\_CREATE\_INFO structure such as DNS\_RPC\_ZONE\_CREATE\_INFO\_W2K (section [2.2.5.2.7.1](#)), then the DNS RPC server MUST construct a current version of DNS\_RPC\_ZONE\_CREATE\_INFO structure such as DNS\_RPC\_ZONE\_CREATE\_INFO\_DOTNET, using steps as specified below:

1. Copy the same value for fields that are common to input and current version of DNS\_RPC\_ZONE\_CREATE\_INFO structures.
2. dwRpcStructureVersion field MUST be set to 1.
3. All other fields that are defined only in DNS\_RPC\_ZONE\_CREATE\_INFO\_DOTNET and are not defined in DNS\_RPC\_ZONE\_CREATE\_INFO\_W2K (section [2.2.5.2.7.1](#)), MUST be set to 0.

### 2.2.5.2.7.3 DNS\_RPC\_ZONE\_CREATE\_INFO\_LONGHORN

All fields have same definition as specified in section DNS\_RPC\_ZONE\_CREATE\_INFO\_DOTNET (section [2.2.5.2.7.2](#)), with the following exceptions:

```

typedef struct _DnsRpcZoneCreateInfoLonghorn {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [string] char* pszZoneName;
    DWORD dwZoneType;
    DWORD fAllowUpdate;
    DWORD fAging;
    DWORD dwFlags;
    [string] char* pszDataFile;
    DWORD fDsIntegrated;
}

```



```

    DWORD fLoadExisting;
    [string] char* pszAdmin;
    PDNS_ADDR_ARRAY aipMasters;
    PDNS_ADDR_ARRAY aipSecondaries;
    DWORD fSecureSecondaries;
    DWORD fNotifyLevel;
    DWORD dwTimeout;
    DWORD fRecurseAfterForwarding;
    DWORD dwDpFlags;
    [string] char* pszDpFqdn;
    DWORD dwReserved[32];
} DNS_RPC_ZONE_CREATE_INFO_LONGHORN,
*PDNS_RPC_ZONE_CREATE_INFO_LONGHORN,
DNS_RPC_ZONE_CREATE_INFO,
*PDNS_RPC_ZONE_CREATE_INFO;

```

**dwRpcStructureVersion:** As specified in section [2.2.5.2.4.3](#).

**aipMasters:** As specified in section [2.2.5.2.4.3](#).

**aipSecondaries:** As specified in section [2.2.5.2.4.3](#).

If the DNS RPC client sends an older version of DNS\_RPC\_ZONE\_CREATE\_INFO structure such as DNS\_RPC\_ZONE\_CREATE\_INFO\_W2K (section [2.2.5.2.7.1](#)) or DNS\_RPC\_ZONE\_CREATE\_INFO\_DOTNET (section [2.2.5.2.7.2](#)), then the DNS RPC server MUST construct a current version of DNS\_RPC\_ZONE\_CREATE\_INFO structure such as DNS\_RPC\_ZONE\_CREATE\_INFO\_LONGHORN, using steps as specified below:

1. Copy the same value for fields that are common to input and current version of DNS\_RPC\_ZONE\_CREATE\_INFO structures.
2. dwRpcStructureVersion field MUST be set to 2.
3. The values for aipMasters and aipSecondaries fields MUST be obtained from input structure as IP4\_ARRAY type and MUST be converted to DNS\_ADDR\_ARRAY type, and then assigned to aipMasters and aipSecondaries fields in the DNS\_RPC\_ZONE\_CREATE\_INFO\_LONGHORN structure.

**Note** DNS\_RPC\_ZONE\_CREATE\_INFO\_W2K (section [2.2.5.2.7.1](#)) and DNS\_RPC\_ZONE\_CREATE\_INFO\_DOTNET (section [2.2.5.2.7.2](#)), do not support IPv6 address list of **aipMasters** and **aipSecondaries**.

4. All other fields that are defined only in DNS\_RPC\_ZONE\_CREATE\_INFO\_LONGHORN and are not defined in DNS\_RPC\_ZONE\_CREATE\_INFO\_W2K (section [2.2.5.2.7.1](#)) or DNS\_RPC\_ZONE\_CREATE\_INFO\_DOTNET (section [2.2.5.2.7.2](#)) structure MUST be set to 0.

### 2.2.5.2.8 DNS\_RPC\_ZONE\_EXPORT\_INFO

The DNS\_RPC\_ZONE\_EXPORT\_INFO structure contains the information file to which a zone is exported on the DNS server. [<41>](#)

```

typedef struct _DnssrvRpcZoneExport {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [string] char* pszZoneExportFile;
}

```

```
} DNS_RPC_ZONE_EXPORT_INFO,  
 *PDNS_RPC_ZONE_EXPORT_INFO;
```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt

**pszZoneExportFile:** A pointer to a null-terminated UTF-8 string that specifies the name of the file to which a zone should be exported by the DNS server.

### 2.2.5.2.9 DNS\_RPC\_ENUM\_ZONES\_FILTER

The DNS\_RPC\_ENUM\_ZONES\_FILTER structure specifies zone filtering criteria.

```
typedef struct _DnsRpcEnumZonesFilter {  
    DWORD dwRpcStructureVersion;  
    DWORD dwReserved0;  
    DWORD dwFilter;  
    [string] char* pszPartitionFqdn;  
    [string] char* pszQueryString;  
    [string] char* pszReserved[6];  
} DNS_RPC_ENUM_ZONES_FILTER,  
 *PDNS_RPC_ENUM_ZONES_FILTER;
```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwFilter:** A filter value that specifies the zone types that are to be selected as part of the output zone list. This value MUST be set to any combination of the ZONE\_REQUEST\_FILTERS (section [2.2.5.1.4](#)).

**pszPartitionFqdn:** A pointer to a null-terminated UTF-8 string that specifies the distinguished name for an application directory partition location from which the server is to enumerate zones; if this is NULL then zone enumeration is not restricted based on the application directory partition.

**pszQueryString:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

**pszReserved:** Reserved for future use. Server MUST set to zero and receiver MUST ignore this value.

### 2.2.5.2.10 DNS\_RPC\_FORWARDERS

The DNS\_RPC\_FORWARDERS structure contains information about forwarders configured on the DNS server. There are different versions of the DNS\_RPC\_FORWARDERS structure. The DNS server MUST use the structures corresponding to the value of *dwClientVersion* in DNS Server Management Protocol method calls (section [3.1.4](#)) in the following table, or if the method call does not specify the value of *dwClientVersion*, the DNS\_RPC\_FORWARDERS\_W2K version of the structure MUST be used.

Value	Structure
0x00000000	DNS_RPC_FORWARDERS_W2K (section <a href="#">2.2.5.2.10.1</a> )
0x00060000	DNS_RPC_FORWARDERS_DOTNET (section <a href="#">2.2.5.2.10.2</a> )
0x00070000	DNS_RPC_FORWARDERS_LONGHORN (section <a href="#">2.2.5.2.10.3</a> )

### 2.2.5.2.10.1 DNS\_RPC\_FORWARDERS\_W2K

```
typedef struct _DnssrvRpcForwardersW2K {
    DWORD fRecurseAfterForwarding;
    DWORD dwForwardTimeout;
    PIP4_ARRAY aipForwarders;
} DNS_RPC_FORWARDERS_W2K,
*PDNS_RPC_FORWARDERS_W2K;
```

**fRecurseAfterForwarding:** A value of 0x00000001 indicates that the DNS server is configured to use normal recursion for name resolution if forwarders are not configured or are unreachable; a value of 0x00000000 indicates it is not.

**dwForwardTimeout:** The time-interval, in seconds, for which the DNS server waits for a response from each server in the forwarders list. No restrictions are applied to the range for the **dwForwardTimeout** member when modifying its value through this structure. If **dwForwardTimeout** is set to zero, then the server SHOULD reset the forward timeout to the default value, 3 minutes (180 seconds).<42>

**aipForwarders:** The list of IP addresses that will be used as forwarders by the DNS server.

### 2.2.5.2.10.2 DNS\_RPC\_FORWARDERS\_DOTNET

All fields have same definition as specified in section DNS\_RPC\_FORWARDERS\_W2K (section [2.2.5.2.10.1](#)), with the following exceptions:

```
typedef struct _DnssrvRpcForwardersDotNet {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    DWORD fRecurseAfterForwarding;
    DWORD dwForwardTimeout;
    PIP4_ARRAY aipForwarders;
} DNS_RPC_FORWARDERS_DOTNET,
*PDNS_RPC_FORWARDERS_DOTNET;
```

**dwRpcStructureVersion:** The structure version number. It MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

### 2.2.5.2.10.3 DNS\_RPC\_FORWARDERS\_LONGHORN

All fields have same definition as specified in section DNS\_RPC\_FORWARDERS\_DOTNET (section [2.2.5.2.10.2](#)), with the following exceptions:

```

typedef struct _DnssrvRpcForwardersLonghorn {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    DWORD fRecurseAfterForwarding;
    DWORD dwForwardTimeout;
    PDNS_ADDR_ARRAY aipForwarders;
} DNS_RPC_FORWARDERS_LONGHORN,
*PDNS_RPC_FORWARDERS_LONGHORN,
DNS_RPC_FORWARDERS,
*PDNS_RPC_FORWARDERS;

```

**dwRpcStructureVersion:** The structure version number. It MUST be set to 0x00000002.

**aipForwarders:** A pointer to a structure of type DNS\_ADDR\_ARRAY (section [2.2.3.2.3](#)) specifies a list of IP addresses that will be used as forwarders by the DNS server.

## 2.2.6 Zone Update Messages

### 2.2.6.1 Enumerations and Constants

#### 2.2.6.1.1 DNS\_ZONE\_UPDATE

A DNS\_ZONE\_UPDATE value is a 2-bit value that indicates the type of dynamic update that is allowed for a zone.

Constant/value	Description
ZONE_UPDATE_OFF 0x0	No updates are allowed for the zone.
ZONE_UPDATE_UNSECURE 0x1	All updates (secure and unsecure) are allowed for the zone.
ZONE_UPDATE_SECURE 0x2	The zone only allows secure updates, that is, the DNS packet MUST have a <b>TSIG</b> <a href="#">[RFC2845]</a> present in the additional section.

#### 2.2.6.1.2 KeySignScope

**KeySignScope**[<43>](#) defines the scope of the key in a signed zone. It is used to indicate different signing and publishing scopes of the key.

Constant/value	Description
SIGN_SCOPE_DEFAULT 0x00000000	The key is used for its default purpose: If the <b>signing key descriptor's fIsKSK</b> field is set to 0x00000001, the key is used to sign only DNSKEY records in the zone. If the signing key descriptor's <b>fIsKSK</b> field is set to 0x00000000, the key is used to sign all records in the zone.
SIGN_SCOPE_DNSKEY_ONLY 0x00000001	The key is used to sign only DNSKEY records in the zone.
SIGN_SCOPE_ALL_RECORDS	The key is used to sign all records in the zone.

Constant/value	Description
0x00000002	
SIGN_SCOPE_ADD_ONLY 0x00000003	The key is published as a DNSKEY in the zone, but it is not used to sign any records.
SIGN_SCOPE_DO_NOT_PUBLISH 0x00000004	The key is not published to the zone and is not used to sign any records.
SIGN_SCOPE_REVOKED 0x00000005	The key is published as a DNSKEY in the zone with its Revoked bit ( <a href="#">RFC5011</a> section 2.1) set. It is used to sign DNSKEY records.

### 2.2.6.1.3 ImportOpResult

ImportOpResult<[44](#)> defines the operations to be executed when the DNS\_RPC\_ZONE\_DNSSEC\_SETTINGS is imported on a zone.

Constant/value	Description
IMPORT_STATUS_NOOP 0x0	No change was detected in the imported signing settings or the signing settings of the zone.
IMPORT_STATUS_SIGNING_READY 0x1	The zone is marked for re-signing.
IMPORT_STATUS_UNSIGNING_READY 0x2	The zone is marked for unsigning.
IMPORT_STATUS_CHANGED 0x3	The change was detected in signing settings imported and were incorporated, but no re-signing or unsigning is required.

## 2.2.6.2 Structures

### 2.2.6.2.1 DNS\_RPC\_SKD

The **DNS\_RPC\_SKD** structure specifies a signing key descriptor.

```
typedef struct _DnssrvRpcSigningKeyDescriptor {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    GUID Guid;
    [string] wchar_t* pwszKeyStorageProvider;
    BOOL fStoreKeysInDirectory;
    BOOL fIsKSK;
    BYTE bSigningAlgorithm;
    DWORD dwKeyLength;
    DWORD dwInitialRolloverOffset;
    DWORD dwDNSKEYSignatureValidityPeriod;
    DWORD dwDSSignatureValidityPeriod;
    DWORD dwStandardSignatureValidityPeriod;
    DWORD dwRolloverType;
    DWORD dwRolloverPeriod;
    DWORD dwNextRolloverAction;
    DWORD dwReserved;
```

```
} DNS_RPC_SKD,  
 *PDNS_RPC_SKD;
```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**Guid:** A unique identifier for this signing key descriptor.

**pwszKeyStorageProvider:** The Key Storage Provider that will be used to generate and store keys for this signing key descriptor.

**fStoreKeysInDirectory:** A value of 0x00000001 indicates that the DNS server should export keys generated for this signing key descriptor and store them on the DNS zone object in Active Directory. A value of 0x00000000 indicates that the DNS server should store keys in self-signed certificates in the local machine certificate store.

**fIsKSK:** A value of 0x00000001 indicates that this signing key descriptor describes a key signing key; a value of 0x00000000 indicates a zone signing key [\[RFC4641\]](#).

**bSigningAlgorithm:** The cryptographic algorithm used to generate signing keys. The DNS server SHOULD support all values given by [\[IANA-DNSSECAN\]](#) and [\[DRAFT-DNSSEC-ECDSA\].<45>](#)

**dwKeyLength:** The length, in bits, of cryptographic signing keys. This value MUST be ignored if the signing algorithm does not have variable key length.

**dwInitialRolloverOffset:** The amount of time, in seconds, to delay the first scheduled key rollover for this signing key descriptor. The value SHOULD be limited to the range 0x00000000 to 0x00278D00 (30 days), inclusive, but it MAY be any value. The default value SHOULD be 0x00000000, and the value zero MUST be allowed and treated literally.

**dwDNSKEYSignatureValidityPeriod:** Amount of time, in seconds, that signatures covering DNSKEY record sets generated for this signing key descriptor's keys should be valid, as defined by [\[RFC4034\]](#) section 3.1.5. The value SHOULD be limited to the range 0x00005460 (6 hours) to 0x00093A80 (7 days), inclusive, but it can be any value. The default value SHOULD be 0x0003F480 (3 days).

**dwDSSignatureValidityPeriod:** Amount of time, in seconds, that signatures covering DS record sets generated for this signing key descriptor's keys should be valid, as defined by [\[RFC4034\]](#) section 3.1.5. The value SHOULD be limited to the range 0x00005460 (6 hours) to 0x00093A80 (7 days), inclusive, but it can be any value. The default value SHOULD be 0x0003F480 (3 days).

**dwStandardSignatureValidityPeriod:** Amount of time, in seconds, that signatures covering record sets not of type DNSKEY or DS generated for this signing key descriptor's keys should be valid, as defined by [\[RFC4034\]](#) section 3.1.5. The value SHOULD be limited to the range 0x00005460 (6 hours) to 0x00278D00 (30 days), inclusive, but it can be any value. The default value SHOULD be 0x000D2F00 (10 days).

**dwRolloverType:** When sent, this value MUST be set to one of the allowed values specified in [ZONE\\_SKD\\_ROLLOVER\\_TYPE \(section 2.2.5.1.5\)](#) as follows: this value MUST be DNS\_ROLLOVER\_TYPE\_PREPUBLISH if **fIsKSK** is 0x00000000 and MUST be DNS\_ROLLOVER\_TYPE\_DOUBLE\_SIGNATURE if **fIsKSK** is 0x00000001. On receipt, this value MUST be ignored.

**dwRolloverPeriod:** Amount of time, in seconds, between scheduled key rollovers; or 0xFFFFFFFF to disable automatic key rollovers. This value SHOULD be limited to the range 0x00093A80 (1 week) to 0x25980600 (20 years), inclusive, in addition to 0xFFFFFFFF, when **fIsKSK** is 0x00000001, and the range 0x00093A80 (1 week) to 0x09660180 (5 years), inclusive, in addition to 0xFFFFFFFF, when **fIsKSK** is 0x00000000. The default SHOULD be 0x02022900 (13 months) when **fIsKSK** is 0x00000001, and 0x0x00278D00 (1 month) when **fIsKSK** is 0x00000000.

**dwNextRolloverAction:** This value describes the next key rollover action that the DNS server will take for this signing key descriptor. This value MUST be set to one of the allowed values specified in [ZONE SKD ROLLOVER ACTION \(section 2.2.5.1.6\)](#).

**dwReserved:** This value MUST be set to 0x00000000 when sent by the client and ignored on receipt by the server.

### 2.2.6.2.2 DNS\_RPC\_SKD\_LIST

The **DNS\_RPC\_SKD\_LIST** structure contains the information about a list of signing key descriptors that are present for a particular zone on the DNS server.

```
typedef struct _DnssrvRpcZoneSigningKeyList {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [range(0,1000)] DWORD dwCount;
    [size_is(dwCount)] PDNS_RPC_SKD SkdArray[];
} DNS_RPC_SKD_LIST,
*PDNS_RPC_SKD_LIST;
```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwCount:** The number of signing key descriptors present in the array of signing key descriptors pointed to by **SkdArray**.

**SkdArray:** An array of structures of type [DNS\\_RPC\\_SKD \(section 2.2.6.2.1\)](#). Each element of the array represents one signing key descriptor.

### 2.2.6.2.3 DNS\_RPC\_SKD\_STATE

The **DNS\_RPC\_SKD\_STATE** structure contains information about the current state of a signing key descriptor.

```
typedef struct _DnsRpcSigningKeyDescriptorState {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    GUID Guid;
    FILETIME ftLastRolloverTime;
    FILETIME ftNextRolloverTime;
    DWORD dwState;
    DWORD dwCurrentRolloverStatus;
    [string] wchar_t* pwszActiveKey;
    [string] wchar_t* pwszStandbyKey;
    [string] wchar_t* pwszNextKey;
    DWORD dwReserved;
```

```

} DNS_RPC_SKD_STATE,
 *PDNS_RPC_SKD_STATE;

```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**Guid:** A unique identifier for this signing key descriptor.

**ftLastRolloverTime:** The time at which the last successful rollover event was performed for this signing key descriptor. This value MUST be 0 if no rollover has been performed on this signing key descriptor.

**ftNextRolloverTime:** The time at which the next rollover for this signing key descriptor is scheduled. This MAY be 0 if no rollover event is scheduled. When the time comes for a key rollover to start, the signing key descriptor is added to the Rollover Queue, and its **dwCurrentRolloverStatus** is changed to `DNS_SKD_STATUS_QUEUED`. If another signing key descriptor in the zone is in the process of rolling, **ftNextRolloverTime** MAY represent a time in the past.

**dwState:** The current state of this signing key descriptor. This MUST be set to one of the following values.

Value	Meaning
DNS_SKD_STATE_ACTIVE 0x00000000	The signing key descriptor is active and in use for <b>online signing</b> of the zone.
DNS_SKD_STATE_RETIRED 0x00000001	The signing key descriptor is no longer in use for online signing.

**dwCurrentRolloverStatus:** The current rollover status of this signing key descriptor. This MUST be set to one of the following values, representing the various stages of key rollover as described in [\[RFC4641\]](#) and [\[RFC5011\]](#):

Value	Meaning
DNS_SKD_STATUS_NOT_ROLLING 0x00000000	The signing key descriptor is not currently in the process of rolling over keys.
DNS_SKD_STATUS_QUEUED 0x00000001	This signing key descriptor is waiting for another rollover to complete before its rollover can begin. After the signing key descriptor reaches the head of the Rollover Queue, the signing key descriptor will move into the <code>DNS_SKD_STATUS_ROLL_STARTED</code> state.
DNS_SKD_STATUS_ROLL_STARTED 0x00000002	This signing key descriptor has begun the process of key rollover. Signing key descriptors representing <b>ZSKs</b> will move from this state to the <code>DNS_SKD_STATUS_ZSK_WAITING_FOR_DNSKEY_TTL</code> state, and signing key descriptors representing KSKs will move from this state to the



Value	Meaning
	DNS_SKD_STATUS_KSK_WAITING_FOR_DS_UP DATE state.
DNS_SKD_STATUS_ZSK_WAITING_FOR_DNSKEY_T TL 0x00000003	This ZSK signing key descriptor is waiting for the previous key to expire in all caching resolvers (it is waiting for the DNSKEY TTL to expire). The signing key descriptor will next move into the DNS_SKD_STATUS_ZSK_WAITING_FOR_MAXZONE_TTL state.
DNS_SKD_STATUS_ZSK_WAITING_FOR_MAXZONE_TTL 0x00000004	This ZSK signing key descriptor is waiting for the signatures using the previous key to expire in all caching resolvers (it is waiting for the largest record TTL to expire). This is the final state for ZSK rollovers. The signing key descriptor will move into the DNS_SKD_STATUS_NOT_ROLLING state when this portion of key rollover is complete.
DNS_SKD_STATUS_KSK_WAITING_FOR_DS_UPDAT E 0x00000005	This KSK signing key descriptor is waiting for a DS record corresponding to the new key to appear in the parent zone. If DS records are found in the parent zone, the server MUST set the zone's "ParentHasSecureDelegation" property to 0x00000001 and continue to wait until the parent's DS record set includes the new key, at which point the signing key descriptor will move into the DNS_SKD_STATUS_KSK_WAITING_FOR_DS_TTL state. If no DS records are found in the parent zone, the server MUST set the zone's "ParentHasSecureDelegation" property to 0x00000000 and transition the signing key descriptor into the DNS_SKD_STATUS_KSK_WAITING_FOR_DNSKEY_TTL state. If there is an error or if the presence of a DS record set in the parent zone cannot be determined, the server MUST continue to attempt to query for this record set for up to 15 minutes if the zone's "ParentHasSecureDelegation" property is 0x00000000 or until the "PokeZoneKeyRollover" command is received if it is 0x00000001.
DNS_SKD_STATUS_KSK_WAITING_FOR_DS_TTL 0x00000006	This KSK signing key descriptor is waiting for the DS record set in the parent zone to expire in all caching resolvers (it is waiting for the parent DS TTL to expire). If the zone's "RFC5011KeyRollovers" property is TRUE, the signing key descriptor will next move into the DNS_SKD_STATUS_WAITING_FOR_5011_REMOVE_HOLD_DOWN state. Otherwise, this is a final state for KSK rollovers, and signing key descriptors will move into the DNS_SKD_STATUS_NOT_ROLLING state when this portion of key rollover is complete.

Value	Meaning
DNS_SKD_STATUS_KSK_WAITING_FOR_DNSKEY_TTL 0x00000007	This KSK signing key descriptor is waiting for the previous key to expire in all caching resolvers (it is waiting for the DNSKEY TTL to expire). This is a final state for KSK rollovers, and signing key descriptors will move into the DNS_SKD_STATUS_NOT_ROLLING state when this portion of key rollover is complete.
DNS_SKD_STATUS_KSK_WAITING_FOR_5011_REMOVE_HOLD_DOWN 0x00000008	This KSK signing key descriptor is waiting for the RFC5011 remove hold-down time before the revoked previous key can be removed. This is a final state for KSK rollovers, and signing key descriptors will move into the DNS_SKD_STATUS_NOT_ROLLING state when this portion of key rollover is complete.
DNS_SKD_STATUS_ROLL_ERROR 0x00000009	This signing key descriptor experienced an unrecoverable error during the key rollover.

**pwszActiveKey:** Signing key pointer string for the signing key descriptor's currently active key. This is the key that is currently being used to sign zone records.

**pwszStandbyKey:** Signing key pointer string for the signing key descriptor's standby key. The key represented by this string has several different roles depending on the associated signing key descriptor's "fIsKSK" flag and its "dwRolloverStatus":

- If the signing key descriptor's "fIsKSK" flag is TRUE, pwszStandbyKey represents the "double signature" key as described in [\[RFC4641\]](#) section 4.2.2, also depicted as the "standby" key in [\[RFC5011\]](#) section 6.
- If the signing key descriptor's "fIsKSK" flag is FALSE, **pwszStandbyKey** will generally be NULL unless the SKD is in the process of key rollover:
  - If the signing key descriptor's "dwRolloverStatus" is DNS\_SKD\_STATUS\_ZSK\_WAITING\_FOR\_DNSKEY\_TTL, **pwszStandbyKey** represents the "pre-publish" key as described in [\[RFC4641\]](#) section 4.2.1.1.
  - If the signing key descriptor's "dwRolloverStatus" is DNS\_SKD\_STATUS\_ZSK\_WAITING\_FOR\_MAXZONE\_TTL, **pwszStandbyKey** represents the previously active key during the "new RRSIGs" phase of Pre-Publish Key Rollover, as described in [\[RFC4641\]](#) section 4.2.1.1.

**pwszNextKey:** Signing key pointer string for the signing key descriptor's next key. The key represented by this string has several different roles depending on the associated signing key descriptor's "fIsKSK" flag:

- If the signing key descriptor's "fIsKSK" flag is TRUE, **pwszNextKey** represents the next key that will be consumed during key rollover. It is not published in the zone and is not used to sign any other records. If the zone's "RFC5011KeyRollovers" property is TRUE, this key becomes the next "standby" key according to [\[RFC5011\]](#). Otherwise, this key is used as the "double signature" key according to [\[RFC4641\]](#) as the signing key descriptor's key rollover process begins.
- If the signing key descriptor's "fIsKSK" flag is FALSE, **pwszNextKey** represents the "pre-publish" key as described in [\[RFC4641\]](#) section 4.2.1.1. When the SKD is in the process of

key rollover, **pwszNextKey** may be populated with a newly generated post-rollover "pre-publish" key.

**dwReserved:** MUST be set to zero when sent and MUST be ignored on receipt.

#### 2.2.6.2.4 DNS\_RPC\_TRUST\_POINT

The **DNS\_RPC\_TRUST\_POINT** structure contains information about a trust point or a node in the "TrustAnchors" zone.

```
typedef struct _DnssrvRpcTrustPoint {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [string] char* pszTrustPointName;
    TRUSTPOINT_STATE eTrustPointState;
    __int64 i64LastActiveRefreshTime;
    __int64 i64NextActiveRefreshTime;
    __int64 i64LastSuccessfulActiveRefreshTime;
    DWORD dwLastActiveRefreshResult;
    DWORD dwReserved;
} DNS_RPC_TRUST_POINT,
*PDNS_RPC_TRUST_POINT;
```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**pszTrustPointName:** The FQDN of the trust point or node in the "TrustAnchors" zone represented by this structure. This MUST be a non-empty, non-null string.

**eTrustPointState:** An enum containing the current trust point state. This MUST be set to one of the following values.

Value	Meaning
TRUSTPOINT_STATE_INITIALIZED 0x0000	This structure represents a node in the "TrustAnchors" zone that does not contain any trust anchors. This node is not a trust point.
TRUSTPOINT_STATE_DSPENDING 0x0001	This trust point contains only DS trust anchors (trust anchors in the TRUSTANCHOR_STATE_DSPENDING or TRUSTANCHOR_STATE_DSINVALID state), rendering it unusable for DNSSEC proofs.
TRUSTPOINT_STATE_ACTIVE 0x0002	This trust point contains at least one trust anchor in the TRUSTANCHOR_STATE_VALID or TRUSTANCHOR_STATE_MISSING state.
TRUSTPOINT_STATE_DELETE_PENDING 0x0003	This trust point contains only trust anchors in the TRUSTANCHOR_STATE_REVOKED state.

**i64LastActiveRefreshTime:** The time of the last **active refresh**, or zero to indicate that no active refresh has occurred. This is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 **Coordinated Universal Time (UTC)**.

**i64NextActiveRefreshTime:** The scheduled time of the next active refresh, or zero to indicate that no active refresh is scheduled. This is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC).

**i64LastSuccessfulActiveRefreshTime:** The time of the last successful active refresh, or zero to indicate that no active refresh has occurred. This is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC). A "successful active refresh" is defined as an active refresh resulting in retrieval of one or more DNSKEY records for the trust point and, if this trust point has trust anchors in the TRUSTANCHOR\_STATE\_VALID state, signifies that one or more of the retrieved DNSKEY records was validated by DNSSEC.

**dwLastActiveRefreshResult:** The result of the last active refresh, either ERROR\_SUCCESS or a nonzero value to indicate that an error has occurred.

**dwReserved:** MUST be set to zero when sent and MUST be ignored on receipt.

### 2.2.6.2.5 DNS\_RPC\_TRUST\_POINT\_LIST

The **DNS\_RPC\_TRUST\_POINT\_LIST** structure contains zero or more [DNS\\_RPC\\_TRUST\\_POINT](#) structures.

```
typedef struct _DnssrvRpcTrustPointList {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [range(0,500000)] DWORD dwTrustPointCount;
    [size_is(dwTrustPointCount)] PDNS_RPC_TRUST_POINT TrustPointArray[];
} DNS_RPC_TRUST_POINT_LIST,
*PDNS_RPC_TRUST_POINT_LIST;
```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwTrustPointCount:** The size of the **TrustPointArray** array.

**TrustPointArray:** An array of pointers to **DNS\_RPC\_TRUST\_POINT** structures.

### 2.2.6.2.6 DNS\_RPC\_TRUST\_ANCHOR

The **DNS\_RPC\_TRUST\_ANCHOR** structure contains information about a trust anchor.

```
typedef struct _DnssrvRpcTrustAnchor {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    WORD wTrustAnchorType;
    WORD wKeyTag;
    WORD wRRLength;
    TRUSTANCHOR_STATE eTrustAnchorState;
    __int64 i64EnteredStateTime;
    __int64 i64NextStateTime;
    DWORD dwReserved;
    [size_is(wRRLength)] BYTE RRData[];
} DNS_RPC_TRUST_ANCHOR,
*PDNS_RPC_TRUST_ANCHOR;
```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**wTrustAnchorType:** The DNS record type corresponding to the trust anchor. This MUST be set to one of the following values.

Value	Meaning
DNS_TYPE_DS 0x002B	A DS record type <a href="#">[RFC4034]</a> .
DNS_TYPE_DNSKEY 0x0030	A DNSKEY record type <a href="#">[RFC4034]</a> .

**wKeyTag:** The DNSSEC key tag for this trust anchor. The key tag for a DS record trust anchor MUST match the value of the record's "Key Tag" field (see [\[RFC4034\]](#)). The key tag for a DNSKEY record trust anchor MUST match the value calculated for the DNSKEY record (see [\[RFC4034\]](#) Appendix B), with the exception that the REVOKE bit of the DNSKEY flags field must be set to zero before the calculation.

**wRRLength:** The length of the **RRData** array.

**eTrustAnchorState:** The current state of the trust anchor. This MUST be one of the following values.

Value	Meaning
TRUSTANCHOR_STATE_DSPENDING 0x0001	This <b>trust anchor</b> can be replaced with a matching DNSKEY trust anchor when the associated trust point has had a successful active refresh. If this state is set, <b>wTrustAnchorType</b> MUST be DNS_TYPE_DS.
TRUSTANCHOR_STATE_DSINVALID 0x0002	The associated <b>trust point</b> has had a successful active refresh, but no DNSKEY record was found that matches this trust anchor. If this state is set, <b>wTrustAnchorType</b> MUST be DNS_TYPE_DS.
TRUSTANCHOR_STATE_ADDPEND 0x0003	This <b>trust anchor</b> will become valid after the expiration of the RFC 5011 add hold-down time (see <a href="#">[RFC5011]</a> ). This corresponds to the "AddPend" state in RFC 5011.
TRUSTANCHOR_STATE_VALID 0x0004	This <b>trust anchor</b> is trusted for DNSSEC proofs because it was either explicitly added by the administrator or became valid after the expiration of the RFC 5011 add hold-down time (see <a href="#">[RFC5011]</a> ). This corresponds to the "Valid" state in RFC 5011.
TRUSTANCHOR_STATE_MISSING 0x0005	This trust anchor was in the TRUSTANCHOR_STATE_VALID state but was missing in the last successful active refresh. It is still trusted for DNSSEC proofs. This corresponds to the "Valid" state in <a href="#">[RFC5011]</a> .
TRUSTANCHOR_STATE_REVOKED	This <b>trust anchor</b> has been marked as revoked by the

Value	Meaning
0x0006	administrator for the <b>trust point</b> 's zone. It can never again be trusted for DNSSEC proofs. This corresponds to the "Revoked" state in <a href="#">[RFC5011]</a> .

**i64EnteredStateTime:** The time at which this trust anchor entered its current state. This is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC).

**i64NextStateTime:** The time at which this trust anchor is scheduled to exit the current state. This is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC). The meaning is dependent on the value of **eTrustAnchorState**.

Value of eTrustAnchorState	Meaning
TRUSTANCHOR_STATE_DSPENDING 0x0001	Reserved. The value of <b>i64NextStateTime</b> MUST be set to zero when sent and MUST be ignored on receipt.
TRUSTANCHOR_STATE_DSINVALID 0x0002	Reserved. The value of <b>i64NextStateTime</b> MUST be set to zero when sent and MUST be ignored on receipt.
TRUSTANCHOR_STATE_ADDPEND 0x0003	This <b>trust anchor</b> is scheduled to enter the TRUSTANCHOR_STATE_VALID state on or after the value of <b>i64NextStateTime</b> . This MUST be equivalent to the value of <b>i64EnteredStateTime</b> added to the value of the add hold-down time (see <a href="#">[RFC5011]</a> ).
TRUSTANCHOR_STATE_VALID 0x0004	Reserved. The value of <b>i64NextStateTime</b> MUST be set to zero when sent and MUST be ignored on receipt.
TRUSTANCHOR_STATE_MISSING 0x0005	Reserved. The value of <b>i64NextStateTime</b> MUST be set to zero when sent and MUST be ignored on receipt.
TRUSTANCHOR_STATE_REVOKED 0x0006	This <b>trust anchor</b> will become eligible for deletion on or after the value of <b>i64NextStateTime</b> . This MUST be equivalent to the value of <b>i64EnteredStateTime</b> added to the value of the remove hold-down time (see <a href="#">[RFC5011]</a> ).

**dwReserved:** MUST be set to zero when sent and MUST be ignored on receipt.

**RRData:** Binary data in the same format as [DNS RPC RECORD DNSKEY \(section 2.2.2.2.4.15\)](#) if **wTrustAnchorType** is DNS\_TYPE\_DNSKEY, or binary data in the same format as [DNS RPC RECORD DS \(section 2.2.2.2.4.12\)](#) if **wTrustAnchorType** is DNS\_TYPE\_DS.

### 2.2.6.2.7 DNS\_RPC\_TRUST\_ANCHOR\_LIST

The **DNS\_RPC\_TRUST\_ANCHOR\_LIST** structure contains zero or more [DNS\\_RPC\\_TRUST\\_ANCHOR](#) structures.

```
typedef struct _DnssrvRpcTrustAnchorList {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [range(0,500000)] DWORD dwTrustAnchorCount;
    [size_is(dwTrustAnchorCount)] PDNS_RPC_TRUST_ANCHOR TrustAnchorArray[];
} DNS_RPC_TRUST_ANCHOR_LIST,
```

\*PDNS\_RPC\_TRUST\_ANCHOR\_LIST;

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwTrustAnchorCount:** The size of the **TrustAnchorArray** array.

**TrustAnchorArray:** An array of pointers to **DNS\_RPC\_TRUST\_ANCHOR** structures.

#### 2.2.6.2.8 DNS\_RPC\_SIGNING\_VALIDATION\_ERROR

The **DNS\_RPC\_SIGNING\_VALIDATION\_ERROR** structure describes an error that occurred during the use of an SKD.

```
typedef struct _DnssrvRpcSigningValidationError {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    GUID guidSKD;
    [string] wchar_t* pwszSigningKeyPointerString;
    DWORD dwExtendedError;
    DWORD dwReserved;
} DNS_RPC_SIGNING_VALIDATION_ERROR,
*PDNS_RPC_SIGNING_VALIDATION_ERROR;
```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**guidSKD:** The unique identifier for the signing key descriptor that caused the validation error.

**pwszSigningKeyPointerString:** The signing key pointer string of the signing key that caused the validation error.

**dwExtendedError:** MUST be set to zero when sent.

**dwReserved:** MUST be set to zero when sent and MUST be ignored on receipt.

#### 2.2.6.2.9 DNS\_RPC\_ZONE\_DNSSEC\_SETTINGS

The **DNS\_RPC\_ZONE\_DNSSEC\_SETTINGS** structure [<46>](#) represents the DNSSEC properties of a zone.

```
typedef struct DnssrvRpcZoneDnssecSettings {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    DWORD fIsSigned;
    DWORD fSignWithNSEC3;
    DWORD fNSEC3OptOut;
    DWORD dwMaintainTrustAnchor;
    DWORD fParentHasSecureDelegation;
    DWORD dwDSRecordAlgorithms;
    DWORD fRFC5011KeyRollovers;
```

```

BYTE bNSEC3HashAlgorithm;
BYTE bNSEC3RandomSaltLength;
WORD wNSEC3IterationCount;
LPWSTR pwszNSEC3UserSalt;
DWORD dwDNSKEYRecordSetTtl;
DWORD dwDSRecordSetTtl;
DWORD dwSignatureInceptionOffset;
DWORD dwSecureDelegationPollingPeriod;
DWORD dwPropagationTime;
DWORD cbNSEC3CurrentSaltLength;
PBYTE pbNSEC3CurrentSalt;
GUID CurrentRollingSKDGuid;
DWORD pdwBufferLength;
PBYTE ppBuffer;
DWORD dwCount;
PDNS_RPC_ZONE_SKD pZoneSkdArray;
} DNS_RPC_ZONE_DNSSEC_SETTINGS,
*PDNS_RPC_ZONE_DNSSEC_SETTINGS;

```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**fIsSigned:** States whether or not the zone is signed.

**fSignWithNSEC3:** States whether the zone has NSEC3 Authenticated Denial of Existence support.

**fNSEC3OptOut:** States whether the zone has unsecure delegations.

**dwMaintainTrustAnchor:** States whether Trust Anchors are maintained for this zone.

**fParentHasSecureDelegation:** Delegation Status (Secure\UnSecure) for this zone from the parent.

**dwDSRecordAlgorithms:** Algorithms used for generating a hash of the DNSKEY record.

**fRFC5011KeyRollovers:** 5011 rollover status (Enabled\Disabled) for this zone.

**bNSEC3HashAlgorithm:** Algorithm used for generating NSEC3 hash (see [RFC5155](#) section 5).

**bNSEC3RandomSaltLength:** Length of Salt used in generating NSEC3 records for this zone (see [RFC5155](#) section 5).

**wNSEC3IterationCount:** Iteration count for generating NSEC3 records for this zone (see [RFC5155](#) section 5).

**pwszNSEC3UserSalt:** User defined salt used for generating NSEC3 records for this zone (see [RFC5155](#) section 5).

**dwDNSKEYRecordSetTtl:** Time-to-live (TTL) for the DNSKEY resource record.

**dwDSRecordSetTtl:** TTL for the DS Resource Record.

**dwSignatureInceptionOffset:** Time in seconds for Inception of Signatures for RRSIGs as defined in [RFC4034](#) section 3.1.5.



**dwSecureDelegationPollingPeriod:** The interval, in seconds, to refresh the set of delegation signer (DS) records in a secure delegation.

**dwPropagationTime:** The time, in seconds, that it takes for zone data changes to propagate to other copies of the zone.

**cbNSEC3CurrentSaltLength:** Length of the Current User salt for building an NSEC3 chain of zone records.

**pbNSEC3CurrentSalt:** Pointer to the pwszNSEC3UserSalt for building an NSEC3 chain of zone records.

**CurrentRollingSKDGuid:** Unique identifier of a rolling SKD of a zone, if any.

**pdwBufferLength:** A pointer to an integer that on success contains the length of the buffer pointed to by **ppBuffer**.

**ppBuffer:** A pointer to a pointer that points to a buffer containing the enumerated records. The buffer is a series of structures beginning with a [DNS\\_RPC\\_NODE structure \(section 2.2.2.2.3\)](#). The records for the node will be represented by a series of [DNS\\_RPC\\_RECORD structures \(section 2.2.2.2.5\)](#). The number of DNS\_RPC\_RECORD structures following a **DNS\_RPC\_NODE** structure is given by the wRecordCount member of **DNS\_RPC\_NODE**.

**dwCount:** The number of signing key descriptors present in the array of signing key descriptors pointed to by **SkdArray**.

**pZoneSkdArray:** A list of SKDs for a zone. This is the array of the **DNS\_RPC\_ZONE\_SKD** structure.

#### 2.2.6.2.10 DNS\_RPC\_ZONE\_SKD

The **DNS\_RPC\_ZONE\_SKD** structure [<47>](#) groups all the properties of a signing key descriptor of a zone.

```
typedef struct DnssrvRpcZoneSKDSettings {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    PDNS_RPC_SKD pSkd;
    PDNS_RPC_SKD_STATE pSkdState;
    PDNS_RPC_SKD_STATE_EX pSkdStateEx;
} DNS_RPC_ZONE_SKD,
*PDNS_RPC_ZONE_SKD;
```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**pSkd:** The static properties of a zone SKD. This is a [DNS\\_RPC\\_SKD structure \(section 2.2.6.2.1\)](#).

**pSkdState:** The dynamic properties of a zone SKD. This is a [DNS\\_RPC\\_SKD\\_STATE structure \(section 2.2.6.2.3\)](#).

**pSkdStateEx:** The extended dynamic properties of a zone SKD. This is a [DNS\\_RPC\\_SKD\\_STATE\\_EX structure \(section 2.2.6.2.11\)](#).

## 2.2.6.2.11 DNS\_RPC\_SKD\_STATE\_EX

The **DNS\_RPC\_SKD\_STATE\_EX** structure [<48>](#) represents the collection of extended dynamic configuration information of a signing key descriptor state.

```
typedef struct _DnsRpcSigningKeyDescriptorStateEx {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    GUID Guid;
    DWORD dwCurrentRollState;
    DWORD fManualTrigger;
    DWORD dwPreRollEventFired;
    FILETIME ftNextKeyGenerationTime;
    DWORD dwRevokedOrSwappedDnskeysLength;
    PBYTE pRevokedOrSwappedDnskeysBuffer;
    DWORD dwFinalDnskeysLength;
    PBYTE pFinalDnskeys;
    KeySignScope eActiveKeyScope;
    KeySignScope eStandbyKeyScope;
    KeySignScope eNextKeyScope;
} DNS_RPC_SKD_STATE_EX,
*PDNS_RPC_SKD_STATE_EX;
```

**dwRpcStructureVersion:** The structure version number; this MUST be set to 0x00000001.

**dwReserved0:** This MUST be set to zero when sent and MUST be ignored on receipt.

**Guid:** A unique identifier for this signing key descriptor.

**dwCurrentRollState:** The current rollover status of this signing key descriptor. Note that these are DNS Server Management Protocol implementations of the protocol specified rollover states of SKDs in DNS\_RPC\_SKD\_STATE.

Value	Mapping in DNS_RPC_SKD_STATE
DNS_SKD_ROLL_STATE_NOT_STARTED 0x00000000	DNS_SKD_STATUS_NOT_ROLLING
DNS_SKD_ROLL_STATE_ZSK_SWAP_ACTIVE_WITH_ROLLOVER 0x00000001	DNS_SKD_STATUS_ZSK_WAITING_FOR_DNSKEY_TTL
DNS_SKD_ROLL_STATE_ZSK_FINISH 0x00000002	DNS_SKD_STATUS_ZSK_WAITING_FOR_MAXZONE_TTL
DNS_SKD_ROLL_STATE_KSK_WAITING_FOR_DS_UPDATE 0x00000003	DNS_SKD_STATUS_KSK_WAITING_FOR_DS_UPDATE
DNS_SKD_ROLL_STATE_KSK_REVOKE 0x00000004	DNS_SKD_STATUS_KSK_WAITING_FOR_DS_TTL The key is revoked once the rollover is completed.
DNS_SKD_ROLL_STATE_KSK_FINISH 0x00000005	DNS_SKD_STATUS_KSK_WAITING_FOR_DS_TTL

Value	Mapping in DNS_RPC_SKD_STATE
	The key is not revoked and is used with the rolled over key-set.
DNS_SKD_ROLL_STATE_MAX 0x00000005	DNS_SKD_STATUS_KSK_WAITING_FOR_DS_T TL
DNS_SKD_STATUS_ROLL_ERROR 0x00000009	This signing key descriptor experienced an unrecoverable error during the key rollover.

**fManualTrigger:** TRUE if the rollover was triggered manually.

**dwPreRollEventFired:** Specifies which prerollover event has been fired.

**ftNextKeyGenerationTime:** The time at which the next key was added to the zone.

**dwRevokedOrSwappedDnskeysLength:** The length of the revoked or swapped DNSKEY RRSet.

**pRevokedOrSwappedDnskeysBuffer:** Presigned DNSKEY RRSet for ZSK swap. This is a pointer to a pointer that points to a buffer containing the enumerated records. The buffer is a series of structures beginning with a [DNS\\_RPC\\_NODE structure \(section 2.2.2.2.3\)](#). The records for the node will be represented by a series of [DNS\\_RPC\\_RECORD structures \(section 2.2.2.2.5\)](#). The number of **DNS\_RPC\_RECORD** structures following a **DNS\_RPC\_NODE** structure is given by the wRecordCount member of **DNS\_RPC\_NODE**.

**dwFinalDnskeysLength:** Length of presigned DNSKEY RRSet.

**pFinalDnskeys:** Presigned DNSKEY RRSet Post Rollover. This is a pointer to a pointer that points to a buffer containing the enumerated records. The buffer is a series of structures beginning with a **DNS\_RPC\_NODE** structure. The records for the node will be represented by a series of **DNS\_RPC\_RECORD** structures. The number of **DNS\_RPC\_RECORD** structures following a **DNS\_RPC\_NODE** structure is given by the wRecordCount member of **DNS\_RPC\_NODE**.

**eActiveKeyScope:** Signing key scope for the SKD's active key. The signing key scope is defined in [KeySignScope \(section 2.2.6.1.2\)](#).

**eStandbyKeyScope:** Signing key scope for the SKD's standby key. The signing key scope is defined in **KeySignScope**.

**eNextKeyScope:** Signing key scope for the SKD's next key. The signing key scope is defined in **KeySignScope**.

## 2.2.7 Application Directory Partition Messages

### 2.2.7.1 Enumerations and Constants

#### 2.2.7.1.1 DNS\_RPC\_DP\_FLAGS

The **DNS\_RPC\_DP\_FLAGS** enumeration is used by the DNS server to indicate the state of an application directory partition. [<49>](#) Any combination of the values below MAY be specified, with the exception that, of the following values, more than one MUST NOT be specified: "DNS\_DP\_LEGACY", "DNS\_DP\_DOMAIN\_DEFAULT", and "DNS\_DP\_FOREST\_DEFAULT".

If a root hint zone is found in any application directory partition that is not marked with either "DNS\_DP\_LEGACY" or "DNS\_DP\_DOMAIN\_DEFAULT", then the DNS server MUST ignore it.

Constant/value	Description
DNS_DP_AUTOCREATED 0x00000001	The application directory partition was automatically created by the DNS server. This flag is set whenever either the DNS_DP_DOMAIN_DEFAULT or DNS_DP_FOREST_DEFAULT flags are set.
DNS_DP_LEGACY 0x00000002	This application directory partition represents the default naming context in the directory server. This flag is set when the application directory partition's <b>DN</b> matches "CN=MicrosoftDNS, CN=System" appended with the value of the <b>defaultNamingContext</b> attribute of the root DN.
DNS_DP_DOMAIN_DEFAULT 0x00000004	This application directory partition is replicated across all DNS servers in the <b>Active Directory domain</b> . This flag is set when the application directory partition's DN matches the value of the "DomainDirectoryPartitionBaseName" DNS server <b>string</b> property (surrounded by "CN=" and ",") and appended with the value of the <b>defaultNamingContext</b> attribute of the root DN.
DNS_DP_FOREST_DEFAULT 0x00000008	This application directory partition is replicated across all DNS servers in the <b>Active Directory forest</b> . This flag is set when the application directory partition's DN matches the value of the "ForestDirectoryPartitionBaseName" DNS server <b>string</b> property (surrounded by "CN=" and ",") appended with the value of the <b>rootDomainNamingContext</b> attribute of the root DN.
DNS_DP_ENLISTED 0x00000010	This flag indicates that the DNS server is enlisted in this application directory partition. It is set when the value of the local directory server's <b>dsServiceName</b> attribute (the DN for the local directory server) is present in either the <b>msDS-NC-Replica-Locations</b> or <b>msDS-NC-RO-Replica-Locations</b> <50> attributes (sections 2.365-2.366) of the application directory partition <b>crossRef object</b> (see section 6.1.1.2.1.1.5).
DNS_DP_DELETED 0x00000020	This application directory partition is in the process of being deleted by the directory server. This flag is set when, during a poll of the application directory partitions, an application directory partition that was present during a previous poll is no longer present. If this application directory partition is not present in the directory server the next time the DNS server polls for application directory partition information, the DNS server MUST remove all zones stored in this application directory partition from the in-memory DNS Zone Table (section 3.1.1) and MUST remove this application directory partition from the in-memory Application Directory Partition Table (section 3.1.1).

## 2.2.7.2 Structures

### 2.2.7.2.1 DNS\_RPC\_DP\_INFO

The DNS\_RPC\_DP\_INFO structure represents the current state of an application directory partition on the directory server. <51>

```
typedef struct _DnssrvRpcDirectoryPartition {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [string] char* pszDpFqdn;
    [string] wchar_t* pszDpDn;
    [string] wchar_t* pszCrDn;
```

```

DWORD dwFlags;
DWORD dwZoneCount;
DWORD dwState;
DWORD dwReserved[3];
[string] wchar_t* pwszReserved[3];
[range(0,10000)] DWORD dwReplicaCount;
[size_is(dwReplicaCount)] PDNS_RPC_DP_REPLICA ReplicaArray[];
} DNS_RPC_DP_INFO,
*PDNS_RPC_DP_INFO;

```

**dwRpcStructureVersion:** The structure version number; this value MUST be set to 0x00000000.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**pszDpFqdn:** A pointer to a null-terminated UTF-8 **string** that specifies the FQDN of the application directory partition. This value is read from the **dnsRoot** attribute of the partition crossRef object (see **pszCrDn**) converted to UTF-8 for this application directory partition.

**pszDpDn:** A pointer to a null-terminated Unicode string that specifies the distinguished name for the application directory partition **naming context root** object. This is the value of the **nCName** attribute of the application directory partition crossRef object (see **pszCrDn**).

**pszCrDn:** A pointer to a null-terminated Unicode string that specifies the distinguished name for the application directory partition crossRef object (located beneath "CN=Partitions, CN=Configuration, <Forest DN>").

**dwFlags:** The application directory partition properties; this MUST be set to a combination of allowed values for DNS\_RPC\_DP\_FLAGS (section [2.2.7.1.1](#)).

**dwZoneCount:** The number of zones from this application directory partition that are loaded in the DNS server's memory. This value is incremented or decremented in the Application Directory Partition Table whenever a DNS Zone Table entry corresponding to a zone in this Application Directory Partition is initialized or deleted, respectively.

**dwState:** The current state of this application directory partition. This MUST be set to one of the following values:

Value	Meaning	Source
DNS_DP_OKAY 0x00000000	The application directory partition is running and ready for all operations.	The Application Directory Partition naming context root object's <b>instanceType</b> attribute has neither DS_INSTANCETYPE_NC_COMING (0x00000010), nor the DS_INSTANCETYPE_NC_GOING (0x00000020) bit set.
DNS_DP_STATE_REPL_INCOMING 0x00000001	The application directory partition is replicating onto the <b>directory</b> server but has not completed an initial synchronization so will be ignored for	The Application Directory Partition naming context root object's <b>instanceType</b> attribute has the ( DS_INSTANCETYPE_NC_COMING (0x00000010) bit set.

Value	Meaning	Source
	the time being.	
DNS_DP_STATE_REPL_OUTGOING 0x00000002	The application directory partition is being deleted from the directory server and so will be ignored.	The Application Directory Partition naming context root object's <b>instanceType</b> attribute has the S_INSTANCETYPE_NC_GOING ( 0x00000020) bit set.
DNS_DP_STATE_UNKNOWN 0x00000003	The application directory partition state is unavailable for unknown reasons.	The Application Directory Partition naming context root object's <b>instanceType</b> attribute is unavailable due to an error condition.

**dwReserved:** MUST be set to zero when sent and MUST be ignored on receipt.

**pwszReserved:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwReplicaCount:** The number of replication locations for the application directory partition. This value MUST be between 0 and 10000. This value is calculated from the values of the **msDS-NC-Replica-Locations** and **msDS-NC-RO-Replica-Locations** <52> attributes of the application directory partition crossRef object (see **pszCrDn**), as the sum of the number of DNS listed in each attribute.

**ReplicaArray:** Array of DNS\_RPC\_DP\_REPLICA (section 2.2.7.2.2), that contains information about replication locations for this application directory partition. This structure is populated from the values of the **msDS-NC-Replica-Locations** (section 2.365) and **msDS-NC-RO-Replica-Locations** <53> (section 2.366) attributes of the application directory partition crossRef object (see **pszCrDn**). Failure to read any of those attributes will be treated as if no replica exists for that attribute.

### 2.2.7.2.2 DNS\_RPC\_DP\_REPLICA

The DNS\_RPC\_DP\_REPLICA structure contains information about an application directory partition replica by giving a distinguished name which can be used to uniquely identify the replica. <54>

```
typedef struct _DnssrvRpcDirectoryPartitionReplica {
    [string] wchar_t* pszReplicaDn;
} DNS_RPC_DP_REPLICA,
*PDNS_RPC_DP_REPLICA;
```

**pszReplicaDn:** A pointer to a null-terminated Unicode string that specifies the distinguished name that identifies a specific directory server.

### 2.2.7.2.3 DNS\_RPC\_DP\_ENUM

The DNS\_RPC\_DP\_ENUM structure contains abbreviated information about an application directory partition. <55>

```
typedef struct _DnssrvRpcDirectoryPartitionEnum {
    DWORD dwRpcStructureVersion;
```

```

    DWORD dwReserved0;
    [string] char* pszDpFqdn;
    DWORD dwFlags;
    DWORD dwZoneCount;
} DNS_RPC_DP_ENUM,
*PDNS_RPC_DP_ENUM;

```

**dwRpcStructureVersion:** As specified in section [2.2.7.2.1](#).

**dwReserved0:** As specified in section [2.2.7.2.1](#).

**pszDpFqdn:** As specified in section [2.2.7.2.1](#).

**dwFlags:** As specified in section [2.2.7.2.1](#).

**dwZoneCount:** As specified in section [2.2.7.2.1](#).

#### 2.2.7.2.4 DNS\_RPC\_DP\_LIST

The DNS\_RPC\_DP\_LIST structure contains a list of application directory partition information structures. [<56>](#)

```

typedef struct _DnssrvRpcDirectoryPartitionList {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [range(0,5000)] DWORD dwDpCount;
    [size_is(dwDpCount)] PDNS_RPC_DP_ENUM DpArray[];
} DNS_RPC_DP_LIST,
*PDNS_RPC_DP_LIST;

```

**dwRpcStructureVersion:** As specified in section [2.2.7.2.1](#).

**dwReserved0:** As specified in section [2.2.7.2.1](#).

**dwDpCount:** The number of DNS\_RPC\_DP\_ENUM (section [2.2.7.2.3](#)) structures present in the array pointed to by DpArray.

**DpArray:** An array of DNS\_RPC\_DP\_ENUM structures (section [2.2.7.2.3](#)), containing information about the application directory partitions available to the DNS server.

#### 2.2.7.2.5 DNS\_RPC\_ENLIST\_DP

The DNS\_RPC\_ENLIST\_DP structure contains the information required to create, delete or enumerate application directory partitions. [<57>](#)

```

typedef struct _DnssrvRpcEnlistDirPart {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    [string] char* pszDpFqdn;
    DWORD dwOperation;
} DNS_RPC_ENLIST_DP,
*PDNS_RPC_ENLIST_DP;

```

**dwRpcStructureVersion:** The DNS management structure version number; this value MUST be set to 0x00000001.

**dwReserved0:** As specified in section [2.2.7.2.1](#).

**pszDpFqdn:** As specified in section [2.2.7.2.1](#).

**dwOperation:** The application directory partition operation to be performed by the DNS server; this MUST be set to one of the following values:

Value	Meaning
DNS_DP_OP_CREATE 0x00000001	Create and enlist (DNS_DP_OP_ENLIST) a new application directory partition.
DNS_DP_OP_DELETE 0x00000002	Delete an existing application directory partition. If the application directory partition has been marked DNS_DP_AUTOCREATED, DNS_DP_LEGACY, DNS_DP_DOMAIN_DEFAULT, DNS_DP_FOREST_DEFAULT, or DNS_DP_DELETED, as specified in section <a href="#">2.2.7.1.1</a> , or if the DNS server cannot connect and bind to the <b>FSMO role owner</b> , then the server MUST return an error.
DNS_DP_OP_ENLIST 0x00000003	Enlist this DNS server in an existing application directory partition. If the application directory partition has been marked DNS_DP_ENLISTED or DNS_DP_DELETED, as specified in section <a href="#">2.2.7.1.1</a> , then the DNS server MUST return an error.
DNS_DP_OP_UNENLIST 0x00000004	Un-enlist this DNS server from an existing application directory partition. If the application directory partition has been marked DNS_DP_DELETED, as specified in section <a href="#">2.2.7.1.1</a> , then the DNS server MUST return an error.
DNS_DP_OP_CREATE_DOMAIN 0x00000005	Create a domain partition on the directory server if one does not already exist.
DNS_DP_OP_CREATE_FOREST 0x00000006	Create a forest partition on the directory server if it does not already exist.

### 2.2.7.2.6 DNS\_RPC\_ZONE\_CHANGE\_DP

The DNS\_RPC\_ZONE\_CHANGE\_DP structure contains information required to move a zone to a different application directory partition on the DNS server. [<58>](#)

```
typedef struct _DnssrvRpcZoneChangePartition {  
    DWORD dwRpcStructureVersion;  
    DWORD dwReserved0;  
    [string] char* pszDestPartition;  
} DNS_RPC_ZONE_CHANGE_DP,  
*PDNS_RPC_ZONE_CHANGE_DP;
```

**dwRpcStructureVersion:** As specified in section [2.2.7.2.5](#).



**dwReserved0:** As specified in section [2.2.7.2.1](#).

**pszDestPartition:** A pointer to a null-terminated UTF-8 string that specifies the distinguished name for a new application directory partition to which a zone is to be moved.

## 2.2.8 AutoConfig Messages

### 2.2.8.1 Enumerations and Constants

#### 2.2.8.1.1 DNS\_RPC\_AUTOCONFIG

The **DNS\_RPC\_AUTOCONFIG** enumeration specifies a set of autoconfiguration operations to be immediately performed by the DNS server. Any combination of the values below may be specified, with the exception of the following values, of which at most one value may be specified:

DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT, DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT\_PREPEND, and DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT\_APPEND. An implementation SHOULD support all values in this table. [<59>](#)

If the DNS server will act as the first DNS server for a new domain in a new forest, the following values SHOULD be specified: DNS\_RPC\_AUTOCONFIG\_INTERNAL\_ROOTHINTS, DNS\_RPC\_AUTOCONFIG\_INTERNAL\_FORWARDERS, and DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT.

If the DNS server will act as a DNS server in an existing domain on a writeable domain controller, the following values SHOULD be specified: DNS\_RPC\_AUTOCONFIG\_INTERNAL\_ROOTHINTS, DNS\_RPC\_AUTOCONFIG\_INTERNAL\_FORWARDERS, and DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT\_APPEND. [<60>](#)

In all other cases, including a DNS server for a new child domain or a DNS server operating on a **read only domain controller (RODC)**, the following values SHOULD be specified: DNS\_RPC\_AUTOCONFIG\_INTERNAL\_ROOTHINTS, DNS\_RPC\_AUTOCONFIG\_INTERNAL\_FORWARDERS, and DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT\_PREPEND. [<61>](#)

The DNS server SHOULD perform autoconfiguration in the following order: root hints, forwarders, self-pointing, and zone creation.

The DNS server SHOULD ignore any bit value not specified in the table below, with one exception: A value of 0x00000000 MUST be treated identically to 0xFFFFFFFF (DNS\_RPC\_AUTOCONFIG\_ALL). [<62>](#)

Constant/value	Description
DNS_RPC_AUTOCONFIG_INTERNAL_ROOTHINTS 0x00000001	The server automatically configures root hints. To construct root hints, the server SHOULD send a DNS query of type NS for the DNS root name to each of the DNS servers and for each of the local machine's network adapters. The server SHOULD build its root hints by selecting the set of NS records that appear in each of the aforementioned NS query responses. If the DNS server cannot find a non-empty set of root hints, it SHOULD perform no action.
DNS_RPC_AUTOCONFIG_INTERNAL_FORWARDERS 0x00000002	The server automatically configures forwarders. To construct forwarders, the DNS server SHOULD locate

Constant/value	Description
	<p>a peer DNS server by sending a DNS query of type NS for the domain name specified in the <b>pszNewDomainName</b> field of the DNS_RPC_AUTOCONFIGURE structure specified in section 2.2.8.2.1. For each peer server, the DNS server SHOULD perform a ServerInfo query (section 3.1.4.7). The DNS server SHOULD use the set of forwarders in the result of this query from the first peer available. If no peer servers with forwarders can be found, the DNS server SHOULD collect all of the DNS servers for each of the local machine's network adapters and use the resulting list of IP addresses as the new list of forwarders.</p>
DNS_RPC_AUTOCONFIG_INTERNAL_ZONES 0x00000004	<p>If the AdminConfigured DNS server property (section 3.1.1.1.1) has been set to a nonzero value, the server SHOULD take no action. Otherwise, the server SHOULD check to see whether it is the only DC in the forest. If the server is not a DC or is not the only DC in the forest, the server MUST perform no action. To determine whether the server is the only DC in the forest, it SHOULD perform an LDAP query on the local directory server using the LDAP filter "(objectCategory=ntdsDsa)" with the credentials of the user who initiated the autoconfigure operation or the <b>DNS Server Credentials</b> (section 3.1.1) if user credentials are not available. If the LDAP result count is one, the server can assume that it is the only DC in the forest. If the local directory server does not respond or does not generate a result, then further processing of DNS_RPC_AUTOCONFIG_INTERNAL_ZONES MUST halt, with ERROR_SUCCESS returned.</p> <p>If the AdminConfigured DNS server property (section 3.1.1.1.1) is zero and the DNS server is the only DC in the forest, the DNS server SHOULD look up the locally configured name of the domain of which the server is a member, and construct two zone names: one equal to the domain name and one equal to the domain name prepended by the string "_msdcs.". If neither of these zones currently exists on the DNS server, the DNS server SHOULD create both of these zones.</p>
DNS_RPC_AUTOCONFIG_INTERNAL_SELFPOINT 0x00000010	<p>The server automatically replaces the server list with the appropriate loop-back address. If this flag is specified, then the server MUST ignore the DNS_RPC_AUTOCONFIG_INTERNAL_SELFPOINT_PREPEND and DNS_RPC_AUTOCONFIG_INTERNAL_SELFPOINT_APPEND flags.</p>
DNS_RPC_AUTOCONFIG_INTERNAL_SELFPOINT_PREPEND 0x00000020	<p>The server automatically inserts the appropriate loop-back address at the start of the server list. If this flag is specified, then the server MUST ignore the DNS_RPC_AUTOCONFIG_INTERNAL_SELFPOINT_APPEND flag.</p>

Constant/value	Description
DNS_RPC_AUTOCONFIG_INTERNAL_SELFPOINT_APP END 0x00000040	The server automatically inserts the appropriate loop-back address at the end of the server list.
DNS_RPC_AUTOCONFIG_INTERNAL_RETURN_ERROR S 0x00008000	If this flag is set then the server will return the errors that it encounters while performing auto-configuration; else ERROR_SUCCESS will always be returned.
DNS_RPC_AUTOCONFIG_ALL 0xFFFFFFFF	The server performs all autoconfiguration operations.

## 2.2.8.2 Structures

### 2.2.8.2.1 DNS\_RPC\_AUTOCONFIGURE

The DNS\_RPC\_AUTOCONFIGURE structure contains the information required to auto-configure the DNS server.

```
typedef struct _DnsRpcAutoConfigureLonghorn {
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    DWORD dwAutoConfigFlags;
    DWORD dwReserved1;
    [string] char* pszNewDomainName;
} DNS_RPC_AUTOCONFIGURE,
*PDNS_RPC_AUTOCONFIGURE;
```

**dwRpcStructureVersion:** The structure version number; this value MUST be set to 0x00000000.

**dwReserved0:** MUST be set to zero when sent and MUST be ignored on receipt.

**dwAutoConfigFlags:** The autoconfiguration operation being requested by the client as specified in DNS\_RPC\_AUTOCONFIG (section [2.2.8.1.1](#)).

**dwReserved1:** MUST be set to zero when sent and MUST be ignored on receipt.

**pszNewDomainName:** A pointer to a null-terminated UTF-8 string which contains the name of the directory server domain that this DNS server is about to join.

## 2.2.9 Logging Messages

### 2.2.9.1 Enumerations and Constants

#### 2.2.9.1.1 DNS\_LOG\_LEVELS

The DNS\_LOG\_LEVELS bit field is a 32-bit integer that specifies the various filters and options that can be configured for the DNS server to log packet exchange information to the server log file. There are four layers of filtering:

- Content filter: Filters on the function (that is, the DNS opcode) of the content of a packet.
- Type filter: Filters on whether the packet is a question or an answer.
- Direction filter: Filters on the network direction of the packet (received or sent).
- Transport filter: Filters on the transport mechanism (TCP or UDP).

Since filters are applied independently, and a packet is logged only if allowed by all filters, setting all bits for any given filter to zero indicates that no packets are to be logged.

Bits other than those listed in the following table can be set to any arbitrary value when sent, and MUST be ignored on receipt.

Constant/value	Description
DNS_LOG_LEVEL_QUERY 0x00000001	The server allows query packet exchanges through the content filter.
DNS_LOG_LEVEL_NOTIFY 0x00000010	The server allows packet exchanges related to zone exchange through the content filter.
DNS_LOG_LEVEL_UPDATE 0x00000020	The server allows packet exchanges related to zone updates through the content filter.
DNS_LOG_LEVEL_QUESTIONS 0x00000100	The server allows packets containing questions through the type filter.
DNS_LOG_LEVEL_ANSWERS 0x00000200	The server allows packets containing answers through the type filter.
DNS_LOG_LEVEL_SEND 0x00001000	The server allows packets it sends out through the direction filter.
DNS_LOG_LEVEL_RECV 0x00002000	The server allows packets it receives through the direction filter.
DNS_LOG_LEVEL_UDP 0x00004000	The server allows UDP packet exchange through the transport filter.
DNS_LOG_LEVEL_TCP 0x00008000	The server allows TCP packet exchange through the transport filter.
DNS_LOG_LEVEL_ALL_PACKETS 0x0000FFFF	The server logs operations that fulfill the following filter set: DNS_LOG_LEVEL_SEND or DNS_LOG_LEVEL_RECV, or DNS_LOG_LEVEL_TCP or DNS_LOG_LEVEL_UDP, or DNS_LOG_LEVEL_QUERY or DNS_LOG_LEVEL_NOTIFY or DNS_LOG_LEVEL_UPDATE, or DNS_LOG_LEVEL_QUESTIONS or DNS_LOG_LEVEL_ANSWERS.
DNS_LOG_LEVEL_DS_WRITE 0x00010000	Independent of the values of the other filters, logs Active Directory write operations.
DNS_LOG_LEVEL_DS_UPDATE 0x00020000	Independent of the values of the other filters, logs Active Directory polling operations and operations during DNS updates (secure and unsecure) on Active Directory

Constant/value	Description
	integrated zones.
DNS_LOG_LEVEL_FULL_PACKETS 0x01000000	If allowed by the filters, the server logs the entire packet to the log file.
DNS_LOG_LEVEL_UNMATCHED_RESPONSE 0x02000000	If allowed by the filters, the server logs response packets that do not match any outstanding query. <a href="#">&lt;63&gt;</a>
DNS_LOG_LEVEL_WRITE_THROUGH 0x80000000	If allowed by the filters, the server saves packet logging information to persistent storage.

### 2.2.9.1.2 DNS\_EVENTLOG\_TYPES

The DNS server can enable several levels of event-logging. This MUST be set to one of the values specified below:

Constant/value	Description
EVENT_LOG_SUCCESS 0x00000000	The server will log events for all successful operations.
EVENT_LOG_ERROR_TYPE 0x00000001	The server will log events for all operations that result in an error.
EVENT_LOG_WARNING_TYPE 0x00000002	The server will log events for all operations that result in a warning.
EVENT_LOG_INFORMATION_TYPE 0x00000004	The server will log events for all operations for informative purposes.

## 2.2.10 Server Statistics Messages

### 2.2.10.1 Enumerations and Constants

#### 2.2.10.1.1 DNSSRV\_STATID\_TYPES

The DNSSRV\_STATID\_TYPES is a 32-bit integer that enumerates the possible types of DNS server statistics. When requesting statistics, these values may be combined to request multiple statistics buffers in the same operation. When statistics are returned, each DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)) contains a value in the **StatId** field with a single bit set to indicate the type of statistics that are contained in the associated buffer.

Constant/value	Description
DNSSRV_STATID_TIME 0x00000001	If the <b>StatId</b> field is set in the request, a DNSSRV_TIME_STATS (section <a href="#">2.2.10.2.4</a> ) structure will be included in the output buffer.
DNSSRV_STATID_QUERY 0x00000002	If the <b>StatId</b> field is set in the request, a DNSSRV_QUERY_STATS (section <a href="#">2.2.10.2.5</a> ) structure will be included in the output buffer.

Constant/value	Description
DNSSRV_STATID_QUERY2 0x00000004	If the <b>StatId</b> field is set in the request, a DNSSRV_QUERY2_STATS (section <a href="#">2.2.10.2.6</a> ) structure will be included in the output buffer.
DNSSRV_STATID_RECURSE 0x00000008	If the <b>StatId</b> field is set in the request, a DNSSRV_RECURSE_STATS (section <a href="#">2.2.10.2.7</a> ) structure will be included in the output buffer.
DNSSRV_STATID_MASTER 0x00000010	If the <b>StatId</b> field is set in the request, a DNSSRV_MASTER_STATS (section <a href="#">2.2.10.2.9</a> ) structure will be included in the output buffer.
DNSSRV_STATID_SECONDARY 0x00000020	If the <b>StatId</b> field is set in the request, a DNSSRV_SECONDARY_STATS (section <a href="#">2.2.10.2.10</a> ) structure will be included in the output buffer.
DNSSRV_STATID_WINS 0x00000040	If the <b>StatId</b> field is set in the request, a DNSSRV_WINS_STATS (section <a href="#">2.2.10.2.11</a> ) structure will be included in the output buffer.
DNSSRV_STATID_WIRE_UPDATE 0x00000100	If the <b>StatId</b> field is set in the request, a DNSSRV_UPDATE_STATS (section <a href="#">2.2.10.2.12</a> ) structure will be included in the output buffer.
DNSSRV_STATID_SKWANSEC 0x00000200	If the <b>StatId</b> field is set in the request, a DNSSRV_SKWANSEC_STATS (section <a href="#">2.2.10.2.13</a> ) structure will be included in the output buffer.
DNSSRV_STATID_DS 0x00000400	If the <b>StatId</b> field is set in the request, a DNSSRV_DS_STATS (section <a href="#">2.2.10.2.14</a> ) structure will be included in the output buffer.
DNSSRV_STATID_NONWIRE_UPDATE 0x00000800	If the <b>StatId</b> field is set in the request, a DNSSRV_UPDATE_STATS (section <a href="#">2.2.10.2.12</a> ) structure will be included in the output buffer.
DNSSRV_STATID_MEMORY 0x00010000	If the <b>StatId</b> field is set in the request, a DNSSRV_MEMORY_STATS (section <a href="#">2.2.10.2.16</a> ) structure will be included in the output buffer.
DNSSRV_STATID_TIMEOUT 0x00020000	If the <b>StatId</b> field is set in the request, a DNSSRV_TIMEOUT_STATS (section <a href="#">2.2.10.2.17</a> ) structure will be included in the output buffer.
DNSSRV_STATID_DBASE 0x00040000	If the <b>StatId</b> field is set in the request, a DNSSRV_DBASE_STATS (section <a href="#">2.2.10.2.18</a> ) structure will be included in the output buffer.
DNSSRV_STATID_RECORD 0x00080000	If the <b>StatId</b> field is set in the request, a DNSSRV_RECORD_STATS (section <a href="#">2.2.10.2.19</a> ) structure will be included in the output buffer.
DNSSRV_STATID_PACKET 0x00100000	If the <b>StatId</b> field is set in the request, a DNSSRV_PACKET_STATS (section <a href="#">2.2.10.2.20</a> ) structure will be included in the output buffer.
DNSSRV_STATID_NBSTAT	If the <b>StatId</b> field is set in the request, a DNSSRV_NBSTAT_STATS (section <a href="#">2.2.10.2.21</a> ) structure will be

Constant/value	Description
0x00200000	included in the output buffer.
DNSSRV_STATID_ERRORS 0x00400000	If the <b>StatId</b> field is set in the request, a DNSSRV_ERROR_STATS (section <a href="#">2.2.10.2.23</a> ) structure will be included in the output buffer.
DNSSRV_STATID_CACHE 0x00800000	If the <b>StatId</b> is set in the request, a DNSSRV_CACHE_STATS (section <a href="#">2.2.10.2.24</a> ) structure will be included in the output buffer.
DNSSRV_STATID_DNSSEC 0x01000000	If the <b>StatId</b> is set in the request, a DNSSRV_CACHE_DNSSEC (section <a href="#">2.2.10.2.8</a> ) structure will be included in the output buffer. <64>
DNSSRV_STATID_PRIVATE 0x10000000	If the <b>StatId</b> is set in the request, a DNSSRV_PRIVATE_STATS (section <a href="#">2.2.10.2.22</a> ) structure will be included in the output buffer.

## 2.2.10.2 Structures

### 2.2.10.2.1 DNSSRV\_STAT\_HEADER

The DNSSRV\_STAT\_HEADER precedes each DNSSRV\_STAT (section [2.2.10.2.2](#)) structure which provides DNS server runtime statistics. This structure MUST be formatted as follows:

```
typedef struct _DnsStatHeader {
    DWORD StatId;
    WORD wLength;
    BOOLEAN fClear;
    UCHAR fReserved;
} DNSSRV_STAT_HEADER,
*PDNSSRV_STAT_HEADER;
```

**StatId:** The type of statistics contained in the DNSSRV\_STAT structure. This value MUST be set to one of the allowed values specified in section [2.2.10.1.1](#).

**wLength:** The length, in bytes, of the **Buffer** member in the DNSSRV\_STAT structure.

**fClear:** A Boolean value that indicates whether the server is to clear the statistics buffer for the server attribute indicated at by StatId.

**fReserved:** MUST be set to zero when sent and MUST be ignored on receipt.

### 2.2.10.2.2 DNSSRV\_STATS

The DNSSRV\_STATS structure carries server statistics information. This structure MUST be interpreted as one of the more specific statistics structures specified in sections [2.2.10.2.4](#) through [2.2.10.2.24](#), depending upon the StatId value in the Header member. This structure MUST be formatted as follows:

```
typedef struct _DnsStat {
    DNSSRV_STAT_HEADER Header;
    BYTE Buffer[1];
}
```

```

} DNSSRV_STAT,
 *PDNSSRV_STAT,
 *PDNSSRV_STATS;

```

**Header:** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**Buffer:** A variable length array of bytes that contains information specific to the type of DNS server statistics, as specified by the StatId value in the Header.

### 2.2.10.2.3 DNS\_SYSTEMTIME

The DNS\_SYSTEMTIME structure stores time values for DNS statistics. It is always populated by the server, which MUST supply valid values. This structure MUST be formatted as follows:

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
wYear																wMonth															
wDayOfWeek																wDay															
wHour																wMinute															
wSecond																wMillisecond															

**wYear (2 bytes):** The year, as a 16-bit, unsigned integer. Valid values are from 1601 to 30827.

**wMonth (2 bytes):** The month from 1 (January) to 12 (December), as a 16-bit, unsigned integer.

**wDayOfWeek (2 bytes):** The day of the week from 0 (Sunday) to 6 (Saturday), as a 16-bit, unsigned integer.

**wDay (2 bytes):** The day of the month from 1 to 31, as a 16-bit, unsigned integer.

**wHour (2 bytes):** The hour from 0 to 23, as a 16-bit, unsigned integer.

**wMinute (2 bytes):** The minute from 0 to 59, as a 16-bit, unsigned integer.

**wSecond (2 bytes):** The second from 0 to 59, as a 16-bit, unsigned integer.

**wMillisecond (2 bytes):** The millisecond from 0 to 999, as a 16-bit, unsigned integer.

### 2.2.10.2.4 DNSSRV\_TIME\_STATS

The DNSSRV\_TIME\_STATS structure has the DNS server's time-related statistics. This structure MUST be formatted as follows:



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
Header																															
...																															
ServerStartTimeSeconds																															
LastClearTimeSeconds																															
SecondsSinceServerStart																															
SecondsSinceLastClear																															
ServerStartTime																															
...																															
...																															
...																															
LastClearTime																															
...																															
...																															
...																															

**Header (8 bytes):** A structure of type `DNSSRV_STAT_HEADER` (section [2.2.10.2.1](#)).

**ServerStartTimeSeconds (4 bytes):** The number of seconds that has elapsed since the server machine was last restarted, that is, the operating system uptime in seconds, as a 32-bit unsigned integer. If the value is greater than `0xFFFFFFFF`, then the value modulo `0x100000000` is stored in the field.

**LastClearTimeSeconds (4 bytes):** The number of seconds that elapsed between the time the server machine was restarted and the last time the server statistics were cleared, that is, the operating system uptime in seconds at the time of the last statistics reset, as a 32-bit unsigned integer. If the value is greater than `0xFFFFFFFF`, then the value modulo `0x100000000` is stored in the field.

**SecondsSinceServerStart (4 bytes):** The number of seconds since the server started, that is, the uptime of the DNS server software in seconds, as a 32-bit unsigned integer. If the value is greater than `0xFFFFFFFF`, then the value modulo `0x100000000` is stored in the field.

**SecondsSinceLastClear (4 bytes):** The number of seconds since the last time that the server statistics were cleared, as a 32-bit unsigned integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ServerStartTime (16 bytes):** A DNS\_SYSTEMTIME (section [2.2.10.2.3](#)) structure that contains the time the server started.

**LastClearTime (16 bytes):** A DNS\_SYSTEMTIME (section [2.2.10.2.3](#)) structure that contains the time the server statistics was last cleared.

### 2.2.10.2.5 DNSSRV\_QUERY\_STATS

DNSSRV\_QUERY\_STATS defines a structure that carries the DNS server's statistics values related to query processing over different transports. This structure MUST be formatted as follows:

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
Header																															
...																															
UdpQueries																															
UdpResponses																															
UdpQueriesSent																															
UdpResponsesReceived																															
TcpClientConnections																															
TcpQueries																															
TcpResponses																															
TcpQueriesSent																															
TcpResponsesReceived																															

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**UdpQueries (4 bytes):** The cumulative number of queries received over UDP, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**UdpResponses (4 bytes):** The cumulative number of query responses sent over UDP, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**UdpQueriesSent (4 bytes):** The cumulative number of queries sent over UDP by this server to other remote servers, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**UdpResponsesReceived (4 bytes):** The cumulative number of query responses received over UDP by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TcpClientConnections (4 bytes):** The cumulative number of TCP connections accepted by this server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TcpQueries (4 bytes):** The cumulative number of queries received over TCP, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TcpResponses (4 bytes):** The cumulative number of query responses sent over TCP, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TcpQueriesSent (4 bytes):** The cumulative number of queries sent over TCP by this server to other remote servers, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TcpResponsesReceived (4 bytes):** The cumulative number of query responses over TCP received by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

### 2.2.10.2.6 DNSSRV\_QUERY2\_STATS

The DNSSRV\_QUERY2\_STATS structure contains DNS server statistics related to query processing by type. This structure MUST be formatted as follows:

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
Header																															
...																															
TotalQueries																															
Standard																															
Notify																															
Update																															
TKeyNego (optional)																															
TypeA																															

TypeNs
TypeSoa
TypeMx
TypePtr
TypeSrv
TypeAll
TypeIxfr
TypeAxfr
TypeOther

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**TotalQueries (4 bytes):** The total number of queries received by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**Standard (4 bytes):** The number of standard DNS queries received by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**Notify (4 bytes):** The number of zone notification requests received by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**Update (4 bytes):** The number of dynamic update requests received by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TKeyNego (4 bytes):** The number of TKEY [\[RFC2930\]](#) negotiation requests received by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field. This field SHOULD be present. A client can tell whether the field is present based on the size of the buffer holding this structure. [<65>](#)

**TypeA (4 bytes):** The number of queries received for record type DNS\_TYPE\_A, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TypeNs (4 bytes):** The number of queries received for record type DNS\_TYPE\_NS, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TypeSoa (4 bytes):** The number of queries received for record type DNS\_TYPE\_SOA, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TypeMx (4 bytes):** The number of queries received for record type DNS\_TYPE\_MX, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TypePtr (4 bytes):** The number of queries received for record type DNS\_TYPE\_PTR, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TypeSrv (4 bytes):** The number of queries received for record type DNS\_TYPE\_SRV, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TypeAll (4 bytes):** The number of queries received for record type DNS\_TYPE\_ALL, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TypeIxfr (4 bytes):** The number of queries received for record type DNS\_TYPE\_IXFR, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TypeAxfr (4 bytes):** The number of queries received for record type DNS\_TYPE\_AXFR, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TypeOther (4 bytes):** The number of queries received for any other record type not mentioned above, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

### 2.2.10.2.7 DNSSRV\_RECURSE\_STATS

The DNSSRV\_RECURSE\_STATS structure has the DNS server's statistics related to recursive resource record lookups. This structure MUST be formatted as follows.

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		
Header																																	
...																																	
ReferralPasses																																	
QueriesRecursed																																	
OriginalQuestionRecursed																																	
AdditionalRecursed																																	
TotalQuestionsRecursed																																	
Retries																																	

LookupPasses
Forwards
Sends
Responses
ResponseUnmatched
ResponseMismatched (optional)
ResponseFromForwarder
ResponseAuthoritative
ResponseNotAuth
ResponseAnswer
ResponseNameError
ResponseRcode
ResponseEmpty
ResponseDelegation
ResponseNonZoneData
ResponseUnsecure
ResponseBadPacket
SendResponseDirect
ContinueCurrentRecursion
ContinueCurrentLookup
ContinueNextLookup
RootNsQuery
RootNsResponse

CacheUpdateAlloc
CacheUpdateResponse
CacheUpdateFree
CacheUpdateRetry
SuspendedQuery
ResumeSuspendedQuery
PacketTimeout
FinalTimeoutQueued
FinalTimeoutExpired
Failures
RecursionFailure
ServerFailure
PartialFailure
CacheUpdateFailure
RecursePassFailure
FailureReachAuthority
FailureReachPreviousResponse
FailureRetryCount
TcpTry
TcpConnectFailure
TcpConnect
TcpQuery
TcpResponse

TcpDisconnect
DiscardedDuplicateQueries (optional)
DuplicateCoalesedQueries (optional)
GnzLocalQuery (optional)
GnzRemoteQuery (optional)
GnzRemoteResponse (optional)
GnzRemoteResponseCacheSuccess (optional)
GnzRemoteResponseCacheFailure (optional)
CacheLockingDiscards (optional)

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**ReferralPasses (4 bytes):** The number of times the server returned a referral value, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**QueriesRecursed (4 bytes):** The number of queries received that required recursive lookups, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**OriginalQuestionRecursed (4 bytes):** The number of new recursive queries initiated, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**AdditionalRecursed (4 bytes):** The number of recursions performed to return additional data or CNAME, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TotalQuestionsRecursed (4 bytes):** The number of total recursions including original and additional, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**Retries (4 bytes):** The number of retries performed for recursive queries sent by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**LookupPasses (4 bytes):** The number of recursive lookups performed, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**Forwards (4 bytes):** The number of recursive queries sent to forwarding servers, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.



**Sends (4 bytes):** The total number of recursive queries sent by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**Responses (4 bytes):** The number of query responses received by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseUnmatched (4 bytes):** The number of responses received for which an outstanding query with a matching transaction-id could not be located, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseMismatched (4 bytes):** The number of responses received for which an outstanding query with a matching transaction-id was located but response was invalid for the query, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field. This field SHOULD be present. [<66>](#) A client can tell whether the field is present based on the size of the buffer holding this structure.

**ResponseFromForwarder (4 bytes):** The number of responses received from forwarders, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseAuthoritative (4 bytes):** The number of responses received from the server authoritative for the zone, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseNotAuth (4 bytes):** The number of responses received from a server not authoritative for the zone, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseAnswer (4 bytes):** The number of responses received from other servers for recursive queries, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseNameError (4 bytes):** The number of name errors received by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseRcode (4 bytes):** The number of errors other than name errors received by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseEmpty (4 bytes):** The number of empty responses received from other servers, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseDelegation (4 bytes):** The number of delegation responses received by the server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseNonZoneData (4 bytes):** The number of error responses when a name is not found in the zone, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseUnsecure (4 bytes):** The number of unsecure responses received when the server is configured to receive secure responses, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResponseBadPacket (4 bytes):** The number of bad response packets received, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**SendResponseDirect (4 bytes):** The number of responses that the DNS server received from remote servers and sent directly to clients, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ContinueCurrentRecursion (4 bytes):** The number of additional remote queries generated by the DNS server during normal query processing, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ContinueCurrentLookup (4 bytes):** The number of times the server received a response from a remote DNS server while processing a client query and restarted recursion, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ContinueNextLookup (4 bytes):** The number of times the server started a lookup with the next query, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**RootNsQuery (4 bytes):** The number of times the server sent a query for a root name server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**RootNsResponse (4 bytes):** The number of times the server processed a response from one of its root servers, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**CacheUpdateAlloc (4 bytes):** The number of times the server allocated a query to be sent to update a cache entry, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**CacheUpdateResponse (4 bytes):** The number of times the server received responses for a query sent to update a cache entry, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**CacheUpdateFree (4 bytes):** The number of times the server released a query request or response packet sent to update a cache entry, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**CacheUpdateRetry (4 bytes):** The number of times the server reattempted a query to update cache entry information, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**SuspendedQuery (4 bytes):** The number of times the server suspended sending a query needed to update cache entry information, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ResumeSuspendedQuery (4 bytes):** The number of times the server resumed a suspended query that was needed to update cache entry information, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**PacketTimeout (4 bytes):** The number of timed-out recursive queries, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**FinalTimeoutQueued (4 bytes):** The number of recursive queries enlisted to wait for final time-out before they expire, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**FinalTimeoutExpired (4 bytes):** The number of recursive queries expired without the server receiving any response, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**Failures (4 bytes):** Not used. Senders MUST set this value to zero, and receivers MUST ignore it.

**RecursionFailure (4 bytes):** The number of times the server received failures for recursion queries to remote servers, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**ServerFailure (4 bytes):** The number of times the server sent failures to the client, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**PartialFailure (4 bytes):** The number of times the server received failures for recursion queries to remote servers, when it had already received an answer but was looking up additional records, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**CacheUpdateFailure (4 bytes):** The number of times the server received failure for self-generated cache update recursion queries to remote servers, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**RecursePassFailure (4 bytes):** The number of times the server failed to perform recursive lookups on queries, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**FailureReachAuthority (4 bytes):** The number of times the server failed to perform recursive lookups on queries, because it failed to reach an authoritative server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**FailureReachPreviousResponse (4 bytes):** The number of times the server received failure while performing recursive lookup on queries, because the query recursed back to the domain from which a name server had already responded, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**FailureRetryCount (4 bytes):** Not used. Senders MUST set this value to zero, and receivers MUST ignore it.

**TcpTry (4 bytes):** The number of times the server started a recursive query over TCP, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TcpConnectFailure (4 bytes):** Not used. Senders MUST set this value to zero, and receivers MUST ignore it.

**TcpConnect (4 bytes):** The number of times the server successfully established a TCP connection to send a recursive query, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TcpQuery (4 bytes):** The number of times the server sent a recursive query over TCP, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TcpResponse (4 bytes):** The number of times the server received a recursive query response over TCP, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**TcpDisconnect (4 bytes):** The number of times the server disconnected a connection that was established to send a recursive query over TCP to a remote server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field.

**DiscardedDuplicateQueries (4 bytes):** The number of times the server discarded a query that was received from the same client with the same transaction ID when there was already a query with the same query name, type ID, and transaction ID outstanding, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field. This field **MUST** be present if and only if ResponseMismatched is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**DuplicateCoalescedQueries (4 bytes):** The number of times the server coalesced a query that was received from a client while another query with the same query name and type ID was outstanding at the server for processing, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field. This field **MUST** be present if and only if ResponseMismatched is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**GnzLocalQuery (4 bytes):** The number of times a **GNZ** lookup query was answered locally, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field. This field **SHOULD** [<67>](#) be present but **MUST** be absent if ResponseMismatched is absent. A client can tell whether the field is present based on the size of the buffer holding this structure.

**GnzRemoteQuery (4 bytes):** The number of times a GNZ lookup query was sent to a remote server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field. This field **MUST** be present if and only if GnzLocalQuery is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**GnzRemoteResponse (4 bytes):** The number of times a GNZ lookup query response was received from a remote server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field. This field **MUST** be present if and only if GnzLocalQuery is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**GnzRemoteResponseCacheSuccess (4 bytes):** The number of times a GNZ cache update query response was successfully received from a remote server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field. This field **MUST** be present if and only if GnzLocalQuery is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**GnzRemoteResponseCacheFailure (4 bytes):** The number of times the server received failure for GNZ cache update query requests sent to a remote server, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field. This field **MUST** be present if and only if GnzLocalQuery is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**CacheLockingDiscards (4 bytes):** The number of times the server discarded a cache update due to cache record locking, as an unsigned 32-bit integer. If the value is greater than 0xFFFFFFFF, then the value modulo 0x100000000 is stored in the field. This field **SHOULD** [<68>](#) be present. A client can tell whether the field is present based on the size of the buffer holding this structure.

### 2.2.10.2.8 DNSSRV\_DNSSEC\_STATS

The DNSSRV\_DNSSEC\_STATS structure has the DNS server statistics related a DNSSEC signature or DS digest hash succeeding or failing. [<69>](#)

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
Header																															
...																															
SuccessfulValidations																															
FailedValidations																															
RecursionFailures																															

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**SuccessfulValidations (4 bytes):** The number of times a validation attempt on a DNSSEC signature or DS digest hash succeeded.

**FailedValidations (4 bytes):** The number of times a validation attempt on a DNSSEC signature or DS digest hash failed.

**RecursionFailures (4 bytes):** The number of times a validating recursive name resolution query attempt failed while fetching DNSSEC data.

### 2.2.10.2.9 DNSSRV\_MASTER\_STATS

The DNSSRV\_MASTER\_STATS structure has the DNS server statistics related to overall DNS protocol processing. This structure **MUST** be formatted as follows:

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
Header																															

...
NotifySent
Request
NameError
FormError
AxfrLimit
Refused
RefuseSecurity
RefuseShutdown
RefuseLoading (optional)
RefuseZoneLocked
RefuseServerFailure
RefuseNotAuth (optional)
RefuseReadOnly (optional)
Failure
AxfrRequest
AxfrSuccess
StubAxfrRequest (optional)
IxfrRequest
IxfrNoVersion
IxfrUpdateSuccess
IxfrTcpRequest
IxfrTcpSuccess

IxfrAxfr
IxfrUdpRequest
IxfrUdpSuccess
IxfrUdpForceTcp
IxfrUdpForceAxfr

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**NotifySent (4 bytes):** The number of update notifications sent to secondaries by the server.

**Request (4 bytes):** The number of zone transfer requests received by the server.

**NameError (4 bytes):** The number of name error responses returned by the server.

**FormError (4 bytes):** The number of invalid format error responses returned by the server.

**AxfrLimit (4 bytes):** The number of **full zone transfer** requests rejected due to time restrictions between successive full zone transfers.

**Refused (4 bytes):** The total number of times the server rejected requests for dynamic updates or zone transfers.

**RefuseSecurity (4 bytes):** The number of times the server rejected zone transfer requests due to secondary security restrictions.

**RefuseShutdown (4 bytes):** The number of times the server rejected zone transfer requests because zone transfer was disabled or because the requesting IP address was not permitted to transfer the zone.

**RefuseLoading (4 bytes):** The number of times the server rejected zone transfer requests, due to a zone not being fully loaded. This field SHOULD be present, but MUST be absent if StubAxfrRequest is absent. [<70>](#)A client can tell whether the field is present based on the size of the buffer holding this structure.

**RefuseZoneLocked (4 bytes):** The number of times the server rejected zone transfer requests, due to the zone already being locked for some operation.

**RefuseServerFailure (4 bytes):** The number of times the server rejected zone transfer requests, due to processing failures at the server.

**RefuseNotAuth (4 bytes):** The number of times the server rejected zone transfer requests, because the zone is not authoritative on the server. This field MUST be present if and only if RefuseLoading is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**RefuseReadOnly (4 bytes):** The number of times the server rejected zone transfer requests, due to the zone being hosted on an RODC. This field MUST be present if and only if RefuseLoading is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**Failure (4 bytes):** The number of times the server hit a zone transfer failure.

**AxfrRequest (4 bytes):** The number of full zone transfer requests received by the server.

**AxfrSuccess (4 bytes):** The number of full zone transfers successfully completed by the server.

**StubAxfrRequest (4 bytes):** The number of full zone transfer requests received by the server for stub zones. This field SHOULD be present. <71> A client can tell whether the field is present based on the size of the buffer holding this structure.

**IxfrRequest (4 bytes):** The number of **incremental zone transfer** requests received by the server.

**IxfrNoVersion (4 bytes):** The number of servers that received an incremental zone transfer request, but there was not a suitable version number available for incremental zone transfer.

**IxfrUpdateSuccess (4 bytes):** The number of success responses for incremental zone transfer sent by the server.

**IxfrTcpRequest (4 bytes):** The number of incremental zone transfer requests received by the server over TCP.

**IxfrTcpSuccess (4 bytes):** The number of success responses for incremental zone transfers sent by the server over TCP.

**IxfrAxfr (4 bytes):** The number of incremental zone transfer requests received by the server, which required a full zone transfer.

**IxfrUdpRequest (4 bytes):** The number of incremental zone transfer requests received by the server over UDP.

**IxfrUdpSuccess (4 bytes):** The number of success responses for incremental zone transfers sent by the server over UDP.

**IxfrUdpForceTcp (4 bytes):** The number of incremental zone transfer requests received by the server over UDP, for which the server responded using TCP.

**IxfrUdpForceAxfr (4 bytes):** The number of incremental zone transfer requests received by the server over UDP, for which the server responded with a full zone transfer.

## 2.2.10.2.10 DNSSRV\_SECONDARY\_STATS

The DNSSRV\_SECONDARY\_STATS structure has the DNS server statistics related to secondary zone processing.

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
Header																															
...																															
NotifyReceived																															
NotifyInvalid																															



NotifyPrimary
NotifyNonPrimary (optional)
NotifyNoVersion
NotifyNewVersion
NotifyCurrentVersion
NotifyOldVersion
NotifyMasterUnknown
SoaRequest
SoaResponse
SoaResponseInvalid
SoaResponseNameError
AxfrRequest
AxfrResponse
AxfrSuccess
AxfrRefused
AxfrInvalid
StubAxfrRequest (optional)
StubAxfrResponse (optional)
StubAxfrSuccess (optional)
StubAxfrRefused (optional)
StubAxfrInvalid (optional)
IxfrUdpRequest
IxfrUdpResponse

IxfrUdpSuccess
IxfrUdpUseTcp
IxfrUdpUseAxfr
IxfrUdpWrongServer
IxfrUdpNoUpdate
IxfrUdpNewPrimary
IxfrUdpFormerr
IxfrUdpRefused
IxfrUdpInvalid
IxfrTcpRequest
IxfrTcpResponse
IxfrTcpSuccess
IxfrTcpAxfr
IxfrTcpFormerr
IxfrTcpRefused
IxfrTcpInvalid

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**NotifyReceived (4 bytes):** The number of zone notifications received by the server.

**NotifyInvalid (4 bytes):** The number of invalid zone notifications received by the server.

**NotifyPrimary (4 bytes):** The number of zone notifications for primary zones received by the server.

**NotifyNonPrimary (4 bytes):** The number of zone notifications for non-primary zones received by the server. This field SHOULD be present. [<72>](#) A client can tell whether the field is present based on the size of the buffer holding this structure.

**NotifyNoVersion (4 bytes):** The number of zone notifications received by the server, for which the server has no SOA.

**NotifyNewVersion (4 bytes):** The number of zone notifications received by the server, where the received SOA has a newer version number than that of the SOA already present on the server.

**NotifyCurrentVersion (4 bytes):** The number of zone notifications received by the server where the received SOA has same version number as that of the SOA already present on the server.

**NotifyOldVersion (4 bytes):** The number of zone notifications received by the server, where the received SOA has an older version number than the SOA already present on the server.

**NotifyMasterUnknown (4 bytes):** The number of zone notifications received by the server, where notifications are received from a server that is not present in the list of masters for the zone.

**SoaRequest (4 bytes):** The number of SOA query requests sent by the server to zone masters, to initiate zone transfer.

**SoaResponse (4 bytes):** The number of SOA responses received by the server from the zone master.

**SoaResponseInvalid (4 bytes):** The number of invalid SOA responses received by the server from the zone master.

**SoaResponseNameError (4 bytes):** Not used, the receiver MUST ignore this value.

**AxfrRequest (4 bytes):** The number of full zone transfer requests sent by the server.

**AxfrResponse (4 bytes):** The number of full zone transfer responses received by the server.

**AxfrSuccess (4 bytes):** The number of full zone transfer success responses received by the server.

**AxfrRefused (4 bytes):** The number of full zone transfer rejection responses received by the server.

**AxfrInvalid (4 bytes):** The number of full zone transfer invalid responses received by the server.

**StubAxfrRequest (4 bytes):** The number of full zone transfer requests sent by the server for stub zones. This field MUST be present if and only if NotifyNonPrimary is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**StubAxfrResponse (4 bytes):** The number of full zone transfer responses received by the server for stub zones. This field MUST be present if and only if NotifyNonPrimary is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**StubAxfrSuccess (4 bytes):** The number of full zone transfer success responses received by the server for stub zones. This field MUST be present if and only if NotifyNonPrimary is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**StubAxfrRefused (4 bytes):** The number of full zone transfer rejection responses received by the server. This field MUST be present if and only if NotifyNonPrimary is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**StubAxfrInvalid (4 bytes):** The number of full zone transfer invalid responses received by the server. This field **MUST** be present if and only if NotifyNonPrimary is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**IxfrUdpRequest (4 bytes):** The number of incremental zone transfer requests sent by the server over UDP.

**IxfrUdpResponse (4 bytes):** The number of incremental zone transfer success responses received by the server over UDP.

**IxfrUdpSuccess (4 bytes):** The number of incremental zone transfer success responses received by the server over UDP.

**IxfrUdpUseTcp (4 bytes):** The number of incremental zone transfer responses received by the server over UDP, indicating that TCP is needed.

**IxfrUdpUseAxfr (4 bytes):** The number of incremental zone transfer responses received by the server over UDP, indicating that full zone transfer is needed.

**IxfrUdpWrongServer (4 bytes):** The number of incremental zone transfer responses received by the server over UDP, where the remote sender is not among the masters for this zone.

**IxfrUdpNoUpdate (4 bytes):** The number of incremental zone transfer responses received by the server over UDP, where no updates were found and hence no zone transfer is needed.

**IxfrUdpNewPrimary (4 bytes):** The number of incremental zone transfer responses received by the server over UDP, where the SOA indicates a new primary server name.

**IxfrUdpFormerr (4 bytes):** The number of incremental zone transfer responses received by the server over UDP, where either the master does not support incremental zone transfer or the master indicated that the zone transfer request was malformed.

**IxfrUdpRefused (4 bytes):** The number of incremental zone transfer rejection responses received by the server over UDP.

**IxfrUdpInvalid (4 bytes):** The number of incremental zone transfer invalid responses received by the server over UDP.

**IxfrTcpRequest (4 bytes):** The number of incremental zone transfer requests sent by the server over TCP.

**IxfrTcpResponse (4 bytes):** The number of incremental zone transfer success responses received by the server over TCP.

**IxfrTcpSuccess (4 bytes):** The number of incremental zone transfer success responses received by the server over TCP.

**IxfrTcpAxfr (4 bytes):** The number of incremental zone transfer responses received by the server over TCP, indicating that full zone transfer is needed.

**IxfrTcpFormerr (4 bytes):** The number of incremental zone transfer responses received by the server over TCP, where either the primary DNS server does not support incremental zone transfer or the primary DNS server indicated that the zone transfer request was malformed.

**IxfrTcpRefused (4 bytes):** The number of incremental zone transfer rejection responses received by the server over TCP.

**IxfrTcpInvalid (4 bytes):** The number of incremental zone transfer invalid responses received by the server over TCP.

### 2.2.10.2.11 DNSSRV\_WINS\_STATS

The DNSSRV\_WINS\_STATS structure has DNS server statistics related to WINS lookups. This structure MUST be formatted as follows:

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
Header																															
...																															
WinsLookups																															
WinsResponses																															
WinsReverseLookups																															
WinsReverseResponses																															

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**WinsLookups (4 bytes):** The number of WINS lookup requests received by the server.

**WinsResponses (4 bytes):** The number of WINS responses sent by the server.

**WinsReverseLookups (4 bytes):** The number of reverse WINS lookup requests received by the server.

**WinsReverseResponses (4 bytes):** The number of reverse WINS lookup responses sent by the server.

### 2.2.10.2.12 DNSSRV\_UPDATE\_STATS

The DNSSRV\_UPDATE\_STATS structure has DNS server statistics related to dynamic updates processing. This structure MUST be formatted as follows:

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
Header																															
...																															
Received																															
Empty																															

NoOps
Completed
Rejected
FormErr
NxDomain
NotImpl
Refused
YxDomain
YxRrset
NxRrset
NotAuth
NotZone
RefusedNonSecure
RefusedAccessDenied
SecureSuccess
SecureContinue
SecureFailure
SecureDsWriteFailure
DsSuccess
DsWriteFailure
unused_was_Collisions
unused_was_CollisionsRead
unused_was_CollisionsWrite

unused_was_CollisionsDsWrite
Queued
Retry
Timeout
InQueue
Forwards
TcpForwards
ForwardResponses
ForwardTimeouts
ForwardInQueue
UpdateType
...
...
...
...
...
...
...
...
...
(UpdateType cont'd for 31 rows)

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**Received (4 bytes):** The number of dynamic update requests received by the server.

**Empty (4 bytes):** The number of empty dynamic update requests received by the server.

**NoOps (4 bytes):** The number of no-op dynamic update requests (such as a dynamic update request with no update records) received by the server.

**Completed (4 bytes):** The number of completed dynamic update requests received by the server.

**Rejected (4 bytes):** The number of dynamic update requests rejected by the server.

**FormErr (4 bytes):** The number of dynamic update requests rejected by the server, due to malformed packets.

**NxDomain (4 bytes):** The number of dynamic update requests rejected by the server, due to name error.

**NotImpl (4 bytes):** The number of dynamic update requests rejected by the server, due to unimplemented functionality.

**Refused (4 bytes):** The number of dynamic update requests rejected by the server, due to malformed packets.

**YxDomain (4 bytes):** The number of dynamic update requests rejected by the server, due to policy restrictions.

**YxRrset (4 bytes):** The number of dynamic update requests rejected by the server, due to an unknown domain name.

**NxRrset (4 bytes):** The number of dynamic update requests rejected by the server, due to an unknown resource record name.

**NotAuth (4 bytes):** The number of dynamic update requests rejected by the server, due to the server not being authoritative for the zone.

**NotZone (4 bytes):** The number of dynamic update requests rejected by the server, due to the zone name not being recognized as one for which it is authoritative.

**RefusedNonSecure (4 bytes):** The number of dynamic update requests rejected by the server, due to a non-secure update request received for a zone where secure updates are required.

**RefusedAccessDenied (4 bytes):** The number of dynamic update requests rejected by the server, due to a failure to update records in the directory server.

**SecureSuccess (4 bytes):** The number of **secure dynamic update** requests received by the server that were successfully applied.

**SecureContinue (4 bytes):** Not used. Senders MUST set this value to zero and the receiver MUST ignore it

**SecureFailure (4 bytes):** The number of secure dynamic update requests received by the server that could not be successfully applied.

**SecureDsWriteFailure (4 bytes):** The number of secure dynamic update requests received by the server that the server failed to update in the directory server.

**DsSuccess (4 bytes):** The number of unsecure dynamic update requests received by the server, that were successfully updated in the directory server.

**DsWriteFailure (4 bytes):** The number of unsecure dynamic update requests received by the server that the server failed to update in the directory server.



**unused\_was\_Collisions (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

**unused\_was\_CollisionsRead (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

**unused\_was\_CollisionsWrite (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

**unused\_was\_CollisionsDsWrite (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

**Queued (4 bytes):** The number of updates packets received that needed to be sent to other remote servers.

**Retry (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

**Timeout (4 bytes):** The number of update packets received, that timed-out while waiting to update the remote server.

**InQueue (4 bytes):** The number of update packets received which are waiting in the update queue for updates to complete on the remote server.

**Forwards (4 bytes):** The number of update packets received that were forwarded to other servers.

**TcpForwards (4 bytes):** The number of update packets received over TCP that were forwarded to other servers.

**ForwardResponses (4 bytes):** The number of response packets received for the update requests that were forwarded to other servers.

**ForwardTimeouts (4 bytes):** The number of update packets which timed out waiting for a response from other servers.

**ForwardInQueue (4 bytes):** The number of update packets forwarded to other servers and which are waiting for a response.

**UpdateType (156 bytes):** An array of counters that keep track of the number of update requests received for different DNS record types. This array has a total of 39 entries from DNS\_TYPE\_ZERO to DNS\_TYPE\_DNAME.

### 2.2.10.2.13 DNSSRV\_SKWANSEC\_STATS

The DNSSRV\_SKWANSEC\_STATS structure has DNS server statistics related to **security context** processing. This structure MUST be formatted as follows:

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
Header																															
...																															
SecContextCreate																															

SecContextFree
SecContextQueue
SecContextQueueInNego
SecContextQueueNegoComplete
SecContextQueueLength
SecContextDequeue
SecContextTimeout
SecPackAlloc
SecPackFree
SecTkeyInvalid
SecTkeyBadTime
SecTsigFormerr
SecTsigEcho
SecTsigBadKey
SecTsigVerifySuccess
SecTsigVerifyFailed

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**SecContextCreate (4 bytes):** The number of security contexts created by the server since the server was started.

**SecContextFree (4 bytes):** The number of security contexts released by the server since the server was started.

**SecContextQueue (4 bytes):** The total number of security contexts queued in the queue for negotiation on the server since the server was started.

**SecContextQueueInNego (4 bytes):** The number of security contexts entered in negotiation since the server was started.

**SecContextQueueNegoComplete (4 bytes):** The number of security contexts that have completed negotiation since the server was started.

- SecContextQueueLength (4 bytes):** The number of security contexts currently queued.
- SecContextDequeue (4 bytes):** The total number of security contexts removed from the queue for negotiation since the server was started.
- SecContextTimeout (4 bytes):** The total number of security contexts in the negotiation list that timed out since the server was started.
- SecPackAlloc (4 bytes):** The number of buffers allocated by the server for use with GSS-API signature validation.
- SecPackFree (4 bytes):** The number of buffers for use with GSS-API signature validation released by the server.
- SecKeyInvalid (4 bytes):** The number of secure DNS update messages from which TKEY was successfully retrieved.
- SecKeyBadTime (4 bytes):** The number of secure DNS update messages that had TKEY with a skewed time stamp.
- SecTsigFormerr (4 bytes):** The number of TSIG records from which signature extraction failed.
- SecTsigEcho (4 bytes):** The number of echo TSIG records received by the server, indicating that the remote server is not security aware. [\[RFC2845\]](#)
- SecTsigBadKey (4 bytes):** The number of TSIG records received for which the cached security context could not be found.
- SecTsigVerifySuccess (4 bytes):** The number of TSIG records received for which the signature was successfully verified.
- SecTsigVerifyFailed (4 bytes):** The number of TSIG records received for which signature verification failed.

#### 2.2.10.2.14 DNSSRV\_DS\_STATS

The DNSSRV\_DS\_STATS structure has DNS server statistics related to directory server processing. The UpdateLists member through DsWriteType member refer to statistical values, when the server is propagating changes from the in-memory database to the directory server.

This structure MUST be formatted as follows:

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		
Header																																	
...																																	
DsTotalNodesRead																																	
DsTotalRecordsRead																																	

DsNodesLoaded
DsRecordsLoaded
DsTombstonesRead
DsUpdateSearches
DsUpdateNodesRead
DsUpdateRecordsRead
UpdateLists
UpdateNodes
UpdateSuppressed
UpdateWrites
UpdateTombstones
UpdateRecordChange
UpdateAgingRefresh
UpdateAgingOn
UpdateAgingOff
UpdatePacket
UpdatePacketPrecon
UpdateAdmin
UpdateAutoConfig
UpdateScavenge
DsNodesAdded
DsNodesModified
DsNodesTombstoned

DsNodesDeleted
DsRecordsAdded
DsRecordsReplaced
DsWriteSuppressed
DsSerialWrites
LdapTimedWrites
LdapWriteTimeTotal
LdapWriteAverage
LdapWriteMax
LdapWriteBucket0
LdapWriteBucket1
LdapWriteBucket2
LdapWriteBucket3
LdapWriteBucket4
LdapWriteBucket5
LdapSearchTime
FailedDeleteDsEntries
FailedReadRecords
FailedLdapModify
FailedLdapAdd
PollingPassesWithDsErrors (optional)
LdapReconnects (optional)
DsWriteType

...
...
...
...
...
...
...
...
...
(DsWriteType cont'd for 32 rows)

- Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).
- DsTotalNodesRead (4 bytes):** The total number of DNS nodes read from the directory server.
- DsTotalRecordsRead (4 bytes):** The total number of resource records read from the directory server.
- DsNodesLoaded (4 bytes):** The number of valid DNS nodes found in the directory server and loaded in memory by the server.
- DsRecordsLoaded (4 bytes):** The number of resource records loaded in memory by the server.
- DsTombstonesRead (4 bytes):** The number of nodes read from the directory server and found in a tombstoned state.
- DsUpdateSearches (4 bytes):** The number of zone update searches performed on the directory server.
- DsUpdateNodesRead (4 bytes):** The number of DNS nodes that were read from the directory server and contained updated information.
- DsUpdateRecordsRead (4 bytes):** The number of resource records that were read from the directory server and contained updated information.
- UpdateLists (4 bytes):** The number of in-memory nodes with an updated list of record.
- UpdateNodes (4 bytes):** The number of in-memory nodes that required an update in the directory server.
- UpdateSuppressed (4 bytes):** The number of in-memory nodes that did not require any write to the directory server.
- UpdateWrites (4 bytes):** The number of in-memory nodes that required writing to the directory server.
- UpdateTombstones (4 bytes):** The number of in-memory nodes that required tombstoning.

**UpdateRecordChange (4 bytes):** The number of in-memory nodes that required record changes.

**UpdateAgingRefresh (4 bytes):** The number of in-memory nodes that required an aging information refresh.

**UpdateAgingOn (4 bytes):** The number of in-memory nodes that required aging to be enabled.

**UpdateAgingOff (4 bytes):** The number of in-memory nodes that required aging to be disabled.

**UpdatePacket (4 bytes):** The number of in-memory nodes modified as a result of update packets being received.

**UpdatePacketPrecon (4 bytes):** The number of in-memory nodes modified as a result of update packets being received with prerequisites (as discussed in [RFC2136](#) section 2.4).

**UpdateAdmin (4 bytes):** The number of in-memory nodes modified as a result of administrator initiated changes.

**UpdateAutoConfig (4 bytes):** The number of in-memory nodes modified as a result of an auto-configure operation.

**UpdateScavenge (4 bytes):** The number of in-memory nodes modified as a result of a scavenging cycle.

**DsNodesAdded (4 bytes):** The number of new nodes added to the directory server.

**DsNodesModified (4 bytes):** The number of nodes modified in the directory server.

**DsNodesTombstoned (4 bytes):** The number of nodes tombstoned in the directory server.

**DsNodesDeleted (4 bytes):** The number of nodes deleted from the directory server.

**DsRecordsAdded (4 bytes):** The number of records added to the directory server.

**DsRecordsReplaced (4 bytes):** The number of records modified or replaced in the directory server.

**DsWriteSuppressed (4 bytes):** The number of records added to the directory server.

**DsSerialWrites (4 bytes):** The number of records that had matching data and hence were not written to the directory server.

**LdapTimedWrites (4 bytes):** The number of times the server performed a timed LDAP write operation.

**LdapWriteTimeTotal (4 bytes):** The cumulative time, in milliseconds, consumed by server-performed timed LDAP write operations.

**LdapWriteAverage (4 bytes):** The average time, in milliseconds, for all server performed timed LDAP write operations since the server was last restarted.

**LdapWriteMax (4 bytes):** The longest duration taken, in milliseconds, for any single server-performed timed LDAP write-operation.

**LdapWriteBucket0 (4 bytes):** The number of LDAP write-operations that took less than 10 milliseconds.

**LdapWriteBucket1 (4 bytes):** The number of LDAP write-operations that took between 10 and 100 milliseconds.

**LdapWriteBucket2 (4 bytes):** The number of LDAP write-operations that took between 100 milliseconds and 1 second.

**LdapWriteBucket3 (4 bytes):** The number of LDAP write-operations that took between 1 and 10 seconds.

**LdapWriteBucket4 (4 bytes):** The number of LDAP write-operations that took between 10 and 100 seconds.

**LdapWriteBucket5 (4 bytes):** The number of LDAP write-operations that took more than 100 seconds.

**LdapSearchTime (4 bytes):** The cumulative time, in milliseconds, consumed by server-performed LDAP searches.

**FailedDeleteDsEntries (4 bytes):** The number of times the server failed to delete entries from the directory server.

**FailedReadRecords (4 bytes):** The number of times the server failed to read records from the directory server.

**FailedLdapModify (4 bytes):** The number of times the server failed to modify records in the directory server.

**FailedLdapAdd (4 bytes):** The number of times the server failed to add entries to the directory server.

**PollingPassesWithDsErrors (4 bytes):** The number of times the server hit failure while polling zones in the directory server. This field SHOULD be present. [<73>](#) A client can tell whether the field is present based on the size of the buffer holding this structure.

**LdapReconnects (4 bytes):** The number of times the server attempted to reconnect to the directory server. This field MUST be present if and only if PollingPassesWithDsErrors is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**DsWriteType (160 bytes):** An array of 32-bit unsigned integers that keeps track of update requests for different DNS record types as specified DNS\_RECORD\_TYPE (section [2.2.2.1.1](#)). There are a total of 40 entries each corresponding to the number of received update requests for different record types in the order specified in DNS\_RECORD\_TYPE (section [2.2.2.1.1](#)), starting from DNS\_TYPE\_ZERO to DNS\_TYPE\_DNAME.

#### 2.2.10.2.15 DNSSRV\_MEMTAG\_STATS

The DNSSRV\_MEMTAG\_STATS structure has DNS server statistics related to memory allocations for a given purpose. This structure is used by DNSSRV\_MEMORY\_STATS (section [2.2.10.2.16](#)) to collect the list of statistics of memory allocated for various purposes. This structure MUST be formatted as follows:



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
Alloc																															
Free																															
Memory																															

**Alloc (4 bytes):** The cumulative number of times memory allocations have been performed for a given purpose.

**Free (4 bytes):** The cumulative number of times memory has been released for a given purpose.

**Memory (4 bytes):** The total size of memory, in bytes, currently in use for a given purpose.

### 2.2.10.2.16 DNSSRV\_MEMORY\_STATS

The DNSSRV\_MEMORY\_STATS structure has DNS server statistics related to memory usage for different operations on the server. It provides statistical information about memory usage since the server started or server statistics were last cleared. The structure supports allocations of two types: blocks of common (but implementation-specific) sizes, and blocks of arbitrary sizes. This allows servers to use a separate internal mechanism to optimize allocations of common sizes if they so choose. This structure **MUST** be formatted as follows:

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
Header																															
...																															
Memory																															
Alloc																															
Free																															
StdUsed																															
StdReturn																															
StdInUse																															
StdMemory																															
StdToHeapAlloc																															

StdToHeapFree
StdToHeapInUse
StdToHeapMemory
StdBlockAlloc
StdBlockUsed
StdBlockReturn
StdBlockInUse
StdBlockFreeList
StdBlockFreeListMemory
StdBlockMemory
MemTags
...
...
...
...
...
...
...
...
...
(MemTags cont'd for 148 rows)

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**Memory (4 bytes):** The total memory currently allocated by the servers, in bytes.

**Alloc (4 bytes):** The cumulative number of times memory was allocated by the server.

**Free (4 bytes):** The cumulative number of times memory was released by the server.

**StdUsed (4 bytes):** The cumulative number of times a common-size block of memory was allocated by the server.

**StdReturn (4 bytes):** The cumulative number of times a common-size block of memory was released by the server.

**StdInUse (4 bytes):** The total number of common-size blocks of allocated memory currently used by the server.

**StdMemory (4 bytes):** The total size, in bytes, of common-size blocks that are currently being used by the server.

**StdToHeapAlloc (4 bytes):** The cumulative number of arbitrary-size blocks of memory allocated from system memory.

**StdToHeapFree (4 bytes):** The cumulative number of arbitrary-size blocks of memory released to system memory.

**StdToHeapInUse (4 bytes):** The number of arbitrary-size blocks of memory currently in use.

**StdToHeapMemory (4 bytes):** The total size of memory, in bytes, currently used by non-standard sized blocks.

**StdBlockAlloc (4 bytes):** The cumulative number of common-size blocks allocated by the server.

**StdBlockUsed (4 bytes):** The cumulative number of common-size blocks allocated from an internal free list.

**StdBlockReturn (4 bytes):** The cumulative number of common-size blocks returned from an internal free list.

**StdBlockInUse (4 bytes):** The number of common-size blocks currently being used.

**StdBlockFreeList (4 bytes):** The number of common-size blocks currently on internal free lists.

**StdBlockFreeListMemory (4 bytes):** The total size of memory, in bytes, of common size blocks currently on internal free lists.

**StdBlockMemory (4 bytes):** The total size of memory, in bytes, of all currently allocated blocks.

**MemTags (624 bytes):** An array of 52 DNSSRV\_MEMTAG\_STATS (section [2.2.10.2.15](#)) specifying memory statistics for various server operations. The table below gives the context applicable to each element of this array, where the Value column indicates the element number.

Value	Meaning
MEMTAG_NONE 0x00000001	Not related to a particular operation.
MEMTAG_PACKET_UDP 0x00000002	UDP Packets.
MEMTAG_PACKET_TCP 0x00000003	TCP Packets.
MEMTAG_NAME	Name-related operations.

<b>Value</b>	<b>Meaning</b>
0x00000004	
MEMTAG_ZONE 0x00000005	Zone operations.
MEMTAG_UPDATE 0x00000006	Name updates.
MEMTAG_UPDATE_LIST 0x00000007	Record update list.
MEMTAG_TIMEOUT 0x00000008	Timeout
MEMTAG_NODEHASH 0x00000009	Node hash.
MEMTAG_DS_DN 0x0000000A	Directory server distinguished name.
MEMTAG_DS_MOD 0x0000000B	Directory server module.
MEMTAG_DS_RECORD 0x0000000C	Directory server records.
MEMTAG_DS_OTHER 0x0000000D	Other directory server related operations.
MEMTAG_THREAD 0x0000000E	Thread management.
MEMTAG_NBSTAT 0x0000000F	NBSTAT packets operations.
MEMTAG_DNSLIB 0x00000010	DNS library management.
MEMTAG_TABLE 0x00000011	Record table operations.
MEMTAG_SOCKET 0x00000012	Socket operations.
MEMTAG_CONNECTION 0x00000013	Connection establishment / destruction.
MEMTAG_REGISTRY 0x00000014	Registry operations.
MEMTAG_RPC 0x00000015	RPC operations.
MEMTAG_STUFF 0x00000016	Miscellaneous operations.
MEMTAG_FILEBUF	File buffer operations.

<b>Value</b>	<b>Meaning</b>
0x00000017	
MEMTAG_REMOTE 0x00000018	Remote IP address operations.
MEMTAG_EVTCTRL 0x00000019	Event control operations.
MEMTAG_SAFE 0x0000001A	Miscellaneous queuing operations.
MEMTAG_RECORD_UNKNOWN 0x0000001B	Record operations.
MEMTAG_RECORD_FILE 0x0000001C	File-based operations.
MEMTAG_RECORD_DS 0x0000001D	Directory server-based RR operations.
MEMTAG_RECORD_AXFR 0x0000001E	Complete zone transfer operations.
MEMTAG_RECORD_IXFR 0x0000001F	Single Record transfer operations.
MEMTAG_RECORD_DYNUP 0x00000020	RR operations for dynamic update.
MEMTAG_RECORD_ADMIN 0x00000021	RR operations for administration.
MEMTAG_RECORD_AUTO 0x00000022	RR operations for autoconfig.
MEMTAG_RECORD_CACHE 0x00000023	RR operations for cache.
MEMTAG_RECORD_NOEXIST 0x00000024	RR operations for non-existent records.
MEMTAG_RECORD_WINS 0x00000025	RR operations for WINS.
MEMTAG_RECORD_WINSPTR 0x00000026	RR operations for WINS-PTR.
MEMTAG_RECORD_COPY 0x00000027	RR copy operations.
MEMTAG_NODE_UNKNOWN 0x00000028	Node operations for database.
MEMTAG_NODE_FILE 0x00000029	Node operations for file.
MEMTAG_NODE_DS	Node operations for directory server.

Value	Meaning
0x0000002A	
MEMTAG_NODE_AXFR 0x0000002B	Node operations for complete zone transfer.
MEMTAG_NODE_IXFR 0x0000002C	Node operations for single record transfer.
MEMTAG_NODE_DYNUP 0x0000002D	Node operations for dynamic update.
MEMTAG_NODE_ADMIN 0x0000002E	Node operations for administration.
MEMTAG_NODE_AUTO 0x0000002F	Node operations for autoconfig.
MEMTAG_NODE_CACHE 0x00000030	Node operations for cache.
MEMTAG_NODE_NOEXIST 0x00000031	Node operations for non-existent records.
MEMTAG_NODE_WINS 0x00000032	Node operations for WINS.
MEMTAG_NODE_WINSPTR 0x00000033	Node operations for WINS-PTR.
MEMTAG_NODE_COPY 0x00000034	Node operations for copy.

### 2.2.10.2.17 DNSSRV\_TIMEOUT\_STATS

The DNSSRV\_TIMEOUT\_STATS structure has DNS server statistics related to timeout operations on the server. This structure MUST be formatted as follows:

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
Header																															
...																															
SetTotal																															
SetDirect																															
SetFromDereference																															
SetFromChildDelete																															

AlreadyInSystem
Checks
RecentAccess
ActiveRecord
CanNotDelete
Deleted
ArrayBlocksCreated
ArrayBlocksDeleted
DelayedFreesQueued
DelayedFreesQueuedWithFunction
DelayedFreesExecuted
DelayedFreesExecutedWithFunction

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**SetTotal (4 bytes):** The total number of times the server marked a node as being eligible for deletion when it is no longer in use by the cache.

**SetDirect (4 bytes):** The number of times the server marked a node as being eligible for deletion when it is no longer in use by the cache, by directly referencing the node.

**SetFromDereference (4 bytes):** The number of times the server marked a node as being eligible for deletion when it is no longer in use by the cache because the last reference was deleted.

**SetFromChildDelete (4 bytes):** The number of times the server marked a node as being eligible for deletion when it is no longer in use by the cache because the node's last child was deleted.

**AlreadyInSystem (4 bytes):** The number of times the server marked a node as being eligible for deletion when it is no longer in use by the cache when the node was already so marked.

**Checks (4 bytes):** The number of times the server performed any node timeout marking operation.

**RecentAccess (4 bytes):** The number of times the server encountered a cache node that it could not delete because the node had recently been accessed.

**ActiveRecord (4 bytes):** The number of times while performing checks the server encountered a cache node that had records present while checking nodes for deletion.

**CanNotDelete (4 bytes):** The number of times the server encountered a cache node that was marked for deletion that could not be deleted because it had been recently accessed or because it had active records or child nodes.

**Deleted (4 bytes):** The number of times the server successfully deleted a cache node that was marked as eligible for deletion.

**ArrayBlocksCreated (4 bytes):** The number of times the server created a block to hold more references to cache nodes eligible for deletion.

**ArrayBlocksDeleted (4 bytes):** The number of times the server deleted a block to hold references to cache nodes eligible for deletion.

**DelayedFreesQueued (4 bytes):** The number of times the server entered a block of memory into an internal list of memory blocks that may be freed at some time in the future.

**DelayedFreesQueuedWithFunction (4 bytes):** The number of times the server entered a block of memory into an internal list of memory blocks that may be freed at some time in the future where the block must be freed using a function other than the standard memory free function.

**DelayedFreesExecuted (4 bytes):** The number of times the server released a block of memory that had previously been entered into an internal list of memory blocks that may be freed at some time in the future.

**DelayedFreesExecutedWithFunction (4 bytes):** The number of times the server released a block of memory that had previously been entered into an internal list of memory blocks that may be freed at some time in the future, where a function other than the standard memory free function was used for release.

#### 2.2.10.2.18 DNSSRV\_DBASE\_STATS

The DNSSRV\_DBASE\_STATS structure has DNS server statistics related to the database tree. This structure MUST be formatted as follows:

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
Header																															
...																															
NodeMemory																															
NodeInUse																															
NodeUsed																															
NodeReturn																															

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**NodeMemory (4 bytes):** The total size, in bytes, of server memory currently used for nodes.



**NodeInUse (4 bytes):** The number of nodes currently allocated for use in the record database.

**NodeUsed (4 bytes):** The cumulative number of nodes allocated for use in the record database.

**NodeReturn (4 bytes):** The cumulative number of nodes freed from the record database.

### 2.2.10.2.19 DNSSRV\_RECORD\_STATS

The DNSSRV\_RECORD\_STATS structure has DNS server statistics related to record usage. This structure MUST be formatted as follows:

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
Header																															
...																															
InUse																															
Used																															
Return																															
Memory																															
CacheTotal																															
CacheCurrent																															
CacheTimeouts																															
SlowFreeQueued																															
SlowFreeFinished																															

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**InUse (4 bytes):** The number of resource records currently allocated by the server.

**Used (4 bytes):** The cumulative number of resource records allocated by the server.

**Return (4 bytes):** The cumulative number of resource records freed by the server.

**Memory (4 bytes):** The amount of memory, in bytes, currently allocated for resource records by the server.

**CacheTotal (4 bytes):** The cumulative number resource records cached by the server.

**CacheCurrent (4 bytes):** The number of resource records currently cached by the server.

**CacheTimeouts (4 bytes):** The cumulative number of resource records that have been freed from the DNS server's cache.

**SlowFreeQueued (4 bytes):** Some cached record types, such as NS and SOA, are not immediately freed to the pool of allocated records, instead they are placed in a timeout queue and returned after this timeout expires. This is the cumulative count of such slow-free records that have been entered into the timeout queue.

**SlowFreeFinished (4 bytes):** The number of slow frees (see **SlowFreeQueued** above) that have been completed.

### 2.2.10.2.20 DNSSRV\_PACKET\_STATS

The DNSSRV\_PACKET\_STATS structure has DNS server statistics related to packets usage [<74>](#). This structure MUST be formatted as follows:

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		
Header																																	
...																																	
UdpAlloc																																	
UdpFree																																	
UdpNetAllocs																																	
UdpMemory																																	
UdpUsed																																	
UdpReturn																																	
UdpResponseReturn																																	
UdpQueryReturn																																	
UdpInUse																																	
UdpInFreeList																																	
TcpAlloc																																	
TcpRealloc																																	
TcpFree																																	

TcpNetAllocs
TcpMemory
RecursePacketUsed
RecursePacketReturn
PacketsForNsListUsed (optional)
PacketsForNsListReturned (optional)
PacketsForNsListInUse (optional)

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**UdpAlloc (4 bytes):** The cumulative number of UDP packets allocated by the server from system memory.

**UdpFree (4 bytes):** The cumulative number of UDP packets returned by the server to system memory.

**UdpNetAllocs (4 bytes):** The number of currently allocated UDP packets.

**UdpMemory (4 bytes):** The number of bytes of memory used by allocated or taken from the free list for use in query processing.

**UdpUsed (4 bytes):** The cumulative number of UDP packets from the pool of packets used by the server.

**UdpReturn (4 bytes):** The cumulative number of UDP packets freed or returned to the free list by the server.

**UdpResponseReturn (4 bytes):** The cumulative number of UDP packets freed or returned to the free list by the server that were UDP response packets.

**UdpQueryReturn (4 bytes):** The cumulative number of UDP query packets freed or returned to the free list by the server that were UDP query packets.

**UdpInUse (4 bytes):** The number of UDP packets currently in use to process queries.

**UdpInFreeList (4 bytes):** The number of UDP packets currently on the server's free list.

**TcpAlloc (4 bytes):** The cumulative number of TCP buffers allocated by the server from system memory.

**TcpRealloc (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**TcpFree (4 bytes):** The cumulative number of TCP buffers returned by the server to system memory.

**TcpNetAllocs (4 bytes):** The number of allocated TCP buffers currently allocated by the server.

**TcpMemory (4 bytes):** The total system memory, in bytes, used by TCP buffers currently allocated by the server.

**RecursePacketUsed (4 bytes):** The cumulative number of packets used by the server for recursion queries.

**RecursePacketReturn (4 bytes):** The cumulative number of packets that were used for recursive queries and then returned by the server to the pool of packets.

**PacketsForNsListUsed (4 bytes):** The total number of TCP buffers used by the server for name server list query messages. This field SHOULD be present. [<75>](#)A client can tell whether the field is present based on the size of the buffer holding this structure.

**PacketsForNsListReturned (4 bytes):** The total number of TCP buffers that were used for name server lists in query messages, returned by the server to the pool of packets. This field MUST be present if and only if PacketsForNsListUsed is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

**PacketsForNsListInUse (4 bytes):** The number of TCP buffers that are currently being used by the server for name lists in query messages. This field MUST be present if and only if PacketsForNsListUsed is present. A client can tell whether the field is present based on the size of the buffer holding this structure.

### 2.2.10.2.21 DNSSRV\_NBSTAT\_STATS

The DNSSRV\_NBSTAT\_STATS structure has DNS server statistics related to NBSTAT buffers usage. This structure MUST be formatted as follows:

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
Header																															
...																															
NbstatAlloc																															
NbstatFree																															
NbstatNetAllocs																															
NbstatMemory																															
NbstatUsed																															
NbstatReturn																															
NbstatInUse																															
NbstatInFreeList																															

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**NbstatAlloc (4 bytes):** The cumulative number of NetBIOS packet buffers allocated by the server from system memory.

**NbstatFree (4 bytes):** The cumulative number of NetBIOS packet buffers returned by the server to system memory.

**NbstatNetAllocs (4 bytes):** The number of NetBIOS packet buffers currently allocated by the server.

**NbstatMemory (4 bytes):** The total memory used by the NetBIOS packet buffers currently allocated by the server.

**NbstatUsed (4 bytes):** The cumulative number of NetBIOS buffers currently in use by the server either servicing queries or in a free list.

**NbstatReturn (4 bytes):** The cumulative number of NetBIOS buffers freed or returned by the server to a free list.

**NbstatInUse (4 bytes):** The number of NetBIOS buffers currently being used by the server to service queries or being held in a free list.

**NbstatInFreeList (4 bytes):** The number of NetBIOS buffers currently in a free list.

## 2.2.10.2.22 DNSSRV\_PRIVATE\_STATS

The DNSSRV\_PRIVATE\_STATS structure has DNS server statistics related to internal server processing. This structure MUST be formatted as follows:

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
Header																															
...																															
RecordFile																															
RecordFileFree																															
RecordDs																															
RecordDsFree																															
RecordAdmin																															
RecordAdminFree																															
RecordDynUp																															
RecordDynUpFree																															

RecordAxfr
RecordAxfrFree
RecordIxfr
RecordIxfrFree
RecordCopy
RecordCopyFree
RecordCache
RecordCacheFree
UdpSocketPnpDelete
UdpRecvFailure
UdpErrorMessageSize
UdpConnResets
UdpConnResetRetryOverflow
UdpGQCSFailure
UdpGQCSFailureWithContext
UdpGQCSCConnReset
UdpIndicateRecvFailures
UdpRestartRecvOnSockets
TcpConnectAttempt
TcpConnectFailure
TcpConnect
TcpQuery
TcpDisconnect

SecTsigVerifyOldSig
SecTsigVerifyOldFailed
SecBigTimeSkewBypass
ZoneLoadInit
ZoneLoadComplete
ZoneDbaseDelete
ZoneDbaseDelayedDelete

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**RecordFile (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordFileFree (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordDs (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordDsFree (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordAdmin (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordAdminFree (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordDynUp (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordDynUpFree (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordAxfr (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordAxfrFree (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordIxfr (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordIxfrFree (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordCopy (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordCopyFree (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**RecordCache (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

**RecordCacheFree (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**UdpSocketPnpDelete (4 bytes):** The number of UDP sockets that have been closed and had their locally allocated state freed by the server because a UDP error occurred or because the socket was closed in response to an IP address change on the local machine.

**UdpRecvFailure (4 bytes):** The number of times the server failed to receive UDP packet.

**UdpErrorMessageSize (4 bytes):** The number of times the server received an error from UDP socket due to the large size of the receive packet.

**UdpConnResets (4 bytes):** The number of times the server received a connection reset error from UDP.

**UdpConnResetRetryOverflow (4 bytes):** The number of times the server received a connection reset error from UDP and could not clear the error by resubmitting a receive operation.

**UdpGQCSFailure (4 bytes):** The number of times the server received an error from UDP.

**UdpGQCSFailureWithContext (4 bytes):** The number of times the server received an error from UDP where no internal state for the UDP operation was available.

**UdpGQCSConnReset (4 bytes):** The number of times the server received an error from UDP indicating that a remote address was unreachable.

**UdpIndicateRecvFailures (4 bytes):** The number of times the server received a critical error while attempting to perform a UDP receive operation.

**UdpRestartRecvOnSockets (4 bytes):** The number of times the server attempted to restart receive operations on its UDP sockets due to UDP errors.

**TcpConnectAttempt (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

**TcpConnectFailure (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

**TcpConnect (4 bytes):** The number of times the server was able to successfully establish a TCP connection to a remote the server.

**TcpQuery (4 bytes):** The number of times the server sent a recursive query over a TCP connection.

**TcpDisconnect (4 bytes):** The number of times the server disconnected a TCP connection.

**SecTsigVerifyOldSig (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**SecTsigVerifyOldFailed (4 bytes):** Unused. Senders MUST set this value to zero and receivers MUST ignore it.



**SecBigTimeSkewBypass (4 bytes):** The number of times the server received a TKEY that had a time-skew within the allowable range of 1 day.

**ZoneLoadInit (4 bytes):** The number of times the server prepared to load or reload a zone from persistent storage or from a zone transfer.

**ZoneLoadComplete (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

**ZoneDbaseDelete (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

**ZoneDbaseDelayedDelete (4 bytes):** MUST be set to zero when sent and MUST be ignored on receipt.

### 2.2.10.2.23 DNSSRV\_ERROR\_STATS

The DNSSRV\_ERROR\_STATS structure has DNS server statistics related to the different types of errors returned by the server. This structure MUST be formatted as follows:

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
Header																															
...																															
NoError																															
FormError																															
ServFail																															
NxDomain																															
NotImpl																															
Refused																															
YxDomain																															
YxRRSet																															
NxRRSet																															
NotAuth																															
NotZone																															
Max																															

BadSig
BadKey
BadTime
UnknownError

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**NoError (4 bytes):** The number of occurrences where the server returned success (0x00000000) and query was successfully responded to.

**FormError (4 bytes):** The number of occurrences where the server returned error code 0x00000001 due to a malformed query.

**ServFail (4 bytes):** The number of occurrences where the server returned error code 0x00000002 due to a failure in query processing at server.

**NxDomain (4 bytes):** The number of occurrences where the server returned a name error code 0x00000003.

**NotImpl (4 bytes):** The number of occurrences where the server returned error code 0x00000004 due to unimplemented functionality.

**Refused (4 bytes):** The number of occurrences where the server returned error code 0x00000005 due to policy restrictions.

**YxDomain (4 bytes):** The number of occurrences where the server returned error code 0x00000006 due to a domain not being found.

**YxRRSet (4 bytes):** The number of occurrences where the server returned error code 0x00000007 due to the unexpected existence of a resource record.

**NxRRSet (4 bytes):** The number of occurrences where the server returned error code 0x00000008, because the requested resource record did not exist.

**NotAuth (4 bytes):** The number of occurrences where the server returned error code 0x00000009 due to the server not being authoritative for the zone.

**NotZone (4 bytes):** The number of occurrences where the server returned error 0x0000000A (10) due to the requested zone not being found.

**Max (4 bytes):** The number of occurrences where the server returned an error code 0x0000000F (15) which is larger than 4 bits and the server needed to introduce the OPT field in the response packet.

**BadSig (4 bytes):** The number of occurrences where the server returned error 0x00000010 (16) due to a bad signature being present in the query.

**BadKey (4 bytes):** The number of occurrences where the server returned error 0x00000011 (17) due to a bad key being present in the query.

**BadTime (4 bytes):** The number of occurrences where the server returned error 0x00000012 (18) due to a bad time stamp being present in the query.

**UnknownError (4 bytes):** The number of occurrences where the server returned an error code that was caused by any other reason than those listed above.

### 2.2.10.2.24 DNSSRV\_CACHE\_STATS

The DNSSRV\_CACHE\_STATS structure has DNS server statistics related to the server cache. This structure MUST be formatted as follows:

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
Header																															
...																															
CacheExceededLimitChecks																															
SuccessfulFreePasses																															
FailedFreePasses																															
PassesWithNoFrees																															
PassesRequiringAggressiveFree																															

**Header (8 bytes):** A structure of type DNSSRV\_STAT\_HEADER (section [2.2.10.2.1](#)).

**CacheExceededLimitChecks (4 bytes):** Not used. Senders MUST set this value to zero and receivers MUST ignore it.

**SuccessfulFreePasses (4 bytes):** The number of times since the server last started that the server cache was found to exceed the cache size limit, which is 90 percent of the MaxCacheSize (section [3.1.1.1.1](#)), and that an attempt to free nodes resulted in the cache size limit being met. After reaching 0xFFFFFFFF, the value increments to 0x00000000.

**FailedFreePasses (4 bytes):** The number of times since the server last started that the server cache was found to exceed the cache size limit, which is 90 percent of the MaxCacheSize (section [3.1.1.1.1](#)), and that an attempt to free nodes was unsuccessful in meeting the cache size limit. After reaching 0xFFFFFFFF, the value increments to 0x00000000.

**PassesWithNoFrees (4 bytes):** The number of times since the server last started that the server cache was found to exceed the cache size limit, which is 90 percent of the MaxCacheSize (section [3.1.1.1.1](#)), but when the server scanned the cache looking for nodes containing no records or only expired records to free, it found no nodes that could be freed. After reaching 0xFFFFFFFF, the value increments to 0x00000000.

**PassesRequiringAggressiveFree (4 bytes):** The number of times since the server last started that the server cache was found to exceed the cache size limit, which is 90 percent of the MaxCacheSize (section [3.1.1.1.1](#)), and that the server scanned the cache aggressively attempting to free even nodes that contain unexpired records. An aggressive scan frees, in order, nodes containing records that are to expire within the next hour, records that are to expire within the next day, and all records if needed, and halts the freeing process

immediately once the cache size limit is reached. After reaching 0xFFFFFFFF, the value increments to 0x00000000.

## 2.2.11 Key Structures

### 2.2.11.1 Enumerations and Constants

#### 2.2.11.1.1 Cryptographic Algorithm Name

In the context of this protocol, a Cryptographic Algorithm Name is a null-terminated Unicode string representing a digital signature algorithm. The following values are acceptable.

Name	Description
"RSA"	The RSA signature algorithm specified in <a href="#">[FIPS186]</a> section 5. Key pairs for use with this algorithm are represented in the RSA Key Pair format specified in section <a href="#">2.2.11.2.1</a> .
"ECDSA_P256"	The ECDSA signature algorithm specified in <a href="#">[FIPS186]</a> section 6, computed over the curve P-256 specified in <a href="#">[FIPS186]</a> Appendix D.1.2.3. Key pairs for use with this algorithm are represented in the ECDSA Key Pair format specified in section <a href="#">2.2.11.2.2</a> .
"ECDSA_P384"	The ECDSA signature algorithm specified in <a href="#">[FIPS186]</a> section 6, computed over the curve P-384 specified in <a href="#">[FIPS186]</a> Appendix D.1.2.4. Key pairs for use with this algorithm are represented in the ECDSA Key Pair format specified in section <a href="#">2.2.11.2.3</a> .

### 2.2.11.2 Structures

#### 2.2.11.2.1 RSA Key Pair

The RSA Key Pair structure is used to store a key pair (a public key and corresponding private key) for use with the RSA digital signature algorithm.

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
Reserved1																															
ModulusBitLength																															
LengthOfPublicExponent																															
LengthOfModulus																															
LengthOfPrime1																															
LengthOfPrime2																															
PublicExponent (variable)																															
...																															

Modulus (variable)
...
Prime1 (variable)
...
Prime2 (variable)
...

**Reserved1 (4 bytes):** MUST be 0x32415352.

**ModulusBitLength (4 bytes):** This MUST be a 32-bit unsigned integer in little-endian format. It MUST be the length of the RSA modulus, in bits, equal to 8 times the quantity referred to as **k** in [\[RFC3447\]](#) section 2.

**LengthOfPublicExponent (4 bytes):** This MUST be a 32-bit unsigned integer in little-endian format. It MUST be the length of the **PublicExponent** field, in bytes.

**LengthOfModulus (4 bytes):** This MUST be a 32-bit unsigned integer in little-endian format. It MUST be the length of the **Modulus** field, in bytes.

**LengthOfPrime1 (4 bytes):** This MUST be a 32-bit unsigned integer in little-endian format. It MUST be the length of the **Prime1** field, in bytes.

**LengthOfPrime2 (4 bytes):** This MUST be a 32-bit unsigned integer in little-endian format. It MUST be the length of the **Prime2** field, in bytes.

**PublicExponent (variable):** This MUST be the public exponent of the key pair, referred to as **e** in [\[RFC3447\]](#) section 2. It MUST be encoded in big-endian format.

**Modulus (variable):** This MUST be the RSA modulus, referred to as **n** in [\[RFC3447\]](#) section 2. It MUST be equal to **Prime1** \* **Prime2**. It MUST be encoded in big-endian format.

**Prime1 (variable):** This MUST be the first prime factor of the RSA modulus, referred to as **p** in [\[RFC3447\]](#) section 2. It MUST be encoded in big-endian format.

**Prime2 (variable):** This MUST be the second prime factor of the RSA modulus, referred to as **q** in [\[RFC3447\]](#) section 2. It MUST be encoded in big-endian format.

### 2.2.11.2.2 ECDSA\_P256 Key Pair

The ECDSA\_P256 Key Pair structure is used to store an ECDSA\_P256 key pair (a public key and corresponding private key) for use with the ECDSA digital signature algorithm.

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
Reserved1																															



...
...

**Reserved1 (4 bytes):** MUST be 0x324B4345.

**Reserved2 (4 bytes):** MUST be 0x00000020.

**X (32 bytes):** This MUST be the x-coordinate of the ECDSA public key. It MUST be encoded in big-endian format.

**Y (32 bytes):** This MUST be the y-coordinate of the ECDSA public key. It MUST be encoded in big-endian format.

**d (32 bytes):** This MUST be the ECDSA private key. It MUST be encoded in big-endian format.

### 2.2.11.2.3 ECDSA\_P384 Key Pair

The ECDSA\_P256 Key Pair structure is used to store an ECDSA\_P384 key pair (a public key and corresponding private key) for use with the ECDSA digital signature algorithm.

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
Reserved1																															
Reserved2																															
X																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
...																															
(X cont'd for 4 rows)																															
Y																															
...																															

...
...
...
...
...
...
...
(Y cont'd for 4 rows)
d
...
...
...
...
...
...
...
...
(d cont'd for 4 rows)

**Reserved1 (4 bytes):** MUST be 0x324B4345.

**Reserved2 (4 bytes):** MUST be 0x00000030.

**X (48 bytes):** This MUST be the x-coordinate of the ECDSA public key. It MUST be encoded in big-endian format.

**Y (48 bytes):** This MUST be the y-coordinate of the ECDSA public key. It MUST be encoded in big-endian format.

**d (48 bytes):** This MUST be the ECDSA private key. It MUST be encoded in big-endian format.

#### 2.2.11.2.4 Protection Key Identifier

The Protection Key Identifier structure is used to store metadata about the key used to cryptographically wrap the DNSSEC key as part of the Exported Key Pair structure specified in section [2.2.11.2.6](#).



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
Reserved1																															
Reserved2																															
Reserved3																															
L0KeyID																															
L1KeyID																															
L2KeyID																															
RootKeyID																															
...																															
...																															
...																															
Reserved4																															
DomainNameLength																															
ForestNameLength																															
DNSDomainName (variable)																															
...																															
DNSForestName (variable)																															
...																															

**Reserved1 (4 bytes):** MUST be 0x00000001.

**Reserved2 (4 bytes):** MUST be 0x4B53444B.

**Reserved3 (4 bytes):** MUST be 0x00000000.

**L0KeyID (4 bytes):** An L0 index, as defined in [\[MS-GKDI\]](#) section 3.1.4.1.

**L1KeyID (4 bytes):** An L1 index, as defined in [\[MS-GKDI\]](#) section 3.1.4.1.

**L2KeyID (4 bytes):** An L2 index, as defined in [\[MS-GKDI\]](#) section 3.1.4.1.

**RootKeyID (16 bytes):** A root key identifier, as defined in [\[MS-GKDI\]](#) section 3.1.4.1.

**Reserved4 (4 bytes):** MUST be 0x00000000.

**DomainNameLength (4 bytes):** A 32-bit unsigned integer, encoded in little-endian format. It MUST be the length, in bytes, of the **DNSDomainName** field.

**ForestNameLength (4 bytes):** A 32-bit unsigned integer, encoded in little-endian format. It MUST be the length, in bytes, of the **DNSForestName** field.

**DNSDomainName (variable):** A null-terminated Unicode string containing the DNS-style name of the Active Directory domain in which this identifier was created.

**DNSForestName (variable):** A null-terminated Unicode string containing the DNS-style name of the Active Directory forest in which this identifier was created.

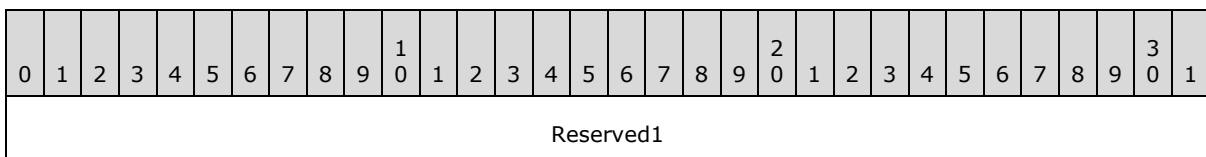
### 2.2.11.2.5 Protection Key Attributes

The Protection Key Attributes structure is used to store some metadata about how a key pair is cryptographically wrapped as part of the Exported Key Pair structure (section [2.2.11.2.6](#)). It is a structure in ASN.1 format, encoded using DER encoding as specified in [\[X690\]](#). The ASN.1 definition for this structure is as follows.

```
microsoft OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) dod(6) internet(1)
private(4) enterprise(1) 311 }
msKeyProtection OBJECT IDENTIFIER := { microsoft 74 }
protectionInfo OBJECT IDENTIFIER ::= { msKeyProtection 1 }
sidProtected OBJECT IDENTIFIER ::= { protectionInfo 1 }
sidName UTF8 STRING ::= "SID"
ProtectionKeyAttribute ::= SEQUENCE {
    protectionInfo OBJECT IDENTIFIER,
    SEQUENCE SIZE (1) {
        sidProtected OBJECT IDENTIFIER,
        SEQUENCE SIZE (1) {
            SEQUENCE SIZE (1) {
                SEQUENCE SIZE (1) {
                    sidName UTF8 STRING,
                    sidString UTF8 STRING
                }
            }
        }
    }
}
```

### 2.2.11.2.6 Exported Key Pair

The Exported Key Pair structure is used to wrap a ZSK for secure storage in the Active Directory database.



Reserved2
LengthOfAlgorithmName
LengthOfProtectedKeyBlob
LengthOfKeyName
AlgorithmName (variable)
...
ProtectedKeyBlob (variable)
...
KeyName (variable)
...

**Reserved1 (4 bytes):** MUST be 0x00000014.

**Reserved2 (4 bytes):** MUST be 0x4B545250.

**LengthOfAlgorithmName (4 bytes):** A 32-bit unsigned integer, encoded using little-endian format. This MUST be equal to the length of the **AlgorithmName** field, in bytes.

**LengthOfProtectedKeyBlob (4 bytes):** A 32-bit unsigned integer, encoded using little-endian format. This MUST be equal to the length of the **ProtectedKeyBlob** field, in bytes.

**LengthOfKeyName (4 bytes):** A 32-bit unsigned integer, encoded using little-endian format. This MUST be equal to the length of the **KeyName** field, in bytes.

**AlgorithmName (variable):** The Cryptographic Algorithm Name, as specified in section [2.2.11.1.1](#), of the digital signature algorithm that the key encapsulated in the **ProtectedKeyBlob** field is intended for use with.

**ProtectedKeyBlob (variable):** A BLOB containing the encrypted key pair, constructed as specified in section [3.1](#).

**KeyName (variable):** A null-terminated Unicode string containing implementation-dependent data.

## 2.2.12 Zone Statistics Messages

### 2.2.12.1 Enumerations and Constants

Constant/Value	Description
MAX_RECORD_TYPES 0x0000001D	Specifies the maximum record types supported.

Constant/Value	Description
MAX_ZONE_TRANSFER_TYPES 0x00000002	Specifies the maximum zone transfer types supported (currently AXFR and IXFR).
DNS_RPC_ZONE_STATS_GET 0x00000000	Enumerates zone statistics.
DNS_RPC_ZONE_STATS_CLEAR 0x00000001	Enumerates zone statistics and resets the zone statistics to zero.

### 2.2.12.1.1 DNS\_ZONE\_STATS\_TYPE

This is the enumerator for all zone statistics types. [<76>](#)

Enumerator Value	Description
ZONE_STATS_TYPE_RECORD_A = 0	A record type
ZONE_STATS_TYPE_RECORD_AAAA	AAAA record type
ZONE_STATS_TYPE_RECORD_PTR	PTR record type
ZONE_STATS_TYPE_RECORD_CNAME	CNAME record type
ZONE_STATS_TYPE_RECORD_MX	MX record type
ZONE_STATS_TYPE_RECORD_AFSDDB	AFSDDB record type
ZONE_STATS_TYPE_RECORD_ATMA	ATMA record type
ZONE_STATS_TYPE_RECORD_DHCID	DHCID record type
ZONE_STATS_TYPE_RECORD_DNAME	DNAME record type
ZONE_STATS_TYPE_RECORD_HINFO	HINFO record type
ZONE_STATS_TYPE_RECORD_ISDN	ISDN record type
ZONE_STATS_TYPE_RECORD_MG	MG record type
ZONE_STATS_TYPE_RECORD_MB	MB record type
ZONE_STATS_TYPE_RECORD_MINFO	MINFO record type
ZONE_STATS_TYPE_RECORD_NAPTR	NAPTR record type
ZONE_STATS_TYPE_RECORD_NXT	NXT record type
ZONE_STATS_TYPE_RECORD_KEY	KEY record type
ZONE_STATS_TYPE_RECORD_MR	MR record type
ZONE_STATS_TYPE_RECORD_RP	RP record type
ZONE_STATS_TYPE_RECORD_RT	RT record type
ZONE_STATS_TYPE_RECORD_SRV	SRV record type

Enumerator Value	Description
ZONE_STATS_TYPE_RECORD_SIG	SIG record type
ZONE_STATS_TYPE_RECORD_TEXT	TXT record type
ZONE_STATS_TYPE_RECORD_WKS	WKS record type
ZONE_STATS_TYPE_RECORD_X25	X25 record type
ZONE_STATS_TYPE_RECORD_DNSKEY	DNSKEY record type
ZONE_STATS_TYPE_RECORD_DS	DS record type
ZONE_STATS_TYPE_RECORD_NS	NS record type
ZONE_STATS_TYPE_RECORD_SOA	SOA record type
ZONE_STATS_TYPE_RECORD_ALL	All record types
ZONE_STATS_TYPE_RECORD_OTHERS	Resource records not specified in this list
ZONE_STATS_TYPE_TRANSFER_AXFR	Axft transfer type
ZONE_STATS_TYPE_TRANSFER_IXFR	Ixfr transfer type
ZONE_STATS_TYPE_UPDATE	Zone Update
MAX_ZONE_STATS_TYPES	Specifies the maximum zone statistics types supported

## 2.2.12.2 Structures

### 2.2.12.2.1 DNSSRV\_ZONE\_TIME\_STATS

The `DNSSRV_ZONE_TIME_STATS` structure [<77>](#) contains zone time statistics information.

```
typedef struct _DnsZoneTimeStats {
    DNS_SYSTEMTIME StatsCollectionStartTime;
} DNSSRV_ZONE_TIME_STATS,
*PDNSSRV_ZONE_TIME_STATS;
```

**StatsCollectionStartTime:** The time when zone statistics collection started.

### 2.2.12.2.2 DNSSRV\_ZONE\_QUERY\_STATS

The `DNSSRV_ZONE_QUERY_STATS` structure [<78>](#) contains per-zone per-record-type statistics.

```
typedef struct _DnsZoneQueryStats {
    DNS_ZONE_STATS_TYPE RecordType;
    ULONG64 QueriesResponded;
    ULONG64 QueriesReceived;
    ULONG64 QueriesFailure;
    ULONG64 QueriesNameError;
} DNSSRV_ZONE_QUERY_STATS,
*PDNSSRV_ZONE_QUERY_STATS;
```

**RecordType:** The type of record for which the query was received. The value should be of type `DNS_ZONE_STATS_TYPE`.

**QueriesResponded:** The total number of queries to which the server responded for a specific zone.

**QueriesReceived:** The total number of queries received by the server for a specific zone.

**QueriesFailure:** The total number of queries for which the server responded with server failure for a specific zone.

**QueriesNameError:** The total number of queries for which the server responded with a name error for a specific zone.

### 2.2.12.2.3 `DNSSRV_ZONE_TRANSFER_STATS`

The `DNSSRV_ZONE_TRANSFER_STATS` structure [<79>](#) contains zone transfer statistics per zone.

```
typedef struct _DnsZoneTransferStats {
    DNS_ZONE_STATS_TYPE TransferType;
    ULONG64 RequestReceived;
    ULONG64 RequestSent;
    ULONG64 ResponseReceived;
    ULONG64 SuccessReceived;
    ULONG64 SuccessSent;
} DNSSRV_ZONE_TRANSFER_STATS,
*PDNSSRV_ZONE_TRANSFER_STATS;
```

**TransferType:** The type of zone transfer request for which the query was received. The value should be of type `DNS_ZONE_STATS_TYPE`.

**RequestReceived:** The total number of zone transfer requests received by the server for a specific zone.

**RequestSent:** The total number of zone transfer requests sent by the server for a specific zone.

**ResponseReceived:** The total number of zone transfer responses received by the server for a specific zone.

**SuccessReceived:** The total number of zone transfer requests received successfully by the receiver for a specific zone.

**SuccessSent:** The total number of zone transfer responses sent by the server for a specific zone.

### 2.2.12.2.4 `DNSSRV_ZONE_UPDATE_STATS`

The `DNSSRV_ZONE_UPDATE_STATS` structure [<80>](#) contains statistics about the **dynamic updates** per zone.

```
typedef struct _DnsZoneUpdateStats {
    DNS_ZONE_STATS_TYPE Type;
```

```

        ULONG64 DynamicUpdateReceived;
        ULONG64 DynamicUpdateRejected;
    } DNSSRV_ZONE_UPDATE_STATS,
    *PDNSSRV_ZONE_UPDATE_STATS;

```

**Type:** The type of zone update request for which statistics are required. The value should be of type [DNS\\_ZONE\\_STATS\\_TYPE \(section 2.2.12.1.1\)](#).

**DynamicUpdateReceived:** The total number of **dynamic updates** accepted on a specified zone.

**DynamicUpdateRejected:** The total number of **dynamic updates** rejected on a specified zone.

### 2.2.12.2.5 DNS\_RPC\_ZONE\_STATS\_V1

The **DNS\_RPC\_ZONE\_STATS\_V1** structure [<81>](#) contains all the statistics about a zone.

```

typedef struct _DnsRpcZoneStats {
    DWORD dwRpcStructureVersion;
    DNSSRV_ZONE_TIME_STATS ZoneTimeStats;
    DNSSRV_ZONE_QUERY_STATS ZoneQueryStats[MAX_RECORD_TYPES];
    DNSSRV_ZONE_TRANSFER_STATS ZoneTransferStats[MAX_ZONE_TRANSFER_TYPES];
    DNSSRV_ZONE_UPDATE_STATS ZoneUpdateStats;
} DNS_RPC_ZONE_STATS_V1,
*PDNS_RPC_ZONE_STATS_V1;

```

**dwRpcStructureVersion:** The DNS management structure version number. This value MUST be set to 0x00000001.

**ZoneTimeStats:** Information about the zone time statistics in DNSSRV\_ZONE\_TIME\_STATS.

**ZoneQueryStats:** An array of DNSSRV\_ZONE\_QUERY\_STATS.

**ZoneTransferStats:** An array of DNSSRV\_ZONE\_TRANSFER\_STATS.

**ZoneUpdateStats:** An array of DNSSRV\_ZONE\_UPDATE\_STATS.

## 2.2.13 Zone Scope Messages

### 2.2.13.1 Enumerations and Constants

### 2.2.13.2 Structures

#### 2.2.13.2.1 DNS\_RPC\_ENUM\_ZONE\_SCOPE\_LIST

The [DNS\\_RPC\\_ENUM\\_ZONE\\_SCOPE\\_LIST](#) structure [<82>](#) contains a list of zone scopes to be enumerated.

```

typedef struct _DnsRpcEnumZoneScopeList {
    DWORD dwRpcStructureVersion;
    DWORD dwZoneScopeCount;
}

```

```

    LPWSTR ZoneScopeArray[1];
} DNS_RPC_ENUM_ZONE_SCOPE_LIST,
*PDNS_RPC_ENUM_ZONE_SCOPE_LIST;

```

**dwRpcStructureVersion:** The DNS management structure version number. This value MUST be set to 0x00000001.

**dwZoneScopeCount:** The number of zone scopes.

**ZoneScopeArray:** An array containing the names of zone scopes.

## 2.2.13.2.2 DNS\_RPC\_ZONE\_SCOPE\_CREATE\_INFO

### 2.2.13.2.2.1 DNS\_RPC\_ZONE\_SCOPE\_CREATE\_INFO\_V1

The **DNS\_RPC\_ZONE\_SCOPE\_CREATE\_INFO\_V1** structure [<83>](#) contains the name of the zone scope.

```

typedef struct _DnsRpcScopeCreate {
    DWORD dwFlags;
    LPWSTR pwszScopeName;
} DNS_RPC_ZONE_SCOPE_CREATE_INFO_V1,
*PDNS_RPC_ZONE_SCOPE_CREATE_INFO_V1;

```

**dwFlags:** Specifies the behavior that the DNS server SHOULD follow when creating the scope. This field is used only when the operation is CreateZoneScope. For any other value than the following, a new empty zone scope is created:

Value	Meaning
DNS_ZONE_LOAD_OVERWRITE_MEMORY 0x00000010	If this value is specified, the DNS server MUST attempt to find and load the zone scope database from persistent storage instead of creating a new empty zone scope. If a pre-existing zone scope database is not found, or if there is a failure in loading the existing database file, the DNS server MUST fail the operation and return an error.

**pwszScopeName:** The name of the zone scope.

## 2.2.13.2.3 DNS\_RPC\_ZONE\_SCOPE\_INFO

### 2.2.13.2.3.1 DNS\_RPC\_ZONE\_SCOPE\_INFO\_V1

The **DNS\_RPC\_ZONE\_SCOPE\_INFO\_V1** structure [<84>](#) contains the details of the zone scope.

```

typedef struct _DnsRpcZoneScopeInfo {
    DWORD dwRpcStructureVersion;
    LPWSTR pwszScopeName;
    LPWSTR pwszDataFile;
} DNS_RPC_ZONE_SCOPE_INFO_V1,

```



\*PDNS\_RPC\_ZONE\_SCOPE\_INFO\_V1;

**dwRpcStructureVersion:** The DNS management structure version number. This value MUST be set to 0x00000001.

**pwszScopeName:** The name of the zone scope.

**pwszDataFile:** The name of the zone scope data file.

## 2.3 Directory Service Schema Elements

The Domain Name Service (DNS) Server Management Protocol accesses the following Directory Service schema classes and attributes listed in the following table. Those listed as "(unused)" are read and stored, but not processed by the protocol. For the syntactic specifications of the following <Class> or <Class> <Attribute> pairs, refer to: [\[MS-ADSC\]](#), [\[MS-ADA1\]](#), [\[MS-ADA2\]](#), and [\[MS-ADA3\]](#). The specifications of the dnsProperty and dnsRecord attributes are described in this section. [<85>](#)

Class	Attribute
container	displayName ntSecurityDescriptor
crossRef	dnsRoot Enabled msDS-NC-Replica-Locations msDS-NC-RO-Replica-Locations nCName ntSecurityDescriptor objectClass systemFlags msDS-Behavior-Version usnChanged (unused)
crossRefContainer	fSMORoleOwner msDS-Behavior-Version
dnsZone	dnsProperty ntSecurityDescriptor objectGUID whenChanged msDNS-IsSigned <a href="#">&lt;86&gt;</a> msDNS-NSEC3OptOut <a href="#">&lt;87&gt;</a> msDNS-SigningKeys <a href="#">&lt;88&gt;</a> msDNS-SignWithNSEC3 <a href="#">&lt;89&gt;</a> msDNS-NSEC3UserSalt <a href="#">&lt;90&gt;</a> msDNS-DNSKEYRecords <a href="#">&lt;91&gt;</a> msDNS-DSRecordSetTTL <a href="#">&lt;92&gt;</a> msDNS-NSEC3Iterations <a href="#">&lt;93&gt;</a>

Class	Attribute
	<a href="#">msDNS-PropagationTime&lt;94&gt;</a> <a href="#">msDNS-NSEC3CurrentSalt&lt;95&gt;</a> <a href="#">msDNS-RFC5011KeyRollovers&lt;96&gt;</a> <a href="#">msDNS-NSEC3HashAlgorithm&lt;97&gt;</a> <a href="#">msDNS-DSRecordAlgorithms&lt;98&gt;</a> <a href="#">msDNS-DNSKEYRecordSetTTL&lt;99&gt;</a> <a href="#">msDNS-MaintainTrustAnchor&lt;100&gt;</a> <a href="#">msDNS-NSEC3RandomSaltLength&lt;101&gt;</a> <a href="#">msDNS-SigningKeyDescriptors&lt;102&gt;</a> <a href="#">msDNS-SignatureInceptionOffset&lt;103&gt;</a> <a href="#">msDNS-ParentHasSecureDelegation&lt;104&gt;</a> <a href="#">msDNS-SecureDelegationPollingPeriod&lt;105&gt;</a>
dnsNode	dnsRecord dnsTombstoned whenChanged usnChanged (unused)
domainDns	instanceType
group	groupType
nTDSDSA	hasMasterNCs msDS-Behavior-Version msDS-HasMasterNCs
rootDse	configurationNamingContext defaultNamingContext dnsHostName dsServiceName namingContexts rootDomainNamingContext schemaNamingContext serverName supportedCapabilities
<a href="#">msDNS-ServerSettings&lt;106&gt;</a>	<a href="#">msDNS-KeymasterZones&lt;107&gt;</a>

## 2.3.1 Object Classes

### 2.3.1.1 msDNS-ServerSettings

An object of this object class MUST be created (if it does not already exist) by any DNS server that becomes the **key master** for a directory service-integrated zone. [<108>](#) The object MUST exist in the directory service as a sibling object of (sharing the same parent object as) the object given by the dsServiceName attribute in the rootDse object on the key master server.

## 2.3.2 Attributes

### 2.3.2.1 dnsProperty

The dnsProperty attribute is used to store zone properties. This attribute MUST be formatted as follows:

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					
DataLength										NameLength										Flag						Version										
Id										Data (variable)																										
...																																				
Name																																				

**DataLength (1 byte):** An unsigned binary integer containing the length, in bytes, of the Data field.

**NameLength (1 byte):** Not Used. The value MUST be ignored and assumed to be 0x00000001.

**Flag (1 byte):** This field is reserved for future use. The value MUST be 0x00000000.

**Version (1 byte):** The version number associated with the property attribute. The value MUST be 0x00000001.

**Id (1 byte):** The property attribute's type. See [Property Id \(section 2.3.2.1.1\)](#).

**Data (variable):** The data associated with an Id. See **Property Id** (section 2.3.2.1.1).

**Name (1 byte):** Not used. The value MUST be of length 1 byte, and MUST be ignored.

#### 2.3.2.1.1 Property Id

The Id specifies the type of data in a dnsProperty's Data field. [<109>](#)

Constant/value	Description
DSPROPERTY_ZONE_TYPE 0x00000001	The zone type. See <a href="#">dwZoneType (section 2.2.5.2.4.1)</a> .
DSPROPERTY_ZONE_ALLOW_UPDATE 0x00000002	Whether dynamic updates are allowed. See <b>fAllowUpdate</b> (section 2.2.5.2.4.1).
DSPROPERTY_ZONE_SECURE_TIME 0x00000008	The time at which the zone became secure. See Time Zone Secured ( <a href="#">3.1.1</a> ).
DSPROPERTY_ZONE_NOREFRESH_INTERVAL 0x00000010	The zone no refresh interval. See <b>dwNoRefreshInterval</b> (section 2.2.5.2.4.1).
DSPROPERTY_ZONE_REFRESH_INTERVAL	The zone refresh interval. See <b>dwRefreshInterval</b>

Constant/value	Description
0x00000020	(section 2.2.5.2.4.1).
DSPROPERTY_ZONE_AGING_STATE 0x00000040	Whether aging is enabled. See <b>fAging</b> (section 2.2.5.2.4.1).
DSPROPERTY_ZONE_SCAVENGING_SERVERS 0x00000011	A list of DNS servers that will perform scavenging. The list is formatted as an <b>IP4 ARRAY</b> (section 2.2.3.2.1). See <b>aipScavengeServers</b> (section 2.2.5.2.4.1).<110>
DSPROPERTY_ZONE_AGING_ENABLED_TIME 0x00000012	The time interval before the next scavenging cycle. See <b>dwAvailForScavengeTime</b> (section 2.2.5.2.4.1).
DSPROPERTY_ZONE_DELETED_FROM_HOSTNAME 0x00000080	The name of the server that deleted the zone. The value is a null-terminated Unicode string. The server SHOULD ignore this value.<111>
DSPROPERTY_ZONE_MASTER_SERVERS 0x00000081	A list of DNS servers that will perform zone transfers. The list is formatted as an <b>IP4 ARRAY</b> (section 2.2.3.2.1). See <b>aipMasters</b> (section 2.2.5.2.4.1).<112>
DSPROPERTY_ZONE_AUTO_NS_SERVERS 0x00000082	A list of servers which MAY autcreate a delegation. The list is formatted as an <b>IP4 ARRAY</b> (section 2.2.3.2.1).<113>
DSPROPERTY_ZONE_DCPROMO_CONVERT 0x00000083	The flag value representing the state of conversion of the zone. See <b>DcPromo Flag</b> (section 2.3.2.1.2).
DSPROPERTY_ZONE_SCAVENGING_SERVERS_DA 0x00000090	A list of DNS servers that will perform scavenging. The list is formatted as a <b>DNS_ADDR_ARRAY</b> (section 2.2.3.2.3). The DNS server MUST read and write the <b>aipScavengeServers</b> (section 2.2.5.2.4.1) setting using property Id DSPROPERTY_ZONE_SCAVENGING_SERVERS and SHOULD also read and write the <b>aipScavengeServers</b> (section 2.2.5.2.4.1) setting using <b>property Id</b> DSPROPERTY_ZONE_SCAVENGING_SERVERS_DA. During read, if the <b>property</b> values are not identical, the DNS server SHOULD use the property value specified by this <b>property Id</b> .<114>
DSPROPERTY_ZONE_MASTER_SERVERS_DA 0x00000091	A list of DNS servers that will perform zone transfers. The list is formatted as a <b>DNS_ADDR_ARRAY</b> (section 2.2.3.2.3). The DNS server MUST read and write this list using <b>property Id</b> DSPROPERTY_ZONE_MASTER_SERVERS and SHOULD also read and write this list using <b>property Id</b> DSPROPERTY_ZONE_MASTER_SERVERS_DA. During read, if the property values are not identical, the DNS server SHOULD<115> use the property value specified by this <b>property Id</b> .
DSPROPERTY_ZONE_AUTO_NS_SERVERS_DA 0x00000092	A list of servers which MAY autcreate a delegation. The list is formatted as a <b>DNS_ADDR_ARRAY</b>

Constant/value	Description
	(section 2.2.3.2.3) The DNS server MUST read and write this list using <b>property Id</b> DSPROPERTY_ZONE_AUTO_NS_SERVERS and SHOULD also read and write this list using <b>property Id</b> DSPROPERTY_ZONE_AUTO_NS_SERVERS_DA. During read, if the property values are not identical, the DNS server SHOULD <a href="#">&lt;116&gt;</a> use the value specified by <b>property Id</b> .
DSPROPERTY_ZONE_NODE_DBFLAGS 0x00000100	See <a href="#">DNS RPC NODE FLAGS (section 2.2.2.1.2)</a> .

### 2.3.2.1.2 DcPromo Flag

The DcPromo flag represents the DcPromo target application directory partition for the zone. If, during zone creation ("ZoneCreate" operation of the R\_Dnssrvoperation (section 3.1.4.1) method call), a zone is placed into the directory partition that represents the default naming context because the correct directory partition was not available at the time, the zone's DcPromo flag is set appropriately to reflect this. While polling the directory server, and if the time elapsed since the last directory server polling operation is more than 15 minutes, the server SHOULD check whether it is in RODC mode (a server is in RODC mode if ForceRODCMode [section 3.1.1.1.1] is TRUE or fReadOnlyDC [section 2.2.4.2.2.3] is TRUE). If the time elapsed is less than 15 minutes, or if the server is in RODC mode, the server MUST NOT perform the following operations. If the server is not in RODC mode and if the DNS server discovers a zone with a nonzero DcPromo flag, the zone and the Zone Access Control List (section 3.1.1) MUST be moved from their current location to the application directory partition specified in the following table if that partition is now available. [<117>](#) [<118>](#)

Constant/value	Description
DCPROMO_CONVERT_NONE 0x00000000	No change to existing zone storage.
DCPROMO_CONVERT_DOMAIN 0x00000001	Zone is to be moved to the DNS domain partition. This is the partition in the Application Directory Partition Table (section 3.1.1) that has the DNS_DP_DOMAIN_DEFAULT bit set in dwDpFlags (section 2.2.5.2.7.2). See DNS_ZONE_CREATE_FOR_DCPROMO (section 2.2.5.2.7.1). If DownlevelDCsInDomain is nonzero, the zone is to be stored in the directory partition that represents the default naming context. See DNS_DP_LEGACY (section 2.2.7.1.1).
DCPROMO_CONVERT_FOREST 0x00000002	Zone is to be moved to the DNS forest partition. This is the partition in the Application Directory Partition Table (section 3.1.1) that has the DNS_DP_FOREST_DEFAULT bit set in dwDpFlags (section 2.2.5.2.7.2). See DNS_ZONE_CREATE_FOR_DCPROMO_FOREST (section 2.2.5.2.7.1).

### 2.3.2.2 dnsRecord

The dnsRecord attribute is used to store DNS resource record definitions. This attribute MUST be formatted as follows:

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
DataLength																Type															
Version								Rank								Flags															
Serial																															
TtlSeconds																															
Reserved																															
TimeStamp																															
Data (variable)																															
...																															

**DataLength (2 bytes):** An unsigned binary integer containing the length, in bytes, of the Data field.

**Type (2 bytes):** The resource record's type. See [DNS\\_RECORD\\_TYPE \(section 2.2.2.1.1\)](#).

**Version (1 byte):** The version number associated with the resource record attribute. The value MUST be 0x05.

**Rank (1 byte):** The least-significant byte of one of the RANK\* flag values. See [dwFlags \(section 2.2.2.2.5\)](#).

**Flags (2 bytes):** Not used. The value MUST be 0x0000.

**Serial (4 bytes):** The serial number of the SOA record of the zone containing this resource record. See [DNS\\_RPC\\_RECORD\\_SOA \(section 2.2.2.4.3\)](#).

**TtlSeconds (4 bytes):** See **dwTtlSeconds** (section 2.2.2.2.5). This field uses big-endian byte order.

**Reserved (4 bytes):** This field is reserved for future use. The value MUST be 0x00000000.

**TimeStamp (4 bytes):** See **dwTimeStamp** (section 2.2.2.2.5).

**Data (variable):** The resource record's data. See [DNS\\_RPC\\_RECORD\\_DATA \(section 2.2.2.2.4\)](#).

### 2.3.2.3 msDNS-IsSigned

This attribute value MUST correspond to the value of the IsSigned zone property (section [3.1.1.2.1](#)).

### 2.3.2.4 msDNS-NSEC3OptOut

This attribute value MUST correspond to the value of the NSEC3OptOut zone property (section [3.1.1.2.1](#)).

### 2.3.2.5 msDNS-SigningKeys

This attribute value MUST contain a collection of exported signing key pairs. Each attribute value MUST be formatted as follows.

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		
GUID																																	
...																																	
...																																	
...																																	
ExportedKey (variable)																																	
...																																	

**GUID (16 bytes):** A GUID value that corresponds to an active, standby, or next key from the [DNS\\_RPC\\_SKD\\_STATE \(section 2.2.6.2.3\)](#) structure for one of the signing key descriptors in the zone's signing key descriptor list.

**ExportedKey (variable):** An exported signing key in [Exported Key Pair](#) format. When this attribute value is written, the DNS server MUST follow the procedure in section [3.1.7.1](#) to export each key. When this attribute value is read, the DNS server MUST follow the procedure in section [3.1.7.2](#) to import each key.

### 2.3.2.6 msDNS-SignWithNSEC3

This attribute value MUST correspond to the value of the SignWithNSEC3 zone property (section [3.1.1.2.1](#)).

### 2.3.2.7 msDNS-NSEC3UserSalt

This attribute value MUST correspond to the value of the NSEC3UserSalt zone property (section [3.1.1.2.3](#)).

### 2.3.2.8 msDNS-DNSKEYRecords

This attribute value MUST contain the DNSKEY records and all RRSIG records covering type DNSKEY for the root of the zone as specified by [\[RFC4035\]](#).

### 2.3.2.9 msDNS-DSRecordSetTTL

This attribute value MUST correspond to the value of the DSRecordSetTTL zone property (section [3.1.1.2.1](#)).

### **2.3.2.10 msDNS-NSEC3Iterations**

This attribute value MUST correspond to the value of the NSEC3Iterations zone property (section [3.1.1.2.1](#)).

### **2.3.2.11 msDNS-PropagationTime**

This attribute value MUST correspond to the value of the PropagationTime zone property (section [3.1.1.2.1](#)).

### **2.3.2.12 msDNS-NSEC3CurrentSalt**

This attribute value MUST correspond to the value of the NSEC3CurrentSalt zone property (section [3.1.1.2.3](#)).

### **2.3.2.13 msDNS-RFC5011KeyRollovers**

This attribute value MUST correspond to the value of the RFC5011KeyRollovers zone property (section [3.1.1.2.1](#)).

### **2.3.2.14 msDNS-NSEC3HashAlgorithm**

This attribute value MUST correspond to the value of the NSEC3HashAlgorithm zone property (section [3.1.1.2.1](#)).

### **2.3.2.15 msDNS-DSRecordAlgorithms**

This attribute value MUST correspond to the value of the DSRecordAlgorithms zone property (section [3.1.1.2.1](#)).

### **2.3.2.16 msDNS-DNSKEYRecordSetTTL**

This attribute value MUST correspond to the value of the DNSKEYRecordSetTTL zone property (section [3.1.1.2.1](#)).

### **2.3.2.17 msDNS-MaintainTrustAnchor**

This attribute value MUST correspond to the value of the MaintainTrustAnchor zone property (section [3.1.1.2.1](#)).

### **2.3.2.18 msDNS-NSEC3RandomSaltLength**

This attribute value MUST correspond to the value of the NSEC3RandomSaltLength zone property (section [3.1.1.2.1](#)).

### **2.3.2.19 msDNS-SigningKeyDescriptors**

This attribute is used to store the zone's signing key descriptor list. Each value of this attribute represents a single signing key descriptor and MUST be formatted as follows.



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																															
fIsKsk																															
Guid																															
...																															
...																															
...																															
pwszKeyStorageProvider (variable)																															
...																															
bSigningAlgorithm																															
dwKeyLength																															
dwInitialRolloverOffset																															
dwDNSKEYSignatureValidityPeriod																															
dwDSSignatureValidityPeriod																															
dwStandardSignatureValidityPeriod																															
dwRolloverType																															
dwRolloverPeriod																															
dwNextRolloverAction																															
ftLastRolloverTime																															
...																															
ftNextRolloverTime																															
...																															
dwState																															

dwCurrentRolloverStatus
dwCurrentRollState
fManualTrigger
dwPreRollEventFired
ftNextKeyGenerationTime
...
RevokedOrSwappedRecordCount
FinalRecordCount
pwszActiveKey (variable)
...
ActiveKeyScope
pwszStandbyKey (variable)
...
StandbyKeyScope
pwszNextKey (variable)
...
NextKeyScope
RevokedOrSwappedDnskeys (variable)
...
FinalDnskeys (variable)
...

**Version (4 bytes):** This value MUST be 0x00000001.

**fIsKsk (4 bytes):** The value of this field MUST correspond to the value from the [DNS\\_RPC\\_SKD \(section 2.2.6.2.1\)](#) structure for this signing key descriptor.

- Guid (16 bytes):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- pwszKeyStorageProvider (variable):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- bSigningAlgorithm (4 bytes):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- dwKeyLength (4 bytes):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- dwInitialRolloverOffset (4 bytes):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- dwDNSKEYSignatureValidityPeriod (4 bytes):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- dwDSSignatureValidityPeriod (4 bytes):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- dwStandardSignatureValidityPeriod (4 bytes):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- dwRolloverType (4 bytes):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- dwRolloverPeriod (4 bytes):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- dwNextRolloverAction (4 bytes):** The value of this field MUST correspond to the value from the **DNS\_RPC\_SKD** (section 2.2.6.2.1) structure for this signing key descriptor.
- ftLastRolloverTime (8 bytes):** This value MUST correspond to the value of **ftLastRolloverTime** from the [DNS\\_RPC\\_SKD\\_STATE \(section 2.2.6.2.3\)](#) structure for this signing descriptor.
- ftNextRolloverTime (8 bytes):** This value MUST correspond to the value of **ftNextRolloverTime** from the **DNS\_RPC\_SKD\_STATE** (section 2.2.6.2.3) structure for this signing descriptor.
- dwState (4 bytes):** This value MUST correspond to the value of **dwState** from the **DNS\_RPC\_SKD\_STATE** (section 2.2.6.2.3) structure for this signing descriptor.
- dwCurrentRolloverStatus (4 bytes):** This value MUST correspond to the value of **dwCurrentRolloverStatus** from the **DNS\_RPC\_SKD\_STATE** (section 2.2.6.2.3) structure for this signing descriptor.
- dwCurrentRollState (4 bytes):** This value corresponds to the next step in a signing key descriptor's rollover process and MUST be set to one of the following values.

Value	Meaning
DNS_SKD_ROLL_STATE_NOT_STARTED 0x00000000	The signing key descriptor is not currently in the process of rolling over keys.

Value	Meaning
DNS_SKD_ROLL_STATE_ZSK_SWAP 0x00000001	The next step for a signing key descriptor whose "fIsKSK" field is 0x00000000 is to move into the "new RRSIGs" phase of the pre-publish key rollover, as described in <a href="#">[RFC4641]</a> section 4.2.1.1.
DNS_SKD_ROLL_STATE_ZSK_FINISH 0x00000002	The next step for a signing key descriptor whose "fIsKSK" field is 0x00000000 is to move into the "DNSKEY removal" phase of the pre-publish key rollover, as described in <a href="#">[RFC4641]</a> section 4.2.1.1.
DNS_SKD_ROLL_STATE_KSK_DS_WAIT 0x00000003	The next step for a signing key descriptor whose "fIsKSK" field is 0x00000001 is to wait for the "DS change" step of the "Key Signing Key Rollovers" process, as described in <a href="#">[RFC4641]</a> section 4.2.2.
DNS_SKD_ROLL_STATE_KSK_REVOKE 0x00000004	The next step for a signing key descriptor whose "fIsKSK" field is 0x00000001 is to revoke a key according to <a href="#">[RFC5011]</a> section 6.3 or 6.5.
DNS_SKD_ROLL_STATE_KSK_FINISH 0x00000005	The next step for a signing key descriptor whose "fIsKSK" field is 0x00000001 is to move into the "DNSKEY removal" phase of the "Key Signing Key Rollovers" process, as described in <a href="#">[RFC4641]</a> section 4.2.2.

**fManualTrigger (4 bytes):** This value MUST be set to 0x00000001 in response to a successful "ZonePerformKeyRollover" operation on a signing key descriptor. When the SKD completes its rollover, this value MUST be set to 0x00000000.

**dwPreRollEventFired (4 bytes):** This value MUST be set to 0x00000001 when 90 percent of the "dwRolloverPeriod" for a signing key descriptor whose "fIsKSK" flag is 0x00000001 has elapsed. It MUST be set to 0x00000002 when 95 percent of this rollover period has elapsed, and it MUST be set to 0x00000003 when there is less than 1 day remaining before such a signing key descriptor begins its key rollover process. Otherwise, this value MUST be 0x00000000.

**ftNextKeyGenerationTime (8 bytes):** This value represents the time at which the most recent value of the "pwszNextKey" field of a signing key descriptor whose "fIsKSK" flag is 0x00000000 was generated.

**RevokedOrSwappedRecordCount (4 bytes):** This value MUST indicate the number of values present in the list of records in the **RevokedOrSwappedDnskeys** field.

**FinalRecordCount (4 bytes):** This value MUST indicate the number of values present in the list of records in the **FinalRecordCount** field.

**pwszActiveKey (variable):** This value MUST correspond to the value of **pwszActiveKey** from the DNS\_RPC\_SKD\_STATE (section 2.2.6.2.3) structure for this signing descriptor.

**ActiveKeyScope (4 bytes):** This value corresponds to the signing scope of **pwszActiveKey** and MUST be set to one of the following values.

Value	Meaning
DNS_SIGN_SCOPE_DEFAULT	The key is used for its default purpose: If the signing key

Value	Meaning
0x00000000	descriptor's "fIsKSK" field is set to 0x00000001, the key is used to sign only DNSKEY records in the zone. If the signing key descriptor's "fIsKSK" field is set to 0x00000000, the key is used to sign all records in the zone.
DNS_SIGN_SCOPE_DNSKEY_ONLY 0x00000001	The key is used to sign only DNSKEY records in the zone.
DNS_SIGN_SCOPE_ALL_RECORDS 0x00000002	The key is used to sign all records in the zone.
DNS_SIGN_SCOPE_ADD_ONLY 0x00000003	The key is published as a DNSKEY in the zone, but it is not used to sign any records.
DNS_SIGN_SCOPE_DO_NOT_PUBLISH 0x00000004	The key is not published to the zone and is not used to sign any records.
DNS_SIGN_SCOPE_REVOKED 0x00000005	The key is published as a DNSKEY in the zone with its "Revoked" bit ( <a href="#">RFC5011</a> section 2.1) set. It is used to sign DNSKEY records.

**pwszStandbyKey (variable):** This value MUST correspond to the value of **pwszStandbyKey** from the DNS\_RPC\_SKD\_STATE (section 2.2.6.2.3) structure for this signing descriptor.

**StandbyKeyScope (4 bytes):** This value corresponds to the signing scope of **pwszActiveKey** and MUST be set to one of the values previously described for "ActiveKeyScope".

**pwszNextKey (variable):** This value MUST correspond to the value of **pwszNextKey** from the DNS\_RPC\_SKD\_STATE (section 2.2.6.2.3) structure for this signing descriptor.

**NextKeyScope (4 bytes):** This value corresponds to the signing scope of **pwszNextKey** and MUST be set to one of the values previously described for "ActiveKeyScope".

**RevokedOrSwappedDnskeys (variable):** This field MUST contain a variable number of DNS records in the same format as the dnsRecord attribute (section [2.3.2.2](#)). The number of records in this list is specified by **RevokedOrSwappedRecordCount**. If **RevokedOrSwappedRecordCount** is zero, the length of this field must be 0 bytes. The DNS records in this field correspond to a precomputed list of DNSKEY and associated RRSIG records that are published to the zone as the SKD transitions into DNS\_SKD\_STATUS\_ZSK\_WAITING\_FOR\_MAXZONE\_TTL status if the signing key descriptor's "fIsKSK" field is set to 0x00000000, and as the SKD transitions into DKS\_SKD\_STATUS\_KSK\_WAITING\_FOR\_5011\_REMOVE\_HOLD\_DOWN if the signing key descriptor's "fIsKSK" field is set to 0x00000001.

**FinalDnskeys (variable):** This field MUST contain a number of DNS records in the same format as the dnsRecord attribute (section [2.3.2.2](#)). The number of records in this list is specified by **FinalRecordCount**. If **FinalRecordCount** is zero, the length of this field must be 0 bytes. The DNS records in this field correspond to a precomputed list of DNSKEY and associated RRSIG records that are published to the zone as the signing key descriptor's key rollover process concludes.

### **2.3.2.20 msDNS-SignatureInceptionOffset**

This attribute value MUST correspond to the value of the SignatureInceptionOffset zone property (section [3.1.1.2.1](#)).

### **2.3.2.21 msDNS-ParentHasSecureDelegation**

This attribute value MUST correspond to the value of the ParentHasSecureDelegation zone property (section [3.1.1.2.1](#)).

### **2.3.2.22 msDNS-SecureDelegationPollingPeriod**

This attribute value MUST correspond to the value of the SecureDelegationPollingPeriod zone property (section [3.1.1.2.1](#)).

### **2.3.2.23 msDNS-KeymasterZones**

This attribute MUST contain a list of distinguished names for all of the directory service-integrated zones for which the DNS server is the key master.

## 3 Protocol Details

No additional timers or other state is required on the client side of this protocol. Calls made by the higher-layer protocol or application are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application.

The following sections specify details of the DNS DnsServer Remote Protocol, including abstract data models, interface method syntax, and message processing rules.

### 3.1 DnsServer Server Details

The DNS Server Management Protocol is stateless; that is, each message in the protocol is independent. Furthermore, the running state of the zone on the server does not affect this protocol. The type of zone (primary, secondary, stub, and so on) can determine which method calls can be executed successfully using that zone. Restrictions on zone type are specified in the descriptions of the method calls.

Certain actions taken by the server require an underlying protocol. For example, the **EnlistDirectoryPartition** operation of the R\_Dnssrvoperation (section [3.1.4.1](#)) method call can cause the server to communicate with the Active Directory server with a separate protocol. Implementations encountering error events or error code returns in the execution of these underlying protocols can, in certain cases, generate an error return for the DNS protocol. Where this is the case, the description of the action undertaken will specify that an error code can be returned.

#### 3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

**Global Server State:** The global state of the server set to one of the following values:

- Loading: the DNS server is loading configuration and zone data.
- Running: the DNS server has loaded all data and is serving queries.
- Stopping: the DNS server is shutting down.

**DNS Server Configuration:** Configuration information for the server, in persistent storage, in the form of (name, value) pairs. The list of metadata information can be found in section [3.1.1.1](#).

**DNS Server AD Connection:** An ADCONNECTION\_HANDLE as defined in [\[MS-DTYP\]](#) section 2.2.2. This element is used every time the DNS server needs to communicate to the directory server.

**DNS Server Configuration Access Control List:** An access control list that specifies what client identities have read and write permissions on the DNS Server Configuration. If the DNS server is **directory server** integrated, the **access control list** is stored in the local directory server. [<119>](#) Otherwise, this access control list is not stored in persistent storage and is dynamically created at server start time. [<120>](#)

**DNS Zone Table:** Configuration information for DNS zones, in persistent and in-memory storage. The Zone Table can be stored persistently either in a file or in the local directory

server, but not both simultaneously. The Zone Table also resides in-memory, behaving as a cache containing the current working copy of both zones loaded from persistent storage, as well as zones retrieved from remote DNS servers. A query to a zone MUST be responded to using only the data in the corresponding in-memory zone. A modification to any elements of a zone, such as a DNS update of records [RFC2136], MUST be reflected immediately in the corresponding in-memory zone and MUST be immediately transactionally committed to DS-integrated storage afterwards (see "WriteDirtyZones" in section 3.1.4.1). Each update to the in-memory zone must be atomic. If an update to the in-memory zone does not succeed, the in-memory zone MUST be restored to its previous state before the update. Queries to a zone during an in-memory zone modification process must be responded to using the post-update zone state. For zones stored in the local directory server, this table may include zones that are in the process of being deleted (see "DeleteZoneFromDs" in section 3.1.4.1).

The in-memory DNS Zone Table is used for queries and modifications for all zone operations (see section 3.1.4). The in-memory Zone Table is populated during server initialization per-zone from either the local persistent storage or DS-integrated Zone Table. When modifications are made to a zone, such as after a DNS update or zone transfer, the changes are reflected first in the in-memory Zone Table, which is then copied per zone, for all zone types except DNS\_ZONE\_TYPE\_CACHE (section 2.2.5.1.1), to either the local persistent storage or immediately to the DS-integrated Zone Table. In-memory data retrieved from remote DNS servers are not copied to local persistent or DS-integrated storage, except for secondary zones with an fDsIntegrated value of FALSE (section 2.2.5.2.6), which MUST eventually be copied to local persistent storage. For DS-integrated zones, the server polls the directory server using the LDAP protocol every DsPollingInterval (section 3.1.1.1) to copy DS-integrated zones to the respective in-memory zones. If an error occurs during polling, the DNS server MUST NOT mark the zone as shutdown (see section 2.2.5.2.2) and MUST attempt to poll the zone again after DsPollingInterval (section 3.1.1.1) seconds have elapsed. Zones of type DNS\_ZONE\_TYPE\_CACHE (section 2.2.5.1.1) are never written to persistent storage. All contents of a zone of type DNS\_ZONE\_TYPE\_CACHE MUST be discarded when the DNS server process is terminated or when the "ClearCache" operation (section 3.1.4.1) is executed.

The local persistent storage DNS Zone Table is copied to the in-memory DNS Zone Table on server initialization and is copied from the in-memory Zone Table after a modification to the in-memory Zone Table is complete.

The directory server-integrated DNS Zone Table is copied to the in-memory DNS Zone Table on server initialization and is copied from the in-memory Zone Table immediately after a modification to the in-memory Zone Table is complete. When changes occur in the DS-integrated Zone Table, the changes are not reflected in the in-memory Zone Table until the DNS server polls the directory server to update the in-memory Zone Table with the modified DS-integrated Zone Table. If changes happened simultaneously to the in-memory Zone Table and the DS-integrated Zone Table, then the post-modified DS-integrated Zone Table is copied to and overwrites the in-memory Zone Table.

For each zone, the DNS Zone Table consists of the following:

**Zone Name:** The name of the zone.

**Zone Status:** Operational state information pertaining to the zone, such as the information in DNS\_RPC\_ZONE\_FLAGS (section 2.2.5.2.2).

**Metadata:** Configuration information for the zone, in the form of (name, value) pairs. The list of metadata information can be found in section 3.1.1.2.

**Zone GUID:** A **GUID** associated with the zone. The default value of this property MUST be NULL. If this zone is stored in the directory server, then this value is initialized from the



**objectGUID** attribute of the associated "dnsZone" object in the DS-integrated DNS Zone Table, which is generated as specified in [\[MS-ADTS\]](#), section [3.1.1.1.2](#), when the zone is first added to the directory server. When the server polls the directory server and discovers that a zone has been deleted from the DS-integrated DNS Zone Table, it uses the **objectGUID** attribute of that "dnsZone" object to identify the corresponding zone in the in-memory copy of the DNS Zone Table.

**DNS Nodes and Records:** The collection of DNS records for each DNS zone. Each distinct DNS owner name (see [\[RFC1035\]](#)) present in the zone is represented by a node. Each node in the zone has a collection of DNS records that pertain to the DNS owner name of the node. Each DNS record also has metadata which is specified in section [3.1.1.2](#). The collection of DNS nodes and records for each zone MUST contain a node representing the name of the zone itself, which MUST contain an SOA record that stores the fields present in the DNS\_RPC\_RECORD\_SOA structure (section [2.2.2.2.4.3](#)). The zone serial number is stored in the dwSerialNo field of this SOA record. DNS records may be accessed or updated by the local directory server through directory server replication and by remote DNS servers through the DNS protocol [\[RFC1035\]](#) and DNS update [\[RFC2136\]](#).

**DNS Node Tombstone State:** Each node in a zone MAY have a [Boolean](#) value indicating if this DNS node is a tombstone. [<121>](#)

**Zone Access Control List:** An access control list that specifies what client identities have permissions on this DNS Zone. If the zone is stored in the directory server, the access control list is stored in the **ntSecurityDescriptor** attribute of the "dnsZone" object (section [2.3](#)) and can be modified using standard LDAP modify operations (see [\[MS-ADTS\]](#) section [3.1.1.5.3](#)). If the zone is not stored in the directory server, the zone does not have an access control list associated with it and instead the **DNS Server Configuration Access Control List** is used as the access control list for the zone. [<122>](#)

**Last Transferred Zone Serial Number:** The zone serial number that was last sent in a complete [\[RFC1035\]](#) or incremental [\[RFC1995\]](#) zone transfer to a remote DNS server.

**Time of Last SOA Check:** For a secondary or stub zone, the time at which the primary zone was last contacted (whether successfully or unsuccessfully) to compare zone serial numbers. The value is expressed as the number of seconds since the system booted, in unsigned 32-bit integer format. The default value of this property MUST be zero.

**Time Zone Secured:** The time at which the zone's AllowUpdate property (section [3.1.1.2.1](#)) was last changed from any value to ZONE\_UPDATE\_SECURE (section [2.2.6.1.1](#)). Changes of the zone's AllowUpdate property to any other value MUST NOT cause a change to Time Zone Secured., The value is expressed as the number of seconds since 12:00 A.M. January 1, 1601 Coordinated Universal Time (UTC) in unsigned 64-bit integer format. The default value of this property MUST be zero. If this zone is stored in the local directory server, then this value is initialized from and written to the "whenChanged" attribute of the "dnsZone".

**Dirty Flag:** A [Boolean](#) variable present only in the in-memory copy of a zone. Set to true only if the zone has been modified in memory and there is a copy of the zone in persistent storage to which the in-memory modifications have not been committed.

**Signing Key Descriptor List:** A list of zero or more signing key descriptors associated with the zone, used to inform the online signing process. Each entry in this list consists of the fields in the [DNS\\_RPC\\_SKD](#) (see section [2.2.6.2.1](#)) and [DNS\\_RPC\\_SKD\\_STATE](#) (see section [2.2.6.2.3](#)) structures. [<123>](#)

**Rollover Queue:** An ordered list of zero or more signing key descriptors in the zone ready for key rollover. A signing key descriptor whose **dwRolloverPeriod** is not 0xFFFFFFFF is automatically added to this queue when **dwRolloverPeriod** seconds have elapsed since the signing key descriptor's last successful key rollover, or since the zone was initially online-signed. A signing key descriptor may be added to this queue prior to this time by invoking the "PerformZoneKeyRollover" operation of the R\_DnsSrvOperation (Opnum 0) method (section [3.1.4.1](#)). The DNS server SHOULD perform key rollover on each item in this queue, one at a time, according to the process specified in [\[RFC4641\]](#) and [\[RFC5011\].<124>](#)

**Application Directory Partition Table:** The DNS server SHOULD [<125>](#) support the concept of application directory partitions. Each entry in the Application Directory Partition Table, stored in memory, consists of the fields in DNS\_RPC\_DP\_INFO (section [2.2.7.2.1](#)). This table is populated during DNS server initialization through queries to the local directory server using the LDAP protocol. To populate this table, the DNS server MUST use LDAP queries to enumerate all objects under "CN=Partitions, CN=Configuration, <Forest DN>" of object class "crossRef" and for each object's read attribute values as specified by the definitions of the fields of DNS\_RPC\_DP\_INFO (section [2.2.7.2.1](#)). Certain fields do not correspond directly to data stored in local directory server (see section [2.2.7.2.1](#)). The Application Directory Partition Table is kept up-to-date as partitions are created or deleted on the local server, by writing changes to the local directory server and immediately polling to refresh the Application Directory Partition Table. The DNS server must poll the directory server at an interval specified by DsPollingInterval (see section [3.1.1.1](#)) to reflect changes made by remote directory servers (see section [3.1.4.1](#)). If the default DNS **Domain Partition** or the default DNS Forest Partition is not present when polling, the server MUST attempt to create and enlist in these partitions as part of the polling process.

**Application Directory Partition Access Control List:** In addition to the fields in DNS\_RPC\_DP\_INFO [2.2.7.2.1](#)) each entry in the Application Directory Partition Table MUST also contain an access control list stored persistently in the directory server in the **ntSecurityDescriptor** attribute of the "crossRef" object associated with the Application Directory Partition. [<126>](#)

**Remote Server Table:** An in-memory state table of EDNS [\[RFC2671\]](#) support statuses of remote DNS servers that the local server has previously contacted. The status of a remote server is cached for the interval specified by EDnsCacheTimeout (section [3.1.1.1.1](#)). The state of a remote server can be set to one of the following values:

- Unknown: the remote server's EDNS [\[RFC2671\]](#) support is indeterminate; initial state.
- Not supported: the remote server does not support EDNS [\[RFC2671\]](#).
- Ok: the remote server supports EDNS [\[RFC2671\]](#).
- Maybe not supported: the remote server has not responded, and an EDNS [\[RFC2671\]](#) support state cannot yet be determined.

**Statistics:** An in-memory structure whose elements correspond directly with the objects specified in sections [2.2.10.2.4](#) through [2.2.10.2.24](#), omitting the "Header" field of each. These values are made available to clients of the DNS Server Management Protocol by the DNS Server when processing R\_DnsSrvComplexOperation2 (section [3.1.4.8](#)) method calls with operation type "Statistics".

**DNS Server Credentials:** The credentials that the DNS server process will be invoked as. These credentials MUST be used for all file, registry, and directory service LDAP operations where user credentials are not available. [<127>](#)

**DownlevelDCsInDomain:** The count of downlevel domain controllers in the domain in an unsigned 32-bit integer format. The default value MUST be zero. <128>

**Domain Naming Master Identity:** Host name of the Domain Naming Master **FSMO role** represented as a string (wchar\*). Whenever the DNS server makes any changes to crossRef objects, it MUST establish an LDAP connection to the Host whose name is stored here, and modify its version of the object accordingly.

**Local security groups:** A list of group identities with accompanying membership information. When permissions are set for an object, a local security group identity can be used to set permissions for all members of that group.

### 3.1.1.1 DNS Server Configuration Information

The list of names that are used in (name, value) pairs in DNS Server Configuration information is given below.

#### 3.1.1.1.1 DNS Server Integer Properties

The following properties are 32-bit integers. The term "Boolean" means that a value of 0x00000000 indicates that the stated property is false, and any nonzero value indicates that the stated property is true. All properties are writable unless stated otherwise. The type ID for these properties is DNS\_SRV\_TYPEID\_NAME\_AND\_PARAM, listed in section 2.2.1.1.1. Property values on reset or load SHOULD be verified to be within the property's allowable range, except when the value is zero and the zero value is allowed. If the value is outside the range, or if the value is zero and the zero value is not allowed, the server SHOULD return an error. <129>

**AddressAnswerLimit:** The maximum number of records that the DNS server will include in a DNS response message. If this value is set to 0x00000000, the DNS server MUST NOT enforce any artificial limit on number of records in a response, and if a response becomes larger than the DNS UDP packet size, the truncation bit MUST be set (see [RFC1035]). The value SHOULD be limited to the range from 0x00000005 to 0x0000001C, inclusive, or the value 0x00000000, but it MAY be any value. The default value MUST be 0x00000000, and the value of zero MUST be allowed.

**AdminConfigured:** A Boolean value indicating whether the server has been configured by an administrator. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value of zero MUST be allowed and treated literally. <130>

**AllowCNAMEatNS:** A Boolean value indicating whether the server will permit the target domain names of NS records to resolve to CNAME records. If true, this pattern of DNS records will be allowed; otherwise, the DNS server will return errors when encountering this pattern of DNS records while resolving queries. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000001, and the value of zero MUST be allowed and treated literally. <131>

**AllowUpdate:** A Boolean value indicating whether the server will permit any DNS update operations. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000001, and the value of zero MUST be allowed and treated literally.

**AutoCacheUpdate:** A Boolean value indicating whether the server should write updated delegation information to persistent storage when it determines that newer information is available. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any

value. The default value MUST be 0x00000000, and the value of zero MUST be allowed and treated literally.

**AutoConfigFileZones:** The type of zones for which SOA and NS records will be automatically configured with the DNS server's local host name as the primary DNS server for the zone when the zone is loaded from file. This property MUST be set to any combination of the following values. If the property value is zero, no automatic configuration will be performed for any zone. The value's range MUST be unlimited. The default value SHOULD be 0x00000001, and the value of zero MUST be allowed and treated literally. <132>

Value	Meaning
0x00000001 ZONE_AUTO_CONFIG_UPDATE	Perform automatic configuration of zones that have a value of "AllowUpdate" not equal to zero.
0x00000002 ZONE_AUTO_CONFIG_STATIC	Perform automatic configuration of zones that have a value of "AllowUpdate" equal to zero.

**BindSecondaries:** A Boolean value indicating whether the server will permit send DNS zone transfer response messages with more than one record in each response if the zone transfer request did not have the characters "MS" appended to it. If true, the DNS server will include only one record in each response if the zone transfer request did not have the characters "MS" appended to it. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value of zero MUST be allowed and treated literally. <133>

**BootMethod:** The DNS\_BOOT\_METHODS (section 2.2.4.1.1) value corresponding to the DNS server's boot method. The value SHOULD be limited to the range from 0x00000000 to 0x00000003, inclusive, but it MAY be any value. The default value MUST be 0x00000000, and the value of zero MUST be allowed and treated literally.

**DebugLevel:** The DNS server MUST ignore any value that is set for this property.

**DefaultAgingState:** A Boolean value that will be used as the default Aging (section 3.1.1.2.1) property value on new zones. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value of zero MUST be allowed and treated literally. <134>

**DefaultNoRefreshInterval:** A value, in hours, that will be used as the default NoRefreshInterval (section 3.1.1.2.1) property value on new zones. The value SHOULD be limited to the range from 0x00000000 to 0x00002238 (1 year), inclusive, but it MAY be any value. The default value MUST be 0x000000A8 (7 days), and the value of zero MUST be allowed and treated literally. <135>

**DefaultRefreshInterval:** A value in hours that will be used as the default RefreshInterval (section 3.1.1.2.1) property value on new zones. The value SHOULD be limited to the range from 0x00000000 to 0x00002238 (1 year), inclusive, but it MAY be any value. The default value MUST be 0x000000A8 (7 days), and the value of zero MUST be allowed and treated literally. <136>

**DeleteOutsideGlue:** A Boolean value indicating whether the DNS server will delete DNS glue records found outside a delegated subzone when reading records from persistent storage. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value of zero MUST be allowed and treated literally.

**DisjointNets:** This is a Boolean value property. The DNS server MUST ignore any value that is set for this property.

**DsLazyUpdateInterval:** A value, in seconds, indicating how frequently the DNS server will submit updates to the directory server without specifying the LDAP\_SERVER\_LAZY\_COMMIT\_OID control ([\[MS-ADTS\]](#) section 3.1.1.3.4.1.7) while processing DNS dynamic update requests. This control instructs the directory server that it may sacrifice durability guarantees on updates to improve performance and is meant to improve DNS server update performance. This control MUST only be sent by the DNS server to the directory server attached to an LDAP update initiated by the DNS server in response to a DNS dynamic update request. If the value is nonzero, LDAP updates performed while processing DNS dynamic update requests MUST NOT specify the LDAP\_SERVER\_LAZY\_COMMIT\_OID control, if a period of fewer than DsLazyUpdateInterval seconds has passed since the last LDAP update specifying this control. If a period of time greater than DsLazyUpdateInterval seconds passes in which the DNS server does not perform an LDAP update specifying this control, the DNS server MUST specify this control on the next update. The value SHOULD be limited to the range from 0x00000000 to 0x0000003c. The default value MUST be 0x00000003, and the value zero MUST be treated as indicating that the DNS server MUST NOT specify the LDAP\_SERVER\_LAZY\_COMMIT\_OID control while processing any DNS dynamic update requests. [<137>](#)

**DsPollingInterval:** The interval, in seconds, at which the DNS server will check the directory service for new or changed DNS zones and records. The value SHOULD be limited to the range from 0x0000001E to 0x00000E10, inclusive, but it MAY be any value. The default value SHOULD be 0x000000B4 (3 minutes), and the value of zero SHOULD be treated as a flag value for the default, but it MAY be allowed and treated literally. Any time a DNS server acting as the primary server for the zone successfully transfers a copy of the zone to a remote DNS server, the DNS server acting as primary MUST copy the zone serial number from the zone transfer response to the zone's Last Transferred Zone Serial Number (section [3.1.1](#)). This value MUST be stored in local non-persistent storage and MUST NOT be replicated to any other DNS server. During polling, if the serial number on a DNS record read from the directory server is higher than the current zone serial number, the current zone serial number MUST be set to the value found in the DNS record. If the DNS server is configured to allow zone transfer for the zone and the current zone serial number is equal to the Last Transferred Zone Serial Number, and if changes to any DNS records for the zone are found during polling where the serial number found in the DNS record is less than or equal to the current zone serial number, the DNS server MUST increment the zone serial number using serial number arithmetic [\[RFC1982\]](#). If the DNS server is not configured to allow zone transfers for the zone the server MUST NOT increment the zone serial number if DNS records are found during polling where the serial number found in the DNS record is less than or equal to the current zone serial number. [<138>](#)

**DsTombstoneInterval:** The age at which tombstone objects in the directory service will be deleted. The value SHOULD be limited to the range from 0x0003F480 (3 days) to 0x0049D400 (8 weeks), inclusive, but it MAY be any value. The default value SHOULD be 0x00127500 (14 days), and the value of zero SHOULD be treated as a flag value for the default, but it MAY be allowed and treated literally. Every day at 2:00 AM local time the DNS server MUST conduct a search of all zones stored in the directory server for nodes which have the dnsTombstoned attribute set to TRUE and an [EntombedTime](#) (section [2.2.2.2.4.23](#)) value greater than **DsTombstoneInterval** seconds in the past. Any such nodes MUST be permanently deleted from the directory server [<139>](#)

**EnableRegistryBoot:** A value which, if present in local persistent configuration at boot time, indicates that the DNS server MUST rewrite the value of the BootMethod property (as described in DNS\_BOOT\_METHODS [2.2.4.1.1](#)), and then delete the value of

EnableRegistryBoot from local persistent configuration. The value of this property MUST be processed before the value of the BootMethod property. The DNS server MUST NOT allow this property to be set using the DNS Server Management Protocol. If the value of this property is locally set to 0x00000000, the DNS server MUST change the value of the BootMethod property to BOOT\_METHOD\_FILE (see section [2.2.4.1.1](#)). If the value of this property is locally set to DNS\_FRESH\_INSTALL\_BOOT\_REGISTRY\_FLAG (0xFFFFFFFF), the DNS server MUST change the value of the BootMethod property to BOOT\_METHOD\_UNINITIALIZED (see section [2.2.4.1.1](#)). If this property is locally set to any other value, the DNS server MUST change the value of the BootMethod property to BOOT\_METHOD\_REGISTRY (see section [2.2.4.1.1](#)). The value's range MUST be unlimited. The default value MUST be DNS\_FRESH\_INSTALL\_BOOT\_REGISTRY\_FLAG, and the value zero MUST be allowed and treated literally.

**EventLogLevel:** All events whose type (as specified in DNS\_EVENTLOG\_TYPES (section [2.2.9.1.2](#))), is less than or equal to **EventLogLevel**, will be written to the event log. The value SHOULD be limited to the range from 0x00000000 to 0x00000007 inclusive, but it MAY be any value. The default value MUST be 0x00000004. Note that a value of **EventLogLevel** in the range from 0x00000004 to 0x00000007, inclusive, will result in all types of event being written to the event log.

**ForceSoaSerial:** User-specified value to use for the SOA **serial number** field [\[RFC1035\]](#) in any new SOA record, or 0x00000000 not to force a user-specified value and to instead use the value 0x00000001 as the default SOA **serial number** value. The value's range MUST be unlimited. The default value MUST be 0x00000000.

**ForceSoaExpire:** User-specified value to use for the SOA expire field [\[RFC1035\]](#) in any new SOA record, or 0x00000000 not to force a user-specified value and to instead use 0x00015180 as the default SOA expire field value. The value's range MUST be unlimited. The default value MUST be 0x00000000.

**ForceSoaRetry:** User-specified value to use for the SOA retry field [\[RFC1035\]](#) in any new SOA record, or 0x00000000 not to force a user-specified value and to instead use the value 0x00000258 as the default SOA retry field value. The value's range MUST be unlimited. The default value MUST be 0x00000000.

**ForceSoaRefresh:** User-specified value to use for the SOA refresh field [\[RFC1035\]](#) in any new SOA record, or 0x00000000 not to force a user-specified value and to instead use the value 0x00000384 as the default SOA refresh field value. The value's range MUST be unlimited. The default value MUST be 0x00000000.

**ForceSoaMinimumTtl:** User-specified value to use for the SOA minimum **TTL** field [\[RFC1035\]](#) in any new SOA record, or 0x00000000 not to force a user-specified value and to instead use the value 0x00000E10 as the default SOA minimum **TTL** field value. The value's range MUST be unlimited. The default value MUST be 0x00000000.

**ForwardDelegations:** A Boolean value indicating how the DNS server will handle forwarding and delegations. If set to true, the DNS server MUST use forwarders instead of a cached delegation when both are available. Otherwise, the DNS server MUST use a cached delegation instead of forwarders when both are available. The value SHOULD be limited to the range from 0x00000000 to 0x00000001 inclusive, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**ForwardingTimeout:** The number of seconds that the DNS server will wait for a response when sending a query to a forwarder before assuming that no response will ever be received. The value SHOULD be limited to the range from 0x00000001 to 0x0000000F, inclusive, but it MAY



be any value. The default value SHOULD be 0x00000003, and the value zero SHOULD be treated as a flag value for the default, but it MAY be allowed and treated literally. <140>

**IsSlave:** A Boolean value indicating whether the DNS server will use normal recursion to resolve queries if all forwarders are unavailable. If true, the DNS server MUST NOT use normal recursion if all forwarders are unavailable. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**LocalNetPriority:** A Boolean value indicating how the DNS server will order IP address records. If true, the DNS server MUST order answer records such that all of those containing IP addresses within the same subnet, when LocalNetPriorityNewMask is applied, as that of the IP address of the client that submitted the query, and are placed first. Also, the server SHOULD randomly order that initial set of answer records with same-subnet IP addresses. If false, the DNS server MUST not reorder answer records in the order that they were retrieved from the database. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value SHOULD be 0x00000001, and the value zero MUST be allowed and treated literally. <141>

**LogFileMaxSize:** The maximum size, in bytes, of the DNS server log file. When the file reaches this size, the DNS server MUST delete the log file and create a new log file. The value's range MUST be unlimited. The default value SHOULD be 0x1DCD6500 (500 MB), and the value zero MUST be allowed and treated literally. <142>

**LogLevel:** The type of information that the DNS server will write to the DNS server log file in DNS\_LOG\_LEVELS (section 2.2.9.1.1) format. The value's range MUST be unlimited. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally. <143>

**LooseWildcarding:** A Boolean value indicating the type of algorithm that the DNS server will use to locate a wildcard node when using a DNS wildcard record [RFC1034] to answer a query. If true, the DNS server will use the first node it encounters with a record of the same type as the query type. Otherwise, the DNS server will use the first node it encounters that has records of any type. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**MaxCacheTtl:** The maximum time duration, in seconds, for which the DNS server can cache a resource record obtained from a remote server as a successful query response. The value SHOULD be limited to the range from 0x00000000 to 0x00278D00 (30 days), inclusive, but it MAY be any value. The default value MUST be 0x00015180 (1 day), and the value zero MUST be allowed and treated literally.

**MaxNegativeCacheTtl:** The maximum time duration, in seconds, for which the DNS server can cache a name error or an empty authoritative response, obtained from a remote server as an unsuccessful query response in its cache (see [RFC2308]). The value SHOULD be limited to the range from 0x00000000 to 0x00278D00 (30 days), inclusive, but it MAY be any value. The default value MUST be 0x00000384 (15 minutes), and the value zero MUST be allowed and treated literally. <144>

**MaxTrustAnchorActiveRefreshInterval:** The maximum time duration, in seconds, for which the DNS server will wait between active refreshes. If an active refresh fails, the retry time MUST be no more than one-tenth of this value. See the calculation of the "queryInterval" and "retryTime" values in [RFC5011]. The value MUST be limited to the range 0x00000E10 (1 hour) to 0x0013C680 (15 days), inclusive. The default value MUST be 0x0013C680 (15 days). <145>

**NameCheckFlag:** The DNS\_NAME\_CHECK\_FLAGS (section [2.2.4.1.2](#)) value corresponding to the level of name checking performed by the DNS server. The value SHOULD be limited to the range from 0x00000000 to 0x00000003, inclusive, but it MAY be any value. The default value SHOULD be 0x00000002 (DNS\_ALLOW\_MULTIBYTE\_NAMES), and the value zero MUST be allowed and treated literally. [<146>](#)

**NoRecursion:** A Boolean value indicating whether the DNS server will perform any recursion. If true, the DNS server MUST NOT recurse and will only answer queries for authoritative data.

**NoUpdateDelegations:** A Boolean value indicating whether the DNS server will accept DNS updates to delegation records of type NS. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value SHOULD be 0x00000000, and the value zero MUST be allowed and treated literally. [<147>](#)

**PublishAutonet:** A Boolean value indicating whether the DNS server will publish local IPv4 addresses in the 169.254.x.x subnet as IPv4 addresses for the local machine's domain name. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally. [<148>](#)

**QuietRecvFaultInterval:** A property used to debug reception of UDP traffic for a recursive query. This property is the minimum time interval, in seconds, starting when the server begins waiting for the query to arrive on the network, after which the server MAY log a debug message indicating that the server is to stop running. If the value is zero or is less than the value of QuietRecvLogInterval, then the value of QuietRecvLogInterval MUST be used. If the value is greater than or equal to the value of QuietRecvLogInterval, then the literal value of QuietRecvFaultInterval MUST be used. The value's range MUST be unlimited. The default value MUST be 0x00000000. The server MAY ignore this property. [<149>](#)

**QuietRecvLogInterval:** A property used to debug reception of UDP traffic for a recursive query. This property is the minimum time interval, in seconds, starting when the server begins waiting for the query to arrive on the network, or when the server logs an eponymous debug message for the query, after which the server MUST log a debug message indicating that the server is still waiting to receive network traffic. If the value is zero, logging associated with the two QuietRecv properties MUST be disabled, and the QuietRecvFaultInterval property MUST be ignored. If the value is non-zero, logging associated with the two QuietRecv properties MUST be enabled, and the QuietRecvFaultInterval property MUST NOT be ignored. The value's range MUST be unlimited. The default value MUST be 0x00000000. The server MAY ignore this property. [<150>](#)

**RecursionRetry:** The time interval, in seconds, for which the DNS server waits before it retries a recursive query to a remote DNS server for which it did not receive a response. The value SHOULD be limited to the range from 0x00000001 to 0x0000000F, inclusive, but it MAY be any value. The default value MUST be 0x00000003, and the value zero SHOULD be treated as a flag value for the default, but it MAY be allowed and treated literally.

**RecursionTimeout:** The time interval, in seconds, for which the DNS server waits for a recursive query response from a remote DNS server. The value SHOULD be limited to the range from 0x00000001 to 0x0000000F, inclusive, but it MAY be any value. The default value SHOULD be 0x00000008, and the value zero SHOULD be treated as a flag value for the default, but it MAY be allowed and treated literally. [<151>](#)

**ReloadException:** A Boolean value indicating whether the DNS server will perform an internal restart if an unexpected fatal error is encountered. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value SHOULD be 0x00000000, and the value zero MUST be allowed and treated literally. [<152>](#)



**RoundRobin:** A Boolean value indicating whether the DNS server will dynamically reorder records in responses to attempt to provide load balancing. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally.

**RpcProtocol:** The DNS\_RPC\_PROTOCOLS (section [2.2.1.1.2](#)) value corresponding to the RPC protocols to which the DNS server will respond. If this value is set to 0x00000000, the DNS server MUST NOT respond to RPC requests for any protocol. The value's range MUST be unlimited, for example, from 0x00000000 to 0xFFFFFFFF. The default value SHOULD be 0x00000005. [<153>](#)

**SecureResponses:** A Boolean value indicating whether the DNS server is configured to cache only those records that are in the same subtree as the name in the original query. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value SHOULD be 0x00000001, and the value zero MUST be allowed and treated literally. [<154>](#)

**SendPort:** The port number to use as the source port when sending UDP queries to a remote DNS server. If set to zero, the DNS server MUST allow the stack to select a random port. The value's range MUST be unlimited. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**ScavengingInterval:** The time interval, in hours, between which the DNS server will schedule DNS stale record scavenging. The value SHOULD be limited to the range from 0x00000000 to 0x00002238, inclusive. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated as a flag to disable scavenging. If the value is non-zero and a "StartScavenging" operation is initiated, the value is ignored and scavenging begins immediately. [<155>](#)

**SocketPoolSize:** The number of **UDP** sockets per address family that the DNS server will use for sending remote queries. The value MUST be limited to the range from 0x00000000 to 0x00002710, inclusive. The default value MUST be 0x000009C4, and the value zero MUST be allowed and treated literally. [<156>](#)

**StrictFileParsing:** A Boolean value indicating whether the DNS server will treat errors encountered while reading zones from a file as fatal. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value SHOULD be 0x00000000, and the value zero MUST be allowed and treated literally.

**SyncDsZoneSerial:** The conditions under which the DNS server should immediately commit uncommitted zone serial numbers to persistent storage. The value SHOULD be limited to the range from 0x00000000 to 0x00000004, inclusive, but it MAY be any value. The default value SHOULD be 0x00000002, and the value zero MUST be allowed and treated literally. [<157>](#)

Value	Meaning
0x00000000 ZONE_SERIAL_SYNC_OFF	Never force immediate commit of serial number to persistent storage.
0x00000001 ZONE_SERIAL_SYNC_SHUTDOWN	Force immediate commit of uncommitted serial numbers to persistent storage when the DNS server is shut down.
0x00000002 ZONE_SERIAL_SYNC_XFER	Force immediate commit of uncommitted serial numbers to persistent storage when the DNS server is shut down or when an uncommitted serial number is advertised during zone transfer.

Value	Meaning
0x00000003 ZONE_SERIAL_SYNC_VIEW	Force immediate commit of uncommitted serial numbers to persistent storage when the DNS server is shut down or when an uncommitted serial number is advertised during zone transfer or when a zone has been loaded or when a zone has been read from Active Directory.<158>
0x00000004 ZONE_SERIAL_SYNC_READ	Force immediate commit of uncommitted serial numbers to persistent storage when the DNS server is shut down or when an uncommitted serial number is advertised during zone transfer or when a zone has been loaded or when a zone has been read from Active Directory.

**UpdateOptions:** The possible zone update settings on the DNS server. Each bit that follows can be used to enable a specific update processing rule to modify the default DNS server update processing behavior. The value's range MUST be unlimited. The default value MUST be 0x0000030F (DNS\_DEFAULT\_UPDATE\_OPTIONS), and the value zero MUST be allowed and treated literally.<159>

The following values are used to disable dynamic updates for non-secure zones.

Value	Meaning
0x00000001 UPDATE_NO_SOA	Disable for SOA records.
0x00000002 UPDATE_NO_ROOT_NS	Disable for root name servers.
0x00000004 UPDATE_NO_DELEGATION_NS	Disable for name servers of delegated zones.
0x00000008 UPDATE_NO_SERVER_HOST	Disable for address records in the DNS server's own host record.

The following values are used to disable dynamic updates for secure zones.

Value	Meaning
0x00000100 UPDATE_SECURE_NO_SOA	Disable for SOA records.
0x00000200 UPDATE_SECURE_NO_ROOT_NS	Disable for root name-servers.
0x00000400 UPDATE_SECURE_NO_DELEGATION_NS	Disable for name-servers of delegated zones.
0x00000800 UPDATE_SECURE_NO_SERVER_HOST	Disable for address records in the DNS server's own host-record.
0x01000000 UPDATE_NO_DS_PEERS	Disable for directory server peers for the DNS server.

Other possible values (regardless of zone type) are as follows.

Value	Meaning
0x00000000 UPDATE_ANY	Server allows dynamic updates for all record types.
0x0000030F DNS_DEFAULT_UPDATE_OPTIONS	Disable all dynamic updates, except for SOA and NS updates for secure zones.
0x01000000 UPDATE_NO_DS_PEERS	Disable relay of server's address record update to remote DNS servers for non-secure zones.

**UseSystemEventLog:** A Boolean value indicating whether the DNS server will write event logs to a repository that is global for the entire system or to a repository that is specific to the DNS server. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**Version:** A read-only 32-bit integer containing the DNS server version in DNSSRV\_VERSION (section [2.2.4.2.1](#)) format. This property is read-only.

**XfrConnectTimeout:** The value, in seconds, that the DNS server will wait, for any DNS TCP connection to a remote DNS server to be established, before assuming that the remote DNS server will not respond. The value SHOULD be limited to the range from 0x00000005 to 0x00000078, inclusive, but it MAY be any value. The default value MUST be 0x0000001E, and the value zero MUST be treated as a flag value for the default. [<160>](#)

**WriteAuthorityNs:** A Boolean value indicating whether the DNS server will include NS records for the root of a zone in DNS responses that are answered using authoritative zone data. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be unlimited. The default value MUST be 0x00000000, and the value zero MUST be allowed. [<161>](#)

**AdditionalRecursionTimeout:** The time interval, in seconds, for which the DNS server waits while recursing to obtain resource records for use in the additional section of DNS responses from a remote DNS server. The value SHOULD be limited to the range from 0x00000000 to 0x0000000F, inclusive, but it MAY be any value. The default value SHOULD be 0x00000004, and the value zero SHOULD be treated as a flag value for the default, but it MAY be allowed and treated literally. [<162>](#)

**AppendMsZoneTransferTag:** A Boolean value indicating whether the DNS server will indicate to the remote DNS servers that it supports multiple DNS records in each zone transfer response message by appending the characters "MS" at the end of zone transfer requests. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value SHOULD be 0x00000000, and the value zero MUST be allowed and treated literally. [<163>](#)

**AutoCreateDelegations:** The possible settings for automatic delegation creation for new zones on the DNS server. The value SHOULD be limited to the range from 0x00000000 to 0x00000002, inclusive, but it MAY be any value. The default value SHOULD be 0x00000002 (DNS\_ACD\_ONLY\_IF\_NO\_DELEGATION\_IN\_PARENT), and the value zero MUST be allowed and treated literally. [<164>](#)

Value	Meaning
0x00000000 DNS_ACD_DONT_CREATE	The server does not create delegations automatically.
0x00000001 DNS_ACD_ALWAYS_CREATE	The server always creates delegations automatically.
0x00000002 DNS_ACD_ONLY_IF_NO_DELEGATION_IN_PARENT	The server creates a new delegation in the parent zone only if there is no existing delegation present for the zone.

**BreakOnAscFailure:** A Boolean value indicating whether the DNS server will execute a debug break if an error is encountered during security negotiation for secure updates. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally. <165>

**CacheEmptyAuthResponses:** A Boolean value indicating if the DNS server will store empty authoritative responses [RFC2308] in the cache. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally. <166>

**DirectoryPartitionAutoEnlistInterval:** The interval, in seconds, at which the DNS server will attempt to enlist itself in the DNS domain partition and DNS forest partition if it is not already enlisted. The value SHOULD be limited to the range from 0x00000E10 (1 hour) to 0x00ED4E00 (180 days), inclusive, but it MAY be any value. The default value MUST be 0x00015180 (1 day), and the value zero SHOULD be treated as a flag value for the default, but it MAY be allowed and treated literally. <167>

**DisableAutoReverseZones:** A Boolean value indicating whether the DNS server will disable the automatic server boot-time creation of three authoritative reverse lookup zones (0.in-addr.arpa, 127.in-addr.arpa, and 255.in-addr.arpa). The value SHOULD be limited to the range from 0x00000000 to 0x00000001, inclusive, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**EDnsCacheTimeout:** The interval, in seconds, for which the DNS server will cache the remote DNS server support of EDNS [RFC2671]. The value SHOULD be limited to the range from 0x0000000A to 0x00015180 (1 day), inclusive, but it MAY be any value. The default value SHOULD be 0x00000384 (15 minutes), and the value zero SHOULD be treated as a flag value for the default, but it MAY be allowed and treated literally. <168>

**EnableDirectoryPartitions:** A Boolean value indicating whether the DNS server will support application directory partitions. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally. <169>

**EnableDnsSec:** A Boolean value indicating whether the DNS server will perform additional query processing for secure DNS records, as specified in [RFC4033], [RFC4034], and [RFC4035]. The value SHOULD be limited to a range of 0x00000000 to 0x00000001, inclusive, but it MAY be any value. The default value MUST be 0x00000001 and the value zero MUST be allowed and treated literally. <170>

**EnableEDnsProbes:** A Boolean value indicating whether the DNS server will include EDNS [RFC2671] records in remote queries (with the possible exception of queries sent to a remote Global Names Zone (GNZ); see "GlobalNamesEnableEDnsProbes" later in this section). The

value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value SHOULD be 0x00000001, and the value zero MUST be allowed and treated literally. <171>

**EnableEDnsReception:** A Boolean value indicating whether the DNS server will accept queries that contain an EDNS [\[RFC2671\]](#) record. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally. <172>

**EnableIPv6:** A Boolean value indicating whether the DNS server will listen on local IPv6 addresses. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value SHOULD be 0x00000001, and the value zero MUST be allowed and treated literally. <173>

**EnableForwarderReordering:** A Boolean value indicating whether the DNS server will perform forwarder list reordering of the DynamicForwarder list (section [3.1.1.1.2](#)) at run time. The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally. <174>

**EnableIQueryResponseGeneration:** A Boolean value indicating whether the DNS server will fabricate IQUERY responses. If set to true, the DNS server MUST fabricate IQUERY responses when it receives queries of type IQUERY. Otherwise, the DNS server will return an error when such queries are received. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally. <175>

**EnableOnlineSigning:** A Boolean value indicating whether the DNS server will sign directory server-integrated zones when these zones are loaded or when records in a zone are added, removed, or modified. The value also indicates whether signing new zones will be permitted. The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally. <176>

**EnableSendErrorSuppression:** A Boolean value indicating whether the DNS server will attempt to suppress large volumes of DNS error responses sent to remote IP addresses that may be attempting to attack the DNS server. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally. <177>

**EnableUpdateForwarding:** A Boolean value indicating whether the DNS server will forward updates received for secondary zones to the primary DNS server for the zone. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally. <178>

**EnableWinsR:** A Boolean value indicating whether the DNS server will perform NetBIOS name resolution in order to map IP addresses to machine names while processing queries in zones where WINS-R information has been configured. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally. <179>

**ForceDsaBehaviorVersion:** The minimum value to use as the behavior version of the local directory server, or 0xFFFFFFFF. Values for this property MUST be limited to those specified in "msDS-Behavior-Version: DC Functional Level", [MS-ADTS] (section [6.1.4.4](#)), in addition to 0xFFFFFFFF, which MUST be the default value. The DNS Server reads the **msDS-BehaviorVersion** attribute of the local directory server's "nTDSDSA" object and compares it with this property. If the value read from the directory server is greater than the specified ForceDsaBehaviorVersion, or if ForceDsaBehaviorVersion is 0xFFFFFFFF, this property is set to

the value read from the directory server. This property SHOULD be returned by the DNS Server in the **dwDsDsaVersion** field of the DNS\_RPC\_SERVER\_INFO structure (section [2.2.4.2.2](#)) when processing the "ServerInfo" operation of the R\_DnssrvQuery method (section [3.1.4.2](#)).

**ForceDomainBehaviorVersion:** The minimum value to use as the behavior version of the domain, or 0xFFFFFFFF. Values for this property MUST be limited to those specified in "msDS-Behavior-Version: Domain NC Functional Level", [MS-ADTS] (section [6.1.4.4](#)), in addition to 0xFFFFFFFF, which MUST be the default value. The DNS Server reads the **msDS-BehaviorVersion** attribute of the domain's crossRef object and compares it with this property. If the value read from the directory server is greater than the specified ForceDomainBehaviorVersion, or if ForceDomainBehaviorVersion is 0xFFFFFFFF, this property is set to the value read from the directory server. This property SHOULD be returned by the DNS Server in the **dwDsDomainVersion** field of the DNS\_RPC\_SERVER\_INFO structure (section [2.2.4.2.2](#)) when processing the "ServerInfo" operation of the R\_DnssrvQuery method (section [3.1.4.2](#)).

**ForceForestBehaviorVersion:** The minimum value to use as the behavior version of the forest, or 0xFFFFFFFF. Values for this property MUST be limited to those specified in "msDS-Behavior-Version: Forest Functional Level", [MS-ADTS] section 6.1.4.4, in addition to 0xFFFFFFFF, which MUST be the default value. The DNS Server reads the **msDS-BehaviorVersion** attribute of the forest's "crossRefContainer" object and compares it with this property. If the value read from the directory server is greater than the specified ForceForestBehaviorVersion, or if ForceForestBehaviorVersion is 0xFFFFFFFF, this property is set to the value read from the directory server. This property SHOULD be returned by the DNS Server in the **dwDsForestVersion** field of the DNS\_RPC\_SERVER\_INFO structure (section [2.2.4.2.2](#)) when processing the [2.2.4.2.2](#) operation of the R\_DnssrvQuery method (section [2.2.4.2.2](#)).

**HeapDebug:** A Boolean value indicating whether the DNS server will execute a debug break when internal memory corruption is detected. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally. [<180>](#)

**LameDelegationTtl:** The number of seconds that must elapse before the DNS server will re-query DNS servers of the parent zone when a **lame delegation** is encountered. The value SHOULD be limited to the range from 0x00000000 to 0x00278D00 (30 days), inclusive, but it MAY be any value. The default value SHOULD be 0x00000000, and the value zero MUST be allowed and treated literally. [<181>](#)

**LocalNetPriorityNetMask:** A value which specifies the network mask the DNS server will use to sort IPv4 addresses. A value of 0xFFFFFFFF indicates that the DNS server MUST use traditional IPv4 network mask for the address. Any other value is a network mask, in host byte order that the DNS server MUST use to retrieve network masks from IP addresses for sorting purposes. The value's range MUST be unlimited. The default value MUST be 0x000000FF, and the value zero MUST be allowed and treated literally. [<182>](#)

**MaxCacheSize:** The maximum size of memory, in kilobytes, that the DNS server SHOULD use to store DNS data in the cache. The value SHOULD be limited to the range from 0x000001F4 to 0xFFFFFFFF, inclusive, or 0x00000000, but it MAY be any value. The default value SHOULD be 0x00000000, which MUST be allowed and treated as a flag value for no limit on maximum size of memory. If the value is nonzero, the DNS server SHOULD treat this as a soft limit, allowing it to be exceeded for limited durations, and also attempt to limit cache memory to 90 percent of this value. [<183>](#)

**MaximumSignatureScanPeriod:** The maximum time, in seconds, before which the server MUST scan all signed zones for a signature refresh. The value MUST be limited to the range



0x00000E10 (1 hour) to 0x00278D00 (30 days), inclusive. The default value MUST be 0x00015180 (1 day), and the value zero MUST be allowed and treated literally. <184>

**MaxResourceRecordsInNonSecureUpdate:** The maximum number of resource records that the DNS server will accept in a single DNS update request. The value SHOULD be limited to the range from 0x0000000A to 0x00000078, inclusive, but it MAY be any value. The default value SHOULD be 0x0000001E, and the value zero SHOULD be treated as a flag value for the default, but it MAY be allowed and treated literally. <185>

**OperationsLogLevel:** The operations (in addition to those specified by OperationsLogLevel2) that will be logged to the DNS server log file using any combination of the values that follow. The value's range MUST be unlimited. The default value SHOULD be 0x00000000, and the value zero MUST be allowed and treated literally. <186>

Value	Meaning
0x00000001 DNSLOG_WRITE_THROUGH	The server saves operational logging information to persistent storage.
0x00000010 DNSLOG_EVENT	The server logs event logging information to the log file.
0x00000020 DNSLOG_INIT	The server logs operational logging information to the log file for server start and stop activities.
0x00002000 DNSLOG_DSPOLL	The server logs operational logging information to the log file for activities related to loading a zone from the directory server.
0x00004000 DNSLOG_DSWRITE	The server logs operational logging information to the log file for activities related to writing zone data to the directory server.
0x00020000 DNSLOG_TOMBSTN	The server logs operational logging information to the log file for activities related to updating tombstoned nodes.
0x00100000 DNSLOG_LOOKUP	The server logs operational logging information to the log file for local resource lookup activities.
0x00200000 DNSLOG_RECURSE	The server logs operational logging information to the log file for activities performed during recursive query lookup.
0x00400000 DNSLOG_REMOTE	The server logs operational logging information to the log file for activities related to interaction with remote name servers.

**OperationsLogLevel2:** The operations (in addition to those specified by OperationsLogLevel) that will be logged to the DNS server log file using any combination of the values that follow. The value's range MUST be unlimited. The default value SHOULD be 0x00000000, and the value zero MUST be allowed and treated literally. <187>

Value	Meaning
0x01000000 DNSLOG_PLUGIN	The server logs operational logging information to the log file for activities related to interaction with plug-in DLLs.

**MaximumUdpPacketSize:** The maximum UDP packet size, in bytes, that the DNS server can accept. The value MUST be limited to 0x00000200 to 0x00004000.<188> The server MUST return an error if an attempt is made to change the value of this property through this protocol. This property may only be changed by modifying the value in persistent storage.<189>

**RecurseToInternetRootMask:** The DNS server MUST ignore any value set for this property.

**SelfTest:** A mask value indicating whether data consistency checking should be performed once, each time the service starts. If the check fails, the server posts an event log warning. If the least significant bit (regardless of other bits) of this value is one, the DNS server will verify for each active and update-allowing primary zone, that the IP address records are present in the zone for the zone's SOA record's master server. If the least significant bit (regardless of other bits) of this value is zero, no data consistency checking will be performed. The value's range MUST be from 0x00000000 to 0xFFFFFFFF, inclusive. The default value MUST be 0xFFFFFFFF.

**SilentlyIgnoreCNameUpdateConflicts:** A Boolean value indicating whether the DNS server will ignore CNAME conflicts during DNS update processing. The value SHOULD be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.<190>

**TcpReceivePacketSize:** The maximum TCP packet size, in bytes, that the DNS server can accept. The value MUST be limited to the range from 0x00004000 to 0x00010000, inclusive. Values outside of this range MUST cause the server to return an error. The default value MUST be 0x00010000.<191>

**XfrThrottleMultiplier:** The multiple used to determine how long the DNS server should refuse zone transfer requests after a successful zone transfer has been completed. The total time for which a zone will refuse another zone transfer request at the end of a successful zone transfer is computed as this value multiplied by the number of seconds required for the zone transfer that just completed. The server SHOULD refuse zone transfer requests for no more than ten minutes. The value SHOULD be limited to the range from 0x00000000 to 0x00000064, inclusive, but it MAY be any value. The default value MUST be 0x0000000A, and the value zero MUST be allowed and treated literally.<192>

**UdpRecvThreadCount:** The number of receive threads handling incoming UDP traffic that the server MUST run simultaneously. The value MUST be limited to the range 0x00000000 to 0x00000800, inclusive. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated as the count of processors in the machine.<193>

The DNS server SHOULD also support the following properties.<194>

**AllowMsdcsLookupRetry:** A Boolean value indicating whether the DNS server will attempt to retry failed lookup operations in the immediate parent of the zone where the lookup was originally performed. This lookup retry MUST only be applied if the name of the zone where the lookup was originally performed began with the string "\_msdcs" and the immediate parent of the zone where the lookup was originally performed is present on the DNS server. The value's range MUST be limited to 0x00000000 and 0x00000001. The default value SHOULD be 0x00000001, and the value zero MUST be allowed and treated literally.<195>

**AllowReadOnlyZoneTransfer:** A Boolean value indicating whether the DNS server will allow zone transfers for zones that are stored in the directory server when the directory server does not support write operations. The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.



**DsBackgroundLoadPaused:** A Boolean value indicating whether the DNS server is enabled to pause background loading of information from directory server if a node is found with same node-name same as pointed to by DsBackgroundPauseName. The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally. <196>

**DsMinimumBackgroundLoadThreads:** The minimum number of background threads that the DNS server will use to load zone data from the directory service. The value MUST be limited to the range from 0x00000000 to 0x00000005, inclusive. If the value is 0x00000000, then the DNS server MUST NOT start background threads to load zone data from the directory service. The default value MUST be 0x00000001, and the value zero MUST be treated as allowed.

**DsRemoteReplicationDelay:** The minimum interval, in seconds, that the DNS server must wait between the time it determines that a single object has changed on a remote directory server and the time it attempts to replicate the single object change. The value MUST be limited to the range from 0x00000005 to 0x00000E10, inclusive. The default value MUST be 0x0000001E, and the value zero MUST be treated as a flag value for the default.

**EnableDuplicateQuerySuppression:** A Boolean value indicating whether the DNS server will not send remote queries when there is already a remote query with the same name and query type outstanding. The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally.

**EnableGlobalNamesSupport:** A Boolean value indicating whether the DNS server will use any GNZ data while responding to DNS queries and updates. The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**EnableVersionQuery:** This property controls what version information the DNS server will respond with when a DNS query with class set to CHAOS and type set to TXT is received. The value's range MUST be limited to the values in the following table. The default value SHOULD be 0x00000000. <197>

Value	Meaning
0x00000000 DNS_VERSION_QUERY_OFF	No version information will be returned.
0x00000001 DNS_VERSION_QUERY_FULL	The server responds with major operating system version, minor operating system version, and operating system revision.
0x00000002 DNS_VERSION_QUERY_MINIMAL	The server responds with major operating system version and minor operating system version.

**EnableRsoForRodc:** A Boolean value indicating whether the DNS server will attempt to replicate single updated DNS objects from remote directory servers ahead of normally scheduled replication when operating on a directory server that does not support write operations. The value MUST be limited to 0x00000000 and 0x00000001, but it MAY be any value. The default value SHOULD be 0x00000001, and the value zero MUST be allowed and treated literally.

**ForceRODCMode:** A Boolean value indicating whether the DNS server will always operate as if the directory server does not support write operations. If TRUE, the DNS server MUST operate as if the directory server does not support write operations; otherwise, the DNS server MUST query the directory server to determine whether it supports write operations. The value MUST

be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally. <198>

**GlobalNamesAlwaysQuerySrv:** A Boolean value that indicates, when FALSE, that the DNS server will attempt to use GNZ service records (SRV records named "\_globalnames.\_msdcs.<forestroot>") from the server's cache when updating the list of remote DNS servers hosting a GNZ, or when TRUE, that the server MUST always attempt a remote DNS query for such records. The value MUST be ignored if the server hosts a GNZ. The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000000, and the value zero MUST be allowed.

**GlobalNamesBlockUpdates:** A Boolean value indicating whether the DNS server will block updates in authoritative zones if they are for FQDNs that would collide with labels found in the GNZ. If the value of this property is 0x00000000, then a check for this collision MUST NOT be performed.

To test whether a name collides with a name present in the GNZ, the DNS server MUST extract the relative portion of the name that is being updated by removing the rightmost labels which comprise the zone name, and then perform a case-insensitive search in the locally hosted GNZ for a name matching the remaining labels. If a match for these labels is found in the locally hosted GNZ and the value of this property is 0x00000001 then the update MUST be blocked.

The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000001, and the value zero MUST be allowed.

**GlobalNamesEnableEDnsProbes:** A Boolean value indicating whether the DNS server will honor the "EnableEDnsProbes" Boolean value for a remote GNZ. A value of TRUE indicates that the server MUST attempt to use EDNS for queries sent to a remote GNZ if the Boolean value of "EnableEDnsProbes" is also TRUE, and otherwise MUST NOT attempt to use EDNS for such queries. A value of FALSE indicates that the server MUST NOT attempt to use EDNS for queries sent to a remote GNZ, regardless of the value of "EnableEDnsProbes". The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000001, and the value zero MUST be allowed.

**GlobalNamesPreferAAAA:** A Boolean value indicating whether the DNS server will prefer type AAAA address records to type A records when sending queries to a remote DNS server that is hosting a GNZ. If the value is 0x00000000 then queries to a remote DNS server hosting a GNZ MUST be sent using IPv4 if any IPv4 addresses for the remote DNS server name can be found. If no IPv4 addresses are found for the remote DNS server name, then IPv6 addresses MUST be used. If the value of this property is 0x00000001, then IPv6 addresses for the remote DNS server MUST be used, and IPv4 addresses MUST NOT be used unless no IPv6 addresses can be found. The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**GlobalNamesQueryOrder:** A Boolean value indicating whether the DNS server will prefer GNZ or authoritative zone data when determining what data to use to answer queries. If TRUE, the DNS server MUST prefer authoritative zone data; otherwise, the DNS server MUST prefer GNZ data. The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**GlobalNamesSendTimeout:** The number of seconds the DNS server will wait when sending a query to a remote GNZ before assuming that no answer will ever be received. The value MUST be limited to the range from 0x00000001 to 0x0000000F, inclusive. The default value MUST be 0x00000003, and the value zero MUST be treated as a flag value for the default.

**GlobalNamesServerQueryInterval:** The maximum interval, in seconds, between queries to refresh the set of remote DNS servers hosting the GNZ. The value MUST be limited to the range from 0x0000003C (60 seconds) to 0x00278D00 (30 days), inclusive. The default value MUST be 0x00005460 (6 hours), and the value zero MUST be treated as a flag value for the default.

**RemoteIPv4RankBoost:** A value to add to all IPv4 addresses for remote DNS servers when selecting between IPv4 and IPv6 remote DNS server addresses. The value MUST be limited to the range from 0x00000000 to 0x0000000A, inclusive. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**RemoteIPv6RankBoost:** A value to add to all IPv6 addresses for remote DNS servers when selecting between IPv4 and IPv6 remote DNS server addresses. The value MUST be limited to the range from 0x00000000 to 0x0000000A, inclusive. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated literally.

**MaximumRodcRsoAttemptsPerCycle:** The maximum number of queued single object replication operations that should be attempted during each five minute interval of DNS server operation. The value MUST be limited to the range from 0x00000000 to 0x000F4240, inclusive. The default value MUST be 0x00000064, and the value 0x00000000 MUST be allowed and treated as no limitation on the number of queued single object replication operations.

**MaximumRodcRsoQueueLength:** The maximum number of single object replication operations that may be queued at any given time by the DNS server. The value MUST be limited to the range from 0x00000000 to 0x000F4240, inclusive. If the value is 0x00000000 the DNS server MUST NOT enforce an upper bound on the number of single object replication operations queued at any given time. The default value MUST be 0x0000012C, and the value zero MUST be allowed.

**EnableGlobalQueryBlockList:** A Boolean value indicating whether the DNS server should block queries in locally hosted primary zones that match entries in the **GlobalQueryBlockList** property (see section [3.1.1.1.4](#)). If the value of this property is 0x00000001, when answering a query using a locally hosted primary zone the DNS server MUST check to see if the relative portion of the query name matches any value in the GlobalQueryBlockList property. If a match is found the DNS server MUST return a name error response instead of a positive answer. The DNS server MUST NOT apply this algorithm to the name of the zone. The block list MUST only be applied to records within each zone. The DNS server MUST NOT perform this check if the value of the EnableGlobalQueryBlockList property is 0x00000000. The value MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally. [<199>](#)

The DNS server SHOULD [<200>](#) also support the following properties.

**OpenACLOnProxyUpdates:** A [Boolean](#) value indicating whether the DNS server allows sharing of DNS records with the DnsUpdateProxy group when processing updates in secure zones that are stored in the directory service. During secure dynamic update [\[RFC3645\]](#) negotiation, the DNS server SHOULD check whether DNS records exist in the zone under the name specified in the update request, [\[RFC2136\]](#). If so, the server SHOULD check the client credentials against the access control lists associated with the existing records in the **directory service (DS)** (see [\[MS-ADTS\]](#) section 5.1.3), before allowing the requested records to be created or modified. If there are no records for the update request name, DNS server SHOULD create records requested by the client, and associate those records with the client's credentials. If OpenACLOnProxyUpdates is set to TRUE, when a member of the DnsUpdateProxy group updates a DNS resource record for which that member has write access, the record access control lists SHOULD be adjusted to grant write privileges to all clients with credentials. The

value of OpenACLOnProxyUpdates MUST be limited to 0x00000000 and 0x00000001. The default value MUST be 0x00000001, and the value zero MUST be allowed and treated literally.

**CacheLockingPercent:** The percentage of the original time-to-live value for which all cache entries from non-authoritative responses MUST be locked and MUST NOT be overwritten by data found in subsequent non-authoritative responses. Locked cache entries MUST still be considered for removal from the cache if the soft limit of the maximum cache size is reached (see the MaxCacheSize property, described previously in this section). The value MUST be limited to the range from 0x00000000 to 0x00000064, inclusive. The default value MUST be 0x00000064, and the value zero MUST be allowed and treated literally.

**ZoneWritebackInterval:** The interval at which the data for a file backed zone is periodically synced to its data file in persistent storage, in case the in-memory copy of the zone is not yet committed to the data file <201>. The values can range from 1 minute to 1 week (7\*24\*60). When ZoneWritebackInterval is set to a value of 0, the automatic flushing of zone data to the data file is disabled. This setting applies to all zone scopes present on the zone as well.

### 3.1.1.1.2 DNS Server Address Array Properties

The following properties are IP address arrays (specified by type IDs DNSSRV\_TYPEID\_ADDRARRAY or DNSSRV\_TYPEID\_IPARRAY, and formatted as [DNS ADDR ARRAY \(section 2.2.3.2.3\)](#) or as [IP4 ARRAY \(section 2.2.3.2.1\)](#), respectively). The DNS server SHOULD support both types. <202>

**Forwarders:** A list of IP addresses indicating to which remote DNS servers this DNS server will forward unresolved queries. If the list is empty then no forwarding will be performed.

**DynamicForwarders:** A list of IP addresses indicating to which remote DNS servers this DNS server will forward unresolved queries where the list may be reordered by the server, sorted by lower prior response time. If the list is empty, no forwarding will be performed if the Forwarders property (section [3.1.1.1.2](#)) is also empty. If the EnableForwarderReordering property (section [3.1.1.1.1](#)) is 0x00000000, the Forwarders property will be used; otherwise, the DynamicForwarders property will be used. <203>

**ListenAddresses:** A list of local IP addresses on which the DNS server should listen for DNS request messages.

The DNS Server SHOULD <204> also support the following properties:

**BreakOnReceiveFrom:** The DNS server will execute a debug break, for debugging purposes, when a DNS query message is received from any IP address in this list. If NULL then no debug breaks will be executed when query messages are received.

**BreakOnUpdateFrom:** The DNS server will execute a debug break, for debugging purposes, when a DNS update message is received from any IP address in this list. If NULL then no debug breaks will be executed when update messages are received.

**LogIPFilterList:** A list of IP addresses indicating traffic to or from which IP addresses should be logged when logging is enabled. If NULL then traffic to and from all IP addresses should be logged when logging is enabled.

### 3.1.1.1.3 DNS Server String Properties

The following properties are strings.

The DNS Server SHOULD <205> support the following properties:

**DomainDirectoryPartitionBaseName:** The first name component to use as the name of the DNS domain partition in UTF-8 format. If empty, the DNS server uses "DomainDnsZones". On input, the type ID for this property is DNSSRV\_TYPEID\_LPWSTR, listed in section [2.2.1.1.1](#). On output, the type ID for this property is DNSSRV\_TYPEID\_LPSTR, listed in section [2.2.1.1.1](#).

**ForestDirectoryPartitionBaseName:** The first name component to use as the name of the DNS forest partition in UTF-8 format. If empty, the DNS server uses "ForestDnsZones". On input, the type ID for this property is DNSSRV\_TYPEID\_LPWSTR, listed in section [2.2.1.1.1](#). On output, the type ID for this property is DNSSRV\_TYPEID\_LPSTR, listed in section [2.2.1.1.1](#).

**LogFilepath:** An absolute or relative pathname of the log file to which the DNS server SHOULD [<206>](#) output logging information. This protocol only treats the pathname as a string, it does not constrain the syntax in any way. This property is encoded as a null-terminated Unicode string, and the type ID for this property is DNSSRV\_TYPEID\_LPWSTR, listed in section [2.2.1.1.1](#).

**ServerLevelPluginDll:** An absolute pathname of a dynamic link library that the DNS server may use to resolve unknown names, or an empty string to remove or disable the previously selected DLL. This protocol only treats the pathname as a string, it does not constrain the syntax in any way. This property is encoded as a null-terminated Unicode string, and the type ID for this property is DNSSRV\_TYPEID\_LPWSTR, listed in section [2.2.1.1.1](#).

The DNS Server SHOULD [<207>](#) also support the following properties:

**DsBackgroundPauseName:** A single-label name in UTF-8 format which, when matched to the current node name being read during zone background loading from the directory service, will cause the DNS server to set the DsBackgroundLoadPaused (section [3.1.1.2.1](#)) property to TRUE and stop loading data in the background until the DsBackgroundLoadPaused property is reset to FALSE. The type ID for this property is DNSSRV\_TYPEID\_LPSTR, listed in section [2.2.1.1.1](#). This property MUST NOT be possible to set using this protocol.

**DoNotRoundRobinTypes:** An array of record types for which the DNS server will disable round robin. The value MUST be an ASCII string containing integers separated by spaces, in either hexadecimal format (prefixed with the ASCII characters '0x' or '0X'), octal format (prefixed with an ASCII '0' and no subsequent 'x' or 'X'), or decimal format (otherwise). Each integer MUST correspond to a DNS record type number, and SHOULD be a value in the range from 0x00000000 to 0x000000FF, though the element values MAY be any value in the range 0x00000000 to 0xFFFFFFFF. This value MUST NOT be changeable via the DNSP protocol. [<208>](#)

**RootTrustAnchorsURL:** A URL in UTF-8 format from which the DNS root trust anchors will be downloaded when the "RetrieveRootTrustAnchors" operation is invoked. The default value MUST be <https://data.iana.org/root-anchors/root-anchors.xml>. [<209>](#)

#### 3.1.1.1.4 DNS Server String List Properties

The following properties are string lists in UTF-8 format. The type ID for this property is DNSSRV\_TYPEID\_UTF8\_STRING\_LIST, listed in section [2.2.1.1.1](#).

The DNS Server SHOULD [<210>](#) support the following properties:

**GlobalQueryBlockList:** A list of single-label strings for which queries will be blocked if the query name matches any name in this list within any primary zone on the DNS server. The DNS server MUST NOT block queries if the value of the **EnableGlobalQueryBlockList** property (see section [3.1.1.1.1](#)) is 0x00000000. For a description of the DNS server's behavior when

blocking queries, see the description of the **EnableGlobalQueryBlockList** property (see section [3.1.1.1.1](#)).

When the DNS server process starts, if for both this property and for the **EnableGlobalQueryBlockList** property, no value is found in persistent storage, then the DNS server MUST construct and store as the value of this property a generic list of query names that should be blocked, and MUST set the value of the **EnableGlobalQueryBlockList** property to 0x00000001 in persistent storage. If the **EnableGlobalQueryBlockList** property is already set to 0x00000001, then an undefined **GlobalQueryBlockList** property is treated as an empty list.

Construction of the generic list of query names that should be blocked MUST be performed as follows: the DNS server MUST enumerate all locally hosted primary and secondary zones. If no locally hosted primary or secondary zone contains a DNS record for the name "isatap" that is not of type TXT then "isatap" MUST be added to the list. If no locally hosted primary or secondary zone contains a DNS record for the name "wpad" that is not of type TXT then "wpad" MUST be added to the list.

**SocketPoolExcludedPortRanges:** A list of numeric port number ranges (for example, {"4000-5000", "34000-34000"}) for which listen sockets will not be opened by the DNS server. Ranges must have the smaller number (if unequal) first, and are inclusive. Even single port exclusions must be specified as a range. Inputs less than 1 or greater than 65,535 are interpreted as 1 and 65,535 respectively.

### 3.1.1.2 DNS Zone Configuration Information

The list of names that are used in (name, value) pairs in DNS Zone metadata is given below.

#### 3.1.1.2.1 DNS Zone Integer Properties

The following properties are 32-bit integers. The term Boolean, as used below, means a 32-bit integer where a value of 0x00000000 indicates that the stated property is false, and any nonzero value indicates that the stated property is true.

**AllowUpdate:** The [DNS\\_ZONE\\_UPDATE \(section 2.2.6.1.1\)](#) value for the zone. The value for this property is limited to those listed in the table in section [2.2.6.1.1](#). If this property's value is changed from any value to ZONE\_UPDATE\_SECURE, the DNS server MUST set the zone's **Time Zone Secured** (section [3.1.1](#)) property to the current time expressed as the number of seconds since 12:00 A.M. January 1, 1601 Coordinated Universal Time (UTC).

**DsIntegrated:** A Boolean indicating whether the zone is stored in the directory server. This property is read-only.

**DsRecordAlgorithms:** The value of the cryptographic hash algorithm used to generate DS records written to a file named "dsset-<ZoneName>" [<211>](#) when the zone is first signed and whenever the DNSKEY record set for the zone is changed. The value MUST be limited to the values in the following table. The default value MUST be 0x00000003. [<212>](#)

Value	Meaning
0x00000000 DNS_ZONE_GENERATE_DS_NONE	Do not generate DS records.
0x00000001 DNS_ZONE_GENERATE_DS_SHA1	Use SHA-1 to generate DS records.



Value	Meaning
0x00000002 DNS_ZONE_GENERATE_DS_SHA256	Use SHA-256 to generate DS records.
0x00000004 DNS_ZONE_GENERATE_DS_SHA384	Use SHA-384 to generate DS records.

**DSRecordSetTTL:** The TTL value, in seconds, to assign to any new DS record created for this zone and written to the "dsset-<ZoneName>" file during zone signing or key rollover. The value MUST be limited to the range 0x00000000 to 0x00093A80 (1 week), inclusive. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated as the zone default TTL. <213>

**DNSKEYRecordSetTTL:** The TTL value, in seconds, that should be assigned to any new DNSKEY record created for this zone during zone signing or key rollover. The value MUST be limited to the range 0x00000000 to 0x00093A80 (1 week), inclusive. The default value MUST be 0x00000000, and the value zero MUST be allowed and treated as the zone default TTL. <214>

**IsKeymaster:** A Boolean indicating whether the DNS server is the key master for this zone. This property can be modified only by using the "TransferKeymasterRole" operation of the [R\\_DnssrvOperation \(Opnum 0\) \(section 3.1.4.1\)](#) method call. The default value MUST be 0x00000000. <215>

**IsSigned:** A Boolean indicating whether the zone is signed via Online Signing. This property can be modified only by using the "ZoneSign" or "ZoneUnsign" operation of the [R\\_DnssrvOperation \(Opnum 0\) \(section 3.1.4.1\)](#) method call. The default value MUST be 0x00000000. <216>

**LogUpdates:** A Boolean indicating whether updates on this zone should be logged to permanent storage.

**MaintainTrustAnchor:** This property controls how the DNS server maintains the list of forest-wide Trust Anchors as key rollover takes place for signing key descriptors whose fIsKSK flag is set. As the rollover progresses, new keys are generated and added to the forest-wide TrustAnchors zone, and old keys are removed. The value's range MUST be limited to the values in the following table. The default for this value SHOULD <217> be 0x00000000.

Value	Meaning
0x00000000 DNS_ZONE_MAINTAIN_TA_NONE	Trust Anchors are not updated as key rollover proceeds.
0x00000001 DNS_ZONE_MAINTAIN_TA_DNSKEY	Keys are stored in the forest-wide TrustAnchors zone as DNSKEY records as the key rollover proceeds.

**NoRefreshInterval:** The No Refresh interval value, in hours, for the zone. The value 0x00000000 MUST be treated as a flag value for the value of "DefaultNoRefreshInterval" (section [3.1.1.1.1](#)).

**NSEC3HashAlgorithm:** The algorithm ID used for hashing node owner names in zones signed with NSEC3 as described in [RFC5155](#) section 3.1.1. The value's range MUST be limited to the values in the following table. The default for this value SHOULD <218> be 0x00000001.

Value	Meaning
0x00000001 DNS_NSEC3_HASH_ALG_ID_SHA1	Use SHA-1 to hash owner names.

**NSEC3Iterations:** The number of additional iterations that the hashing function is used when generating hashed owner names for zones signed with NSEC3, as described in [\[RFC5155\]](#) section 3.1.3 and section 5. The value's range MUST be 0x00000000 to 0x000009C4, inclusive. The default value SHOULD [<219>](#) be 0x00000032.

**NSEC3OptOut:** A Boolean indicating whether NSEC3 records in a zone signed with NSEC3 have their Opt-Out flag set, as described in [\[RFC5155\]](#) section 3.1.2.1. The default value SHOULD [<220>](#) be 0x00000000.

**NSEC3RandomSaltLength:** When zones are signed with NSEC3, salt may be applied to the hashing function when hashed owner names are generated, as described in [\[RFC5155\]](#) section 3.1.5 and section 5. The "NSEC3RandomSaltLength" is the length, in octets, of randomly generated salt. The value 0x00000000 MUST be treated as a flag indicating that the DNS server MUST NOT generate salt randomly but MUST use the "NSEC3UserSalt" zone property. For any other value, the DNS server MUST generate a random salt of the specified length to be used when generating hashed owner names. The value's range MUST be 0x00000000 to 0x000000FF, inclusive. The default value SHOULD [<221>](#) be 0x00000008.

**NotifyLevel:** The DNS\_ZONE\_NOTIFY\_LEVEL (section [2.2.5.1.3](#)) value for the zone. The value for this property is limited to those listed in the table in section [2.2.5.1.3](#).

**ParentHasSecureDelegation:** A Boolean indicating whether this zone has a **secure delegation** from a parent zone. The default value SHOULD [<222>](#) be 0x00000000.

**PropagationTime:** The expected time, in seconds, that it takes for zone data changes to propagate to other copies of the zone, whether these copies are hosted as secondary zones or, if the zone is directory server-integrated, are other primary copies on the directory server. For zones that are directory server-integrated, the default value SHOULD be 0x0002A300 (2 days). Otherwise, the default SHOULD be 0x00000000. [<223>](#)

**RefreshInterval:** The refresh interval value, in hours, for the zone. The value 0x00000000 MUST be treated as a flag value for the value of "DefaultRefreshInterval" (section [3.1.1.1.1](#)).

**RFC5011KeyRollovers:** A Boolean indicating whether the zone follows [\[RFC5011\]](#) section 2 as key rollover takes place for signing key descriptors whose fIsKSK flag is set. The default value SHOULD be 0x00000000. [<224>](#)

**SecureDelegationPollingPeriod:** The interval, in seconds, between queries to refresh the set of delegation signer (DS) records in a secure delegation. The value MUST be limited to the range 0x00000E10 (1 hour) to 0x0x00093A80 (1 week), inclusive. The default value SHOULD [<225>](#) be 0x0000A8C0 (12 hours).

**SecureSecondaries:** The [DNS\\_ZONE\\_SECONDARY\\_SECURITY \(section 2.2.5.1.2\)](#) value for the zone. The value for this property is limited to those listed in the table in section [2.2.5.1.2](#).

**SignatureInceptionOffset:** The interval, in seconds, that the DNS server should subtract from the current time when generating the signature inception field in new RRSIG records ([\[RFC4034\]](#)). The value's range MUST be 0x00000000 to 0x00093A80, inclusive. The default value SHOULD [<226>](#) be 0x00000E10.



**SignWithNSEC3:** A Boolean indicating whether an online-signed zone should be signed using NSEC3 ([\[RFC5155\]](#)) for denial of existence. A zone not using NSEC3 will use NSEC ([\[RFC4034\]](#)). The default value SHOULD [<227>](#) be 0x00000001.

**Type:** The DNS\_ZONE\_TYPE (section [2.2.5.1.1](#)) value for the zone. This property is read-only.

The DNS server SHOULD support the following properties:

**Aging:** A Boolean indicating whether aging is enabled for the zone. [<228>](#)

**ForwarderSlave:** A Boolean indicating whether normal recursion SHOULD be used to resolve queries if the master servers for the forwarder zone are unreachable. [<229>](#)

**ForwarderTimeout:** The number of seconds the DNS server SHOULD [<230>](#) wait for response for a forwarded query.

**Unicode:** The server MUST ignore any value set for this Boolean property. [<231>](#)

**PluginEnabled:** A Boolean indicating whether the zone is configured to use a plugin. If a zone is configured to use a plugin, then incoming queries are forwarded to the plugin for calculation of the appropriate zone scope for query resolution. [<232>](#)

### 3.1.1.2.2 DNS Zone Address Array Properties

The following properties are IP address arrays (specified by type IDs DNSSRV\_TYPEID\_ADDRARRAY or DNSSRV\_TYPEID\_IPARRAY and formatted as DNS\_ADDR\_ARRAY (section [2.2.3.2.3](#)) or IP4\_ARRAY (section [2.2.3.2.1](#)), respectively). The DNS server SHOULD support both types. [<233>](#)

**AllowNSRecordsAutoCreation:** A list of IP addresses used to restrict automatic NS record creation for the zone. If the list is empty then NS record creation is not restricted. This setting is ignored if the zone is not stored in the directory server. The DNS server MUST NOT create an NS record for the fully qualified domain name of the local machine if one the machine's IP addresses is not present in this list.

**ScavengeServers:** A list of IP addresses of DNS servers authorized to perform scavenging of records in the zone.

The DNS Server SHOULD [<234>](#) also support the following properties:

**MasterServers:** A list of IP addresses of primary DNS servers for the zone. This value is required to be non-empty for any zone of a type that requires primary DNS servers: secondary, stub, or forwarder.

**LocalMasterServers:** A list of IP addresses of the zone's primary DNS servers used locally by this DNS server only. If not configured then the MasterServers value should be used, else this list should be used in place of the MasterServers value. This value is ignored if the zone type is not stub.

**NotifyServers:** A list of IP addresses of remote DNS servers to be notified for any changes to the zone. If empty, then no remote DNS servers will be notified when changes are made to this zone.

**SecondaryServers:** A list of IP addresses of authorized secondary DNS servers for the zone.

### 3.1.1.2.3 DNS Zone String Properties

The following properties are strings.

**DatabaseFile:** The name (with no path) of the zone file or NULL if the zone is not stored in a file.

**ApplicationDirectoryPartition:** The FQDN of the Application Directory Partition that the zone is stored in if "DsIntegrated" is TRUE, or NULL if the zone is not stored in a directory server.

The DNS server SHOULD [<235>](#) also support the following properties:

**BreakOnNameUpdate:** The DNS server will execute a debug break when the specified node name within the zone is the target of a DNS update. If this property is NULL or empty the DNS server will not execute debug breaks during DNS update processing.

**Keymaster:** The fully qualified domain name of the key master server for the zone. The default value MUST be NULL. [<236>](#)

**NSEC3UserSalt:** Salt that is to be used when hashed owner names are generated according to [\[RFC5155\]](#) section 5 in zones signed with NSEC3. This property MUST be ignored unless the NSEC3RandomSaltLength property is set to 0x00000000. [<237>](#)

**NSEC3CurrentSalt:** The current salt being used by a zone signed with NSEC3 to generate hashed owner names as described in [\[RFC5155\]](#) section 5. This property is read-only. [<238>](#)

#### 3.1.1.2.4 DNS Record Configuration Information

The list of names that are used in (name, value) pairs in DNS Zone metadata is given below. All properties below are 32-bit integers.

**Aging Time Stamp:** A time stamp value that specifies at what time this record was last updated. This property is initialized and written to the **whenChanged** attribute of its "dnsNode" object (section [2.3](#)).

#### 3.1.1.2.5 DNS Zone Scope Configuration Information

Zone scopes are a unique version of the zone. Multiple zone scopes can be configured inside the same zone. These zone scopes MAY contain different resource record sets from each other. [<239>](#) [<240>](#)

### 3.1.2 Timers

No timers are required beyond those used internally by RPC to implement resiliency to network outages, as specified in [\[MS-RPCE\]](#) section 3.2.3.2.1.

### 3.1.3 Initialization

At initialization time, the server MUST load the DNS Server Configuration (section [3.1.1](#)) from persistent local storage. The server MUST then initialize its zones:

If the server is configured to use a directory server:

- The server MUST invoke the task "Initialize an ADConnection", as defined in [\[MS-ADSO\]](#) section 6.2.6.1.1, with the following parameters:
  - **TaskInputTargetName:** NULL.
  - **TaskInputPortNumber:** 389.

- The server MUST store the new **TaskReturnADConnection** returned from the task as **DNS Server AD Connection**.
- If the **AD** connection is successfully initialized, the server MUST invoke the task "Setting an LDAP Option on an ADConnection", as defined in [\[MS-ADSO\]](#) section 6.2.6.1.2 on the Active Directory connection **DNS Server AD Connection**. Parameters for this task are as follows:
  - **TaskInputOptionName:** LDAP\_OPT\_AREC\_EXCLUSIVE.
  - **TaskInputOptionValue:** TRUE.
  - **TaskInputOptionName:** LDAP\_OPT\_PROTOCOL\_VERSION.
  - **TaskInputOptionValue:** 3.
  - **TaskInputOptionName:** LDAP\_OPT\_TIMELIMIT.
  - **TaskInputOptionValue:** 180.
  - **TaskInputOptionName:** LDAP\_OPT\_REFERRALS.
  - **TaskInputOptionValue:** FALSE.
- After the Active Directory connection is initialized and the option is set, the server MUST invoke the "Establishing an ADConnection" task, as specified in [\[MS-ADSO\]](#) section 6.2.6.1.3, with the **TaskInputADConnection** parameter set to **DNS Server AD Connection**.
- For the final step to complete the connection through LDAP to the local directory server, the server MUST invoke the "Performing an LDAP Bind on an ADConnection" task, as specified in [\[MS-ADSO\]](#) section 6.2.6.1.4, with the **TaskInputADConnection** parameter set to **DNS Server AD Connection**.
- If any of the previous steps returns an error, the server MUST retry the connection with LDAP up to eight times, unless the **Global Server State** changes to "Stopping", in which case it MUST discontinue initialization. If each of the eight attempts to connect with LDAP fails, the server MUST continue initialization.
  - If the connection with LDAP was successfully established:
    - The server MUST check that the "DnsAdmins" group already exists in the **Local security groups** (see section [3.1.1](#)). If it does not exist, and if the server is not a read-only server, then the server MUST create the "DnsAdmins" group in the **Local security groups**. The groupType attribute value for the "DnsAdmins" group MUST be 0x80000004.
  - If the server is not a read-only server, it MUST:
    - Attempt to add the "MicrosoftDNS" container object by invoking the "Performing an LDAP Operation on an ADConnection" task, as specified in [\[MS-ADSO\]](#) section 6.2.6.1.6, with the following parameters:
      - **TaskInputADConnection:** **DNS Server AD Connection**.
      - **TaskInputRequestMessage:** protocolOp is set to **addRequest** ([\[RFC4511\]](#) section 4.7).
      - The parameters of the **addRequest** are set as follows:
        - **entry:** "CN=MicrosoftDNS,CN=System,<Forest DN>"

- **attributes:**
  - **type:** "objectClass"; **vals:** "container"
  - **type:** "cn"; **vals:** "MicrosoftDNS"
- If the operation was successful, or if the operation failed because the object already existed and the "DnsAdmins" group was newly created in the last step, then the server MUST:
  - Attempt to grant all rights and ownership, with container inheritance, for the "MicrosoftDNS" (distinguished name: "CN=MicrosoftDNS,CN=System,<Forest DN>") object to the "DnsAdmins" group by following the procedure specified in section [3.1.6.4](#).
  - Attempt to grant all rights, with container inheritance, for the "MicrosoftDNS" (distinguished name: "CN=MicrosoftDNS,CN=System,<Forest DN>") object to the "Enterprise Domain Controllers" group by following the procedure specified in section [3.1.6.4](#).
  - Attempt to remove all rights for the "MicrosoftDNS" (distinguished name: "CN=MicrosoftDNS,CN=System,<Forest DN>") object from the "Authenticated Users" and "Built-In Administrators" groups by following the procedure specified in section [3.1.6.4](#).
- If the attempted addition of the "MicrosoftDNS" container object was successful, or if it failed because the object already existed, the server MUST:
  - Check that the **displayName** attribute of the object has been set, by invoking the "Performing an LDAP Operation on an ADConnection" task, as specified in [\[MS-ADSO\]](#) section 6.2.6.1.6, with the following parameters:
    - **TaskInputADConnection:** **DNS Server AD Connection.**
    - **TaskInputRequestMessage:** protocolOp is set to searchRequest ([\[RFC4511\]](#) section 4.5).
    - The parameters of the **searchRequest** are set as follows:
      - **baseObject:** "CN=MicrosoftDNS,CN=System,<Forest DN>"
      - **scope:** base (0)
      - **derefAliases:** neverDerefAliases (0)
      - **sizeLimit:** 0
      - **timeLimit:** 360
      - **typesOnly:** FALSE
      - **filter:** "(objectCategory=\*)"
      - **attributes:** **displayName**
  - If the search request was successful and the "MicrosoftDNS" container has no values for the **displayName** attribute, then modify the **displayName** attribute by invoking

the "Performing an LDAP Operation on an ADConnection" task, as specified in [\[MS-ADSO\]](#) section 6.2.6.1.6, with the following parameters:

- **TaskInputADConnection: DNS Server AD Connection.**
- **TaskInputRequestMessage:** protocolOp is set to modifyRequest ([\[RFC4511\]](#) section 4.6).
- The parameters of the **modifyRequest** are set as follows:
  - **object:** "CN =MicrosoftDNS,CN=System,<Forest DN>"
  - **changes:**
    - **operation:** replace
    - **type: displayDNS**
    - **vals:** "DNS Servers"
- The server MUST attempt to enumerate the application directory by invoking the "Performing an LDAP Operation on an ADConnection" task, as specified in [\[MS-ADSO\]](#) section 6.2.6.1.6, with the following parameters:
  - **TaskInputADConnection: DNS Server AD Connection.**
  - **TaskInputRequestMessage: protocolOp** is set to **searchRequest** ([\[RFC4511\]](#) section 4.5).
  - The parameters of the **searchRequest** are set as follows:
    - **baseObject:** "CN=Partitions,CN=Configuration,<Forest DN>"
    - **scope:** singleLevel (1)
    - **derefAliases:** neverDerefAliases (0)
    - **sizeLimit:** 0
    - **timeLimit:** 360
    - **typesOnly:** FALSE
    - **filter:** "(objectCategory=crossRef)"
    - **attributes:** "CN, ntSecurityDescriptor, instanceType, ms-DS-SDReferenceDomain, systemFlags, msDS-NC-Replica-Locations, ms-DS-NC-RO-Replica-Locations, nCName, dnsRoot, objectGUID, whenCreated, whenChanged, usnCreated, usnChanged, Enabled, objectClass"
- For each object found in the search, the server MUST use the configuration, replication, and security metadata values contained in the object to construct a structure of type DNS\_RPC\_DP\_INFO (see section [2.2.7.2.1](#)), computing the value of each field as specified in section [2.2.7.2.1](#), and the server MUST insert the structure as an entry in the Application Directory Partition Table. The server MUST create the in-memory **Application Directory Partition Access Control List** by copying the "ntSecurityDescriptor" attribute of the "crossRef" object. The server MUST also retrieve and store, in memory, the identity of the Domain Naming Master FSMO role owner. If

the default DNS Domain Partition or default DNS Forest Partition is not present during polling, the server MUST attempt to create and enlist in these partitions. If any LDAP operation fails, the server MUST continue initialization.

In all cases:

- The server MUST retrieve the list of zones to load from the source specified by the "BootMethod" setting's value (section [2.2.4.1.1](#)) and attempt to load zones from the configuration source specified by the "BootMethod" setting (section [3.1.1.1.1](#)).
  - If the method is BOOT\_METHOD\_UNINITIALIZED (section [2.2.4.1.1](#)):
    - If a zone loaded from the local directory server results in a zone with no nodes, the server MUST then attempt to load the same zone from file-based persistent storage.
  - If the method is BOOT\_METHOD\_DIRECTORY (section [2.2.4.1.1](#)):
    - If a zone loaded from the local directory server results in a zone with no nodes, the server MUST then attempt to load the same zone from file-based persistent storage.
    - If the LDAP connection to the directory server is unavailable, the server MUST attempt to load those zones specified in the persistent copy of the DNS Zone Table that are stored in local persistent storage.
    - If the LDAP connection to the directory server is available, the server MUST attempt to load the zones specified in the persistent copy of the DNS Zone Table, but only those zones stored in the Application Directory Partitions in which the server is enlisted. This MUST include at minimum the "defaultNamingContext" of the directory server's rootDSE, the default DNS Domain Partition and the default DNS Forest Partition. If the zone is stored in local persistent storage, the server MUST attempt to load the zone. If the zone is directory server-integrated, the server MUST attempt to load the LDAP "dnsZone" and "dnsNode" objects (section [2.3](#)) that represent the zone from the directory server. The DNS server MUST ignore any DNS node in the directory server which has the dnsTombstoned attribute set to "TRUE". If an attempt to load a zone fails for any reason, the server MUST clear the contents of the in-memory zone (if any) and mark the zone state as shutdown (see section [2.2.5.2.2](#)), but continue initialization.
    - If there are no root hints in the local directory server, but root hints were loadable from a file-based persistent storage and are non-empty, the server MUST write the root hints back to the local directory server through the WriteDirtyZones operation [3.1.4.1](#) by using the DNS\_ZONE\_LOAD\_OVERWRITE\_DS flag [2.2.5.2.7.1](#).
- The DNS Server Management Protocol server MUST register the RPC interface and begin listening on the RPC transports, as specified in section [2.1](#), and limited by the flags specified for the "RpcProtocol" property (section [3.1.1.1.1](#)).
- The server SHOULD invoke the **NetlogonControl2Ex** method with function code NETLOGON\_CONTROL\_FORCE\_DNS\_REG on the Netlogon protocol implementation on the local Domain Controller. [<241>](#) (See [\[MS-NRPC\]](#) section 3.5.4.9.1.)

### 3.1.4 Message Processing Events and Sequencing Rules

The server MUST indicate to the RPC runtime that it is to perform a strict **Network Data Representation (NDR)** data consistency check at target level 5.0, as specified in [\[MS-RPCE\]](#) Section [3](#).

Wherever this protocol requires an LDAP operation, if the operation is implemented as an asynchronous LDAP call, then the asynchronous result must be retrieved prior to returning from the DNS Server Management Protocol operation. The timeout for individual LDAP operations, other than the delete operation, MUST be 180 seconds, and the timeout for LDAP delete operations must be 1,440 seconds, unless otherwise stated. LDAP add and modify operations MUST be requested with an asynchronous LDAP method, and the server MUST retry an add or modify operation up to three times if an error prevents submission of the operation, but it MUST NOT retry if the asynchronous result of the operation is unsuccessful. All other operations MUST NOT be retried unless otherwise stated.

Wherever this protocol requires an addition, modification, or deletion of the information in the abstract data model (section 3.1.1), the prescribed operation MUST be performed only on the local in-memory portions of the abstract data model, unless otherwise specified.

Wherever this protocol requires a modification to a DNS Zone Table stored in-memory, the server MUST set the Dirty Flag to TRUE when the modification occurs. Wherever this protocol requires a DNS Zone Table stored in-memory to be written to persistent storage, the server MUST set the Dirty Flag to FALSE upon successful completion of the write.

**Opnums** 0 through 4 are deprecated. Clients SHOULD use opnums 5 through 9 instead and servers SHOULD support all methods in the table. <242>

This interface includes the following methods in RPC opnum order:

Methods in RPC Opnum Order

Method	Description
<a href="#">R_DnssrvOperation</a>	Invokes a specified set of server functions. This method is obsoleted by R_DnssrvOperation2. Opnum: 0
<a href="#">R_DnssrvQuery</a>	Issues type specific information queries to server. This method is obsoleted by R_DnssrvQuery2. Opnum: 1
<a href="#">R_DnssrvComplexOperation</a>	Invokes a specified set of server functions, which return complex structures. This method is obsoleted by R_DnssrvComplexOperation2. Opnum: 2
<a href="#">R_DnssrvEnumRecords</a>	Enumerates DNS records on the server. This method is obsoleted by R_DnssrvEnumRecords2. Opnum: 3
<a href="#">R_DnssrvUpdateRecord</a>	Adds/deletes/modifies DNS records. This method is obsoleted by R_DnssrvUpdateRecord2. Opnum: 4
<a href="#">R_DnssrvOperation2</a>	Invokes a specified set of server functions. Opnum: 5
<a href="#">R_DnssrvQuery2</a>	Issues type specific information queries to server. Opnum: 6
<a href="#">R_DnssrvComplexOperation2</a>	Invokes a specified set of server functions, which return complex

Method	Description
	structures. Opnum: 7
<a href="#">R_DnssrvEnumRecords2</a>	Enumerates DNS records on the server. Opnum: 8
<a href="#">R_DnssrvUpdateRecord2</a>	Adds/deletes/modifies DNS records. Opnum: 9
<a href="#">R_DnssrvUpdateRecord3</a>	Adds, deletes, or modifies DNS records to the zone or zone scope (if specified). Opnum: 10
<a href="#">R_DnssrvEnumRecords3</a>	Enumerates DNS records on the server in a particular zone or zone scope (if specified). Opnum: 11
<a href="#">R_DnssrvOperation3</a>	Invokes a specified set of server functions. Opnum: 12

### 3.1.4.1 R\_DnssrvOperation (Opnum 0)

The R\_DnssrvOperation method is used to invoke a set of server functions specified by pszOperation.

```

LONG R_DnssrvOperation(
    [in] handle_t hBindingHandle,
    [in, unique, string] LPCWSTR pwszServerName,
    [in, unique, string] LPCSTR pszZone,
    [in] DWORD dwContext,
    [in, unique, string] LPCSTR pszOperation,
    [in] DWORD dwTypeId,
    [in, switch_is(dwTypeId)] DNSSRV_RPC_UNION pData
);

```

**hBindingHandle:** An RPC binding handle to the server. Details concerning binding handles are specified in [\[C706\]](#) section 2.3.

**pwszServerName:** The client SHOULD pass a pointer to the fully qualified domain name of the target server as a null-terminated UTF-16LE character string. The server MUST ignore this value.

**pszZone:** A pointer to a null-terminated character string that contains the name of the zone to be queried. For operations specific to a particular zone, this string MUST contain the name of the zone in UTF-8 format or a **multizone operation string** (given in the table that follows) that indicates that the operation should be performed on multiple zones, but only if dwContext is zero. If dwContext is not zero, then the value of pszZone MUST be ignored. For all other operations this value MUST be set to NULL. When pszZone is NULL, the valid operations are in the first table under the pszOperation section that follows, or are a property name listed in section [3.1.1.1.2](#), [3.1.1.1.3](#), or [3.1.1.1.4](#). If this value is not NULL, then this value will be used by certain operations as specified in the second table for pszOperation that follows.



The following table shows what values may be specified to request that the operation be performed on multiple zones, using ZONE\_REQUEST\_FILTERS values (section [2.2.5.1.4](#)).

Value	Meaning
"..AllZones"	ZONE_REQUEST_PRIMARY   ZONE_REQUEST_SECONDARY   ZONE_REQUEST_AUTO   ZONE_REQUEST_FORWARD   ZONE_REQUEST_REVERSE   ZONE_REQUEST_FORWARDER   ZONE_REQUEST_STUB   ZONE_REQUEST_DS   ZONE_REQUEST_NON_DS   ZONE_REQUEST_DOMAIN_DP   ZONE_REQUEST_FOREST_DP   ZONE_REQUEST_CUSTOM_DP   ZONE_REQUEST_LEGACY_DP
"..AllZonesAndCache"	ZONE_REQUEST_PRIMARY   ZONE_REQUEST_SECONDARY   ZONE_REQUEST_CACHE   ZONE_REQUEST_AUTO   ZONE_REQUEST_FORWARD   ZONE_REQUEST_REVERSE   ZONE_REQUEST_FORWARDER   ZONE_REQUEST_STUB   ZONE_REQUEST_DS   ZONE_REQUEST_NON_DS   ZONE_REQUEST_DOMAIN_DP   ZONE_REQUEST_FOREST_DP   ZONE_REQUEST_CUSTOM_DP   ZONE_REQUEST_LEGACY_DP
"..AllPrimaryZones"	ZONE_REQUEST_PRIMARY
"..AllSecondaryZones"	ZONE_REQUEST_SECONDARY
"..AllForwardZones"	ZONE_REQUEST_FORWARD
"..AllReverseZones"	ZONE_REQUEST_REVERSE
"..AllDsZones"	ZONE_REQUEST_DS
"..AllNonDsZones"	ZONE_REQUEST_NON_DS
"..AllPrimaryReverseZones"	ZONE_REQUEST_REVERSE   ZONE_REQUEST_PRIMARY
"..AllPrimaryForwardZones"	ZONE_REQUEST_FORWARD   ZONE_REQUEST_PRIMARY
"..AllSecondaryReverseZones"	ZONE_REQUEST_REVERSE   ZONE_REQUEST_SECONDARY

Value	Meaning
"..AllSecondaryForwardZones"	ZONE_REQUEST_FORWARD   ZONE_REQUEST_SECONDARY

**dwContext:** A value used to specify **multizone operations** in **ZONE\_REQUEST\_FILTERS** (section 2.2.5.1.4) format or zero if the operation is not meant to apply to multiple zones. If pszZone is not NULL and matches the name of a zone hosted by the DNS server then the value of dwContext MUST be ignored.

**pszOperation:** A pointer to a null-terminated ASCII character string that contains the name of operation to be performed on the server. These are two sets of allowed values for pszOperation:

If pszZone is set to NULL, pszOperation MUST be either a property name listed in section [3.1.1.1.2](#), [3.1.1.1.3](#) or [3.1.1.1.4](#) , or one of the following.

Value	Meaning
"ResetDwordProperty"	Update the value of a (name, value) pair in DNS Server Configuration. On input, dwTypeId MUST be set to DNSSRV_TYPEID_NAME_AND_PARAM, and pData MUST point to a structure of type DNS_RPC_NAME_AND_PARAM (section <a href="#">2.2.1.2.5</a> ) that specifies the name of a property listed in section <a href="#">3.1.1.1.1</a> and a new value for that property.
"Restart"	The server SHOULD restart the DNS server process. dwTypeId and pData MUST be ignored by the server. <a href="#">&lt;243&gt;</a>
"ClearDebugLog"	Clear the debug log. dwTypeId and pData MUST be ignored by the server.
"ClearCache"	Delete all cached records from memory. dwTypeId and pData MUST be ignored by the server.
"WriteDirtyZones"	Write all zones that are stored in local persistent storage to local persistent storage if the zone's <b>Dirty Flag</b> (section <a href="#">3.1.1</a> ) is set to TRUE. dwTypeId and pData MUST be ignored by the server.
"ZoneCreate"	Create a zone. On input, dwTypeId SHOULD be set to DNSSRV_TYPEID_ZONE_CREATE. <a href="#">&lt;244&gt;</a> pData MUST point to a structure of one of the types specified in DNS DNS_RPC_ZONE_CREATE_INFO (section <a href="#">2.2.5.2.7</a> ) that contains all parameters of a new zone to be created by the DNS server, and pData MUST conform to the description corresponding to the value of dwTypeId (section <a href="#">2.2.1.1.1</a> ) If pData points to a DNS_ZONE_TYPE_CACHE or DNS_ZONE_TYPE_SECONDARY_CACHE record, the server MUST return a nonzero error. If pData points to a DNS_ZONE_TYPE_STUB, DNS_ZONE_TYPE_SECONDARY, or DNS_ZONE_TYPE_FORWARDER record, the server MAY return a nonzero error, but SHOULD return success. <a href="#">&lt;245&gt;</a>
"ClearStatistics"	Clears server statistics data on the DNS server. dwTypeId and pData MUST be ignored by the server.

Value	Meaning
"EnlistDirectoryPartition"	On input dwTypeId MUST be set to DNSSRV_TYPEID_ENLIST_DP, and the pData MUST point to a DNS_RPC_ENLIST_DP (section <a href="#">2.2.7.2.5</a> ) structure. This operation allows application directory partitions to be added to or deleted from the Application Directory Partition Table, and also allows the DNS server to be directed to add or remove itself from the replication scope of an existing application directory partition.
"StartScavenging"	Initiate a resource record scavenging cycle on the DNS server. dwTypeId, and pData MUST be ignored by the server.
"AbortScavenging"	Terminate a resource record scavenging cycle on the DNS server. dwTypeId and pData MUST be ignored by the server.
"AutoConfigure"	On input, dwTypeId SHOULD be set to DNSSRV_TYPEID_AUTOCONFIGURE, in which case pData MUST point to a structure of type DNS_RPC_AUTOCONFIGURE (section <a href="#">2.2.8.2.1</a> )<246>; dwTypeId MAY instead be set to DNSSRV_TYPEID_DWORD in which case pData MUST point to a DWORD in DNS_RPC_AUTOCONFIG (section <a href="#">2.2.8.1.1</a> ) format.
"ExportSettings"	Export DNS settings on the DNS server to a file on the DNS server. dwTypeId SHOULD be set to DNSSRV_TYPEID_LPWSTR, and pData MUST be ignored by the server.
"PrepareForDemotion"	Prepares for demotion by removing references to this DNS server from all zones stored in the directory server. dwTypeId and pData MUST be ignored by the server.
"PrepareForUninstall"	This operation does nothing on the DNS server. dwTypeId and pData MUST be ignored by the server.
"DeleteNode"	On input dwTypeId MUST be set to DNSSRV_TYPEID_NAME_AND_PARAM, and pData MUST point to a structure of type DNS_RPC_NAME_AND_PARAM (section <a href="#">2.2.1.2.5</a> ) that contains the FQDN of the node pointed to by pszNodeName in the DNS server's cache to be deleted and a Boolean flag in dwParam to indicate if node sub tree should be deleted.
"DeleteRecordSet"	On input dwTypeId MUST be set to DNSSRV_TYPEID_NAME_AND_PARAM, and pData MUST point to a structure of type DNS_RPC_NAME_AND_PARAM (section <a href="#">2.2.1.2.5</a> ). That structure contains the FQDN of the node to be deleted, which is cached on the DNS server, and the type of record set in the dwParam member, which indicates whether the entire set of this type is to be deleted. The type MUST be a DNS_RECORD_TYPE value (section <a href="#">2.2.2.1.1</a> ) or 0x00FF, which specifies all types.
"WriteBackFile"	Write all information for root hints back to persistent storage. dwTypeId and pData MUST be ignored by the

Value	Meaning
	server.
"ListenAddresses"	On input dwTypeId MUST be set to DNSSRV_TYPEID_IPARRAY or DNSSRV_TYPEID_ADDRARRAY and pData MUST point to a structure of type IP4_ARRAY (section <a href="#">2.2.3.2.1</a> ) or DNS_ADDR_ARRAY (section <a href="#">2.2.3.2.3</a> ) respectively, which contains a list of new IP addresses on which the DNS server should listen. The server SHOULD accept DNSSRV_TYPEID_ADDRARRAY and DNS_ADDR_ARRAY, and MAY accept DNSSRV_TYPEID_IPARRAY and IP4_ARRAY. <a href="#">&lt;247&gt;</a>
"Forwarders"	On input dwTypeId SHOULD be set to DNSSRV_TYPEID_FORWARDERS <a href="#">&lt;248&gt;</a> , and pData MUST point to a structure of one of the types specified in DNS_RPC_FORWARDERS (section <a href="#">2.2.5.2.10</a> ), which contains information about new IP addresses to which the DNS server should forward queries.
"LogFilePath"	On input dwTypeId MUST be set to DNSSRV_TYPEID_LPWSTR, and pData MUST point to a Unicode string that contains an absolute or relative pathname or filename for the debug log file on the DNS server.
"LogIPFilterList"	On input dwTypeId MUST be set to DNSSRV_TYPEID_IPARRAY or DNSSRV_TYPEID_ADDRARRAY, and pData MUST point to a structure of type IP4_ARRAY (section <a href="#">2.2.3.2.1</a> ) or DNS_ADDR_ARRAY (section <a href="#">2.2.3.2.3</a> ) respectively, which contains a list of new IP addresses used for debug log filter. The DNS server will write to the debug log only for traffic to/from these IP addresses. The server SHOULD accept DNSSRV_TYPEID_ADDRARRAY and DNS_ADDR_ARRAY, and MAY accept DNSSRV_TYPEID_IPARRAY and IP4_ARRAY. <a href="#">&lt;249&gt;</a>
"ForestDirectoryPartitionBaseName"	The DNS server MUST return an error.
"DomainDirectoryPartitionBaseName"	The DNS server MUST return an error.
"GlobalQueryBlockList"	Update the list of single-label names for which queries should be blocked. Query names that match this list, in any primary zone, will be blocked. On input dwTypeId MUST be set to DNSSRV_TYPEID_UTF8_STRING_LIST, and pData MUST point to a structure of type DNS_RPC_UTF8_STRING_LIST (section <a href="#">2.2.1.2.3</a> ).
"BreakOnReceiveFrom"	On input dwTypeId MUST be set to DNSSRV_TYPEID_IPARRAY or DNSSRV_TYPEID_ADDRARRAY and pData MUST point to a structure of type IP4_ARRAY (section <a href="#">2.2.3.2.1</a> ) or DNS_ADDR_ARRAY (section <a href="#">2.2.3.2.3</a> ) respectively, that contains a list of new IP addresses for which the DNS server will execute a breakpoint if a packet is received from these IP addresses. The server SHOULD accept

Value	Meaning
	DNSSRV_TYPEID_ADDRARRAY and DNS_ADDR_ARRAY, and MAY accept DNSSRV_TYPEID_IPARRAY and IP4_ARRAY. <250>
"BreakOnUpdateFrom"	On input dwTypeId MUST be set to DNSSRV_TYPEID_IPARRAY or DNSSRV_TYPEID_ADDRARRAY, and pData MUST point to a structure of type IP4_ARRAY (section 2.2.3.2.1) or DNS_ADDR_ARRAY (section 2.2.3.2.3) respectively, that contains a list of new IP addresses for which the DNS server will execute a breakpoint if an update is received from these IP addresses. The server SHOULD accept DNSSRV_TYPEID_ADDRARRAY and DNS_ADDR_ARRAY, and MAY accept DNSSRV_TYPEID_IPARRAY and IP4_ARRAY. <251>
"ServerLevelPluginDll"	On input <b>dwTypeId</b> MUST be set to DNSSRV_TYPEID_LPWSTR, and <b>pData</b> MUST point to a Unicode string that contains an absolute pathname for server side plug-in binary on the DNS server or an empty Unicode string.
"ActiveRefreshAllTrustPoints"	Schedules an immediate RFC 5011 <b>active refresh</b> for all trust points, regardless of the time of the last active refresh. The <i>dwTypeId</i> and <i>pData</i> parameters MUST be set to zero/NULL by the client and MUST be ignored by the server. <252>

If pszZone is not NULL, and pszOperation does not match a property name listed in sections 3.1.1.2.2 or 3.1.1.2.3, then pszOperation MUST be one of the following:

Value	Meaning
"ResetDwordProperty"	Update the value of a DNS Zone integer property. On input dwTypeId MUST be set to DNSSRV_TYPEID_NAME_AND_PARAM and pData MUST point to a structure of type DNS_RPC_NAME_AND_PARAM (section 2.2.1.2.5), which contains the name of a property listed in section 3.1.1.2.1 for the zone pointed to by pszZone and a new value for that property. <253>
"ZoneTypeReset"	Change the zone's type, for example to convert a secondary zone into a primary zone. On input dwTypeId SHOULD be set to DNSSRV_TYPEID_ZONE_CREATE, and pData SHOULD point to a structure of one of the types specified in DNS_RPC_ZONE_CREATE_INFO (section 2.2.5.2.7), which contains the new configuration information for the zone. dwTypeId MAY be set to DNSSRV_TYPEID_ZONE_CREATE_W2K or DNSSRV_TYPEID_ZONE_CREATE_DOTNET. <254> The server MUST return a nonzero error if the conversion is not implemented.
"PauseZone"	Pause activities for the zone pointed to by pszZone on the DNS server, and do not use this zone to answer queries or take updates until it is resumed. dwTypeId, and pData MUST be ignored by the server.

Value	Meaning
"ResumeZone"	Resume activities for the zone pointed to by pszZone on the DNS server; the zone thus becomes available to answer queries and take updates. dwTypeId and pData MUST be ignored by the server.
"DeleteZone"	Delete the zone pointed to by pszZone on the DNS server. dwTypeId and pData MUST be ignored by the server.
"ReloadZone"	Reloads data for the zone pointed to by pszZone on the DNS server from persistent storage. dwTypeId, and pData MUST be ignored by the server.
"RefreshZone"	Force a refresh of the secondary zone pointed to by pszZone on the DNS server, from primary zone server. For this operation pszZone MUST point to a secondary zone only. dwTypeId and pData MUST be ignored by the server.
"ExpireZone"	Force expiration of the secondary zone pointed to by pszZone on the DNS server, by invalidating the zone data locally and contacting primary to refresh. For this operation pszZone MUST point to a secondary zone only. dwTypeId, and pData MUST be ignored by the server.
"IncrementVersion"	Same as "WriteBackFile".
"WriteBackFile"	If the zone has uncommitted changes, write back all information for the zone pointed to by pszZone to persistent storage, and notify any secondary DNS servers. dwTypeId and pData MUST be ignored by the server.
"DeleteZoneFromDs"	Delete the zone pointed to by pszZone from the directory server. dwTypeId, and pData MUST be ignored by the server.
"UpdateZoneFromDs"	Refresh data for the zone pointed to by pszZone from the directory server. dwTypeId, and pData MUST be ignored by the server.
"ZoneExport"	Export zone data to a given file on the DNS server. On input dwTypeId MUST be set to DNSSRV_TYPEID_ZONE_EXPORT, and pData MUST point to a structure of type DNS_RPC_ZONE_EXPORT_INFO (section <a href="#">2.2.5.2.8</a> ) that contains a file name pointed to by pszZoneExportFile.
"ZoneChangeDirectoryPartition"	Move a zone to a given application directory partition. On input dwTypeId MUST be set to DNSSRV_TYPEID_ZONE_CHANGE_DP, and pData MUST point to structure of type DNS_RPC_ZONE_CHANGE_DP (section <a href="#">2.2.7.2.6</a> ), which contains the new application directory partition name pointed to by pszDestPartition.
"DeleteNode"	Delete a node. On input dwTypeId MUST be set to DNSSRV_TYPEID_NAME_AND_PARAM, and pData MUST point to a structure of type DNS_RPC_NAME_AND_PARAM (section <a href="#">2.2.1.2.5</a> ), which contains the FQDN of the node pointed to by pszNodeName present in the zone pointed to by pszZone on the DNS server to be deleted and a Boolean flag in dwParam to indicate if the node's sub-tree should be deleted.

Value	Meaning
"DeleteRecordSet"	Delete all the DNS records of a particular type at a particular node from the DNS server's cache. On input dwTypeId MUST be set to DNSSRV_TYPEID_NAME_AND_PARAM, and pData MUST point to a structure of type DNS_RPC_NAME_AND_PARAM (section <a href="#">2.2.1.2.5</a> ). That structure contains the FQDN of the node to be deleted and the DNS record type in the dwParam member. The type MUST be a DNS_RECORD_TYPE value (section <a href="#">2.2.2.1.1</a> ) or 0x00FF, which specifies all types.
"ForceAgingOnNode"	On input dwTypeId MUST be set to DNSSRV_TYPEID_NAME_AND_PARAM, and pData MUST point to a structure of type DNS_RPC_NAME_AND_PARAM (section <a href="#">2.2.1.2.5</a> ), which contains a node name in pszNodeName, and a Boolean flag in dwParam to indicate whether aging should be performed on all nodes in the sub-tree. All DNS records at the specified node in the zone named by pszZone will have their aging time stamp set to the current time. If sub-tree aging is specified by dwParam than all DNS records at all nodes that are children of this node will also have their aging time stamps set to the current time.
"DatabaseFile"	On input dwTypeId SHOULD be set to DNSSRV_TYPEID_ZONE_DATABASE< <a href="#">255</a> >, and pData MUST point to a structure of one of the types specified in DNS_RPC_ZONE_DATABASE (section <a href="#">2.2.5.2.6</a> ), which specifies whether the zone is directory server integrated by setting fDsIntegrated to TRUE, and if it is not then pszFileName MUST point to a Unicode string containing the absolute pathname of a file on the DNS server to which the zone database should be stored.
"MasterServers"	On input dwTypeId MUST be set to DNSSRV_TYPEID_IPARRAY or DNSSRV_TYPEID_ADDRARRAY, and pData MUST point to a structure of type IP4_ARRAY (section <a href="#">2.2.3.2.1</a> ) or DNS_ADDR_ARRAY (section <a href="#">2.2.3.2.3</a> ) respectively, which contains a list of IP addresses of new primary DNS servers for the zone pointed to by pszZone. This operation is valid only for secondary zones present on the server. The server SHOULD accept DNSSRV_TYPEID_ADDRARRAY and DNS_ADDR_ARRAY, and SHOULD accept DNSSRV_TYPEID_IPARRAY and IP4_ARRAY. If the input data of either type is accepted and the DNS server is directory-server integrated, the value of pData SHOULD be written to the directory server.< <a href="#">256</a> >< <a href="#">257</a> >
"LocalMasterServers"	On input dwTypeId MUST be set to DNSSRV_TYPEID_IPARRAY or DNSSRV_TYPEID_ADDRARRAY, and pData MUST point to a structure of type IP4_ARRAY (section <a href="#">2.2.3.2.1</a> ) or DNS_ADDR_ARRAY (section <a href="#">2.2.3.2.3</a> ) respectively, which contains a list of IP addresses of new local primary DNS servers for the zone pointed to by pszZone. This operation is valid only for stub zones present on the server, and if configured, this value overrides any primary DNS server configured in the directory server. The server SHOULD accept DNSSRV_TYPEID_ADDRARRAY and DNS_ADDR_ARRAY, and SHOULD accept DNSSRV_TYPEID_IPARRAY and IP4_ARRAY.< <a href="#">258</a> >

Value	Meaning
"SecondaryServers"	On input dwTypeId SHOULD <a href="#">&lt;259&gt;</a> be set to DNSSRV_TYPEID_ZONE_SECONDARIES, and pData MUST point to a structure of one of the types specified in DNS_RPC_ZONE_SECONDARIES (section <a href="#">2.2.5.2.5</a> ), which contains information about secondary DNS servers for the zone pointed to by pszZone.
"ScavengeServers"	On input dwTypeId MUST be set to DNSSRV_TYPEID_IPARRAY or DNSSRV_TYPEID_ADDRARRAY, and pData MUST point to a structure of type IP4_ARRAY (section <a href="#">2.2.3.2.1</a> ) or DNS_ADDR_ARRAY (section <a href="#">2.2.3.2.3</a> ) respectively, which contains a list of IP addresses of new servers that can run scavenging on the zone pointed to by pszZone. This operation is valid only for directory server integrated zones. The server SHOULD accept DNSSRV_TYPEID_ADDRARRAY, and DNS_ADDR_ARRAY, and SHOULD accept DNSSRV_TYPEID_IPARRAY and IP4_ARRAY. If the input data of either type is accepted and the DNS server is directory server-integrated, the value of pData SHOULD be written to the directory server. <a href="#">&lt;260&gt;&lt;261&gt;</a>
"AllowNSRecordsAutoCreation"	On input dwTypeId MUST be set to DNSSRV_TYPEID_IPARRAY or DNSSRV_TYPEID_ADDRARRAY and pData MUST point to a structure of type IP4_ARRAY (section <a href="#">2.2.3.2.1</a> ) or DNS_ADDR_ARRAY (section <a href="#">2.2.3.2.3</a> ) respectively, which contains a list of IP addresses of new servers that can auto-create NS records for the zone pointed to by pszZone. This operation is valid only for directory server integrated zones. The server SHOULD accept DNSSRV_TYPEID_ADDRARRAY and DNS_ADDR_ARRAY, and SHOULD accept DNSSRV_TYPEID_IPARRAY and IP4_ARRAY. If the input data of either type is accepted and the DNS server is directory server-integrated, the value of pData SHOULD be written to the directory server. <a href="#">&lt;262&gt;&lt;263&gt;</a>
"BreakOnNameUpdate"	On input dwTypeId MUST be set to DNSSRV_TYPEID_LPWSTR, and pData MUST point to a Unicode string that contains the FQDN of the node for which if an update is received the DNS server will execute a breakpoint.
"SignZone"	Sign a zone using DNSSEC, thus making the zone online-signed. The dwTypeId and pData parameters MUST be set to zero/NULL by the client and MUST be ignored by the server. <a href="#">&lt;264&gt;</a>
"UnsignZone"	Unsign a zone signed via online signing and remove all DNSSEC data from the zone. The dwTypeId and pData parameters MUST be set to zero/NULL by the client and MUST be ignored by the server. <a href="#">&lt;265&gt;</a>
"ResignZone"	Refreshes all DNSSEC data in an online-signed zone. The dwTypeId and pData parameters MUST be set to zero/NULL by the client and MUST be ignored by the server. <a href="#">&lt;266&gt;</a>
"PerformZoneKeyRollover"	Queues a signing key descriptor for key rollover. On input, dwTypeId MUST be set to DNSSRV_TYPEID_LPWSTR, and pData MUST point to a Unicode string representation of the GUID of the signing key descriptor to be queued for rollover. <a href="#">&lt;267&gt;</a>



Value	Meaning
"PokeZoneKeyRollover"	Instructs the DNS server to stop waiting for DS records <a href="#">[RFC4034]</a> in the parent zone to be updated and to proceed with key rollover as specified by <a href="#">[RFC4641]</a> . On input, dwTypeId MUST be set to DNSSRV_TYPEID_LPWSTR, and pData MUST point to a Unicode string representation of the GUID of the signing key descriptor to be queued for rollover. <a href="#">.&lt;268&gt;</a>
"RetrieveRootTrustAnchors"	Retrieves the root trust anchors from the XML file specified by the RootTrustAnchorsURL server property (section <a href="#">3.1.1.1.3</a> ) and adds any valid DS records to the root trust anchors. The dwTypeId and pData parameters MUST be set to zero/NULL by the client and MUST be ignored by the server. The pZone parameter MUST be set to "TrustAnchors". <a href="#">.&lt;269&gt;</a>
"TransferKeymasterRole"	Transfers the key master role to the current server. The dwTypeId parameter MUST be set to DNSSRV_TYPEID_DWORD, and pData must point to one of the values defined in the following paragraphs. <a href="#">.&lt;270&gt;</a>
"CreateZoneScope"	Creates a zone scope in the specified zone. The dwTypeId parameter MUST be set to DNSSRV_TYPEID_ZONE_SCOPE_CREATE. pData MUST point to a structure of the type <a href="#">DNS RPC ZONE SCOPE CREATE INFO V1 (section 2.2.13.2.2.1)</a> that contains all the parameters needed to create the zone scope. pszZone MUST be the name of the zone in which the zone scope is to be created. <a href="#">.&lt;271&gt;</a>
"DeleteZoneScope"	Deletes a zone scope from the specified zone. The dwTypeId MUST be set to DNSSRV_TYPEID_LPWSTR. pData MUST point to the name of the zone scope that is to be deleted. pszZone MUST be the name of the zone from which the zone scope is to be deleted. <a href="#">.&lt;272&gt;</a>

**dwTypeId:** A DNS\_RPC\_TYPEID (section [2.2.1.1.1](#)) value that indicates the type of input data pointed to by pData.

**pData:** Input data of type DNSSRV\_RPC\_UNION (section [2.2.1.2.6](#)), which contains a data structure as specified by dwTypeId.

**Return Values:** The method MUST return ERROR\_SUCCESS (0x00000000) on success or a nonzero Win32 error code value if an error occurred. All error values MUST be treated the same.

When processing this call, the server MUST do the following:

- If the Global Server State (section [3.1](#)) is not "Running", return a failure.
- Check that the input parameters conform to the syntax requirements above, and if not, return a failure.
- If pszZone is not NULL, search the DNS Zone Table (section [3.1](#)) record for the zone with a name matching the value of pszZone. If a matching zone cannot be found, search the list of multizone operation strings for a name matching the value of pszZone. If a matching name cannot be found, return a failure.

- Validate, as specified in section [3.1.6.1](#), that the client has permissions to perform the attempted operation. If pszZone is NULL then the DNS server MUST perform the Phase 2 authorization test using the DNS Server Configuration Access Control List. If pszZone is not NULL then the DNS server MUST perform the Phase 2 authorization test using the Zone Access Control List for the zone specified by pszZone. Write privilege MUST be tested for all operations with the following exceptions:
  - If pszOperation is "ZoneCreate" and if the zone will be created in the directory service, then Read privilege MUST be tested for.
  - If pszOperation is "DeleteZone" and if the zone is stored in the directory service, then Read privilege MUST be tested for.
  - If pszOperation is "EnlistDirectoryPartition" or "ExportSettings", then Read privilege MUST be tested for.
- If the client does not have permission to perform the operation, return a failure.
- If dwContext is not zero or pszZone matches a multizone operation string, then find all zone records in the DNS Zone Table matching the specified multizone filter value.
- If pszZone is NULL and dwContext is zero, execute the operation indicated by the value of pszOperation, as follows:
  - If pszOperation is "ResetDwordProperty", the server MUST verify that the property name matches a writable property name listed in section [3.1.1.1.1](#) for which the server supports the "ResetDwordProperty" operation and if not return an error. It SHOULD further verify that the value specified is either within the property's specified allowable range (including zero if it is specified that zero is allowed) and if not return an error. If the value is zero and zero is listed as a flag value for the default in section [3.1.1.1.1](#), the server SHOULD update the value of the property to be the default for the property and return success. The server MAY fail to verify these restrictions.<273> Otherwise, update the value of the property to be the new value specified and return success.<274> All properties are writable unless "this property is read-only" is specified in the property description in section [3.1.1.1.1](#).
  - If pszOperation matches a property name in section [3.1.1.1.2](#), [3.1.1.1.3](#), or [3.1.1.1.4](#) for which the server supports this value reset operation, the server MUST update the value of the property to be the new value specified in pData and return success. If pszOperation matches a property name that the server does not support, the server MUST simply return failure.<275>
  - If pszOperation is "Restart" the server MUST restart the DNS server, and return success.
  - During the restart operation, the DNS server MUST try to reload each zone scope (if present) in a zone, one by one from the database. If the DNS server fails to load any of the scopes of the zone, it MAY log an error event and pursue the next scope and subsequently next zone.<276> No error MUST be returned to the user in the event of failure.
  - If pszOperation is "ClearDebugLog" the server MUST copy the DNS log file specified by the DNS server's "LogFilePath" (section [3.1.1.1.3](#)) property to the implementation-specific backup directory, if the directory already exists, and overwrite an existing backup directory log file if needed. If the directory does not exist, the copy action MUST NOT be performed. Then, the server MUST delete the current contents of the DNS log file, and return success. The server MUST return success even if file operations on the DNS log file fail.<277>
  - If pszOperation is "ClearCache" the server MUST delete all records cached by the DNS server from memory. If the server is configured to use directory server, the server MUST search for the cache zone (with the LDAP search operation), create (LDAP add) the zone if it doesn't

exist, load (LDAP search) any default cache records into the local copy of the cache, and return success. If any of these LDAP operations fails, the server MUST return failure.

- If pszOperation is "WriteDirtyZones", the server MUST, for each primary zone or cache zone on the server:
  - Do nothing, if the zone's **Dirty Flag** (section [3.1.1](#)) is set to FALSE and the zone is not configured with any zone scopes or the zone is not stored in a file.
  - If the zone's **Dirty Flag** is set to TRUE, write the uncommitted information for the zone to the zone's file, send DNS notify [RFC1996](#) messages to all other servers hosting the zones, if they exist, and continue processing zones. If the zone specified is the cache zone, the server SHOULD write the root hints to their permanent storage. If the root hints are stored on the directory server and DownlevelDCsInDomain is nonzero, the server MUST check whether the root hints are empty. If the root hints are empty, the root hints MUST NOT be written to their permanent storage. Otherwise, if the root hints are stored on the directory server the server MUST use LDAP add, delete, and search operations to replace the root hints on the directory server. If there is a failure in writing the root hints records to the directory server, then the server MUST retry the write operation twice.
  - If the zone is configured with scopes, then for each scope the operation attempts to write the scope data to their respective scope database files, if the scope's **Dirty Flag** (section [3.1.1](#)) is set to TRUE. If any errors occur during this operation, then the operation SHOULD be continued with the next scope in the list and the appropriate error events SHOULD be logged.
  - When all zones have been processed, the server SHOULD return success, regardless of the success or failure of any processing operation. [<278>](#)
- If pszOperation is "ZoneCreate" the server MUST attempt to create a new zone entry in the DNS Zone Table using the parameters specified in pData, and return success or failure based on the result.
  - If the zone already exists, the server MUST return a failure.
  - If the zone to be created will use the directory server for persistent storage, the server MUST identify the correct application directory partition for the zone.
    - If dwFlags has the DNS\_ZONE\_CREATE\_FOR\_DCPROMO **bit** set, this partition MUST be the DNS domain application directory partition.
    - If dwFlags has the DNS\_ZONE\_CREATE\_FOR\_DCPROMO\_FOREST **bit** set, this partition MUST be the DNS forest application directory partition.
    - Otherwise, if the pszDpFqdn field is populated, this partition MUST be set to the value of that field.
    - Or else, this partition MUST be set to the directory partition that represents the default naming context. (See the description of the DNS\_DP\_LEGACY value in section [3.1.1.2.1](#)). [<279>](#)
  - The server MUST perform an LDAP search to verify the existence of this application directory partition, and return a failure if it does not exist, with the following exceptions:
    - If dwFlags is set to DNS\_ZONE\_CREATE\_FOR\_DCPROMO and the DNS domain application directory partition does not exist or is not available, the server MUST replace

the chosen partition with the directory partition that represents the default naming context.

- If dwFlags is set to DNS\_ZONE\_CREATE\_FOR\_DCPROMO\_FOREST and the DNS forest application directory partition does not exist or is not available, the server MUST replace the chosen partition with the directory partition that represents the default naming context.
- Once the existence of the chosen application directory partition has been verified, the server MUST then perform an LDAP search on that application directory partition to determine whether the zone already exists, and if so, return a failure.
- If the zone does not exist in the chosen application directory partition, the server MUST create a "dnsZone" object (section 2.3) for the zone and its default records, represented as "dnsNode" objects (section 2.3), in the chosen application directory partition using LDAP add operations and return success.
  - If dwFlags is set to DNS\_ZONE\_CREATE\_FOR\_DCPROMO, but the zone is being created in the directory partition representing the default naming context, the "DcPromo Flag" in the "dnsZone" object MUST be set to DCPROMO\_CONVERT\_DOMAIN. (See DcPromo Flag (section 2.3.2.1.2))
  - If dwFlags is set to DNS\_ZONE\_CREATE\_FOR\_DCPROMO\_FOREST, but the zone is being created in the directory partition representing the default naming context, the "DcPromo Flag" in the "dnsZone" object MUST be set to DCPROMO\_CONVERT\_FOREST. (See DcPromo Flag (section 2.3.2.1.2))
- If there is a failure in writing the records to the directory server, then the server MUST retry the write operation twice.
- If any of these LDAP operations cannot be completed, even after retries where specified, then the server MUST return failure.
- If pszOperation is "ClearStatistics" the server MUST clear internal server statistics, and return success.
- If pszOperation is "EnlistDirectoryPartition" the server SHOULD perform an application directory partition operation as specified by the contents of the input DNS\_RPC\_ENLIST\_DP (section 2.2.7.2.5) structure.<280> The procedures for these operations are described below:
  - If pszOperation is "EnlistDirectoryPartition" and the DNS\_DP\_OP\_CREATE\_FOREST or DNS\_DP\_OP\_CREATE\_DOMAIN operations are specified by the contents of the input **DNS\_RPC\_ENLIST\_DP** structure, the server SHOULD:
    - Check its local state to determine whether the partition specified by dwOperation is already present, and if so, the server MUST check its local state to determine whether it is not enlisted in the partition, and if so, connect to the directory server that is the FSMO role owner of the domain naming master FSMO role and perform an LDAP modify operation to add or remove the local server's name from the enlistment list (distinguished name "msDS-NC-Replica-Locations" **RODCs** use "msDS-NC-RO-Replica-Locations").<281>. If, based on the local state, the partition already exists and this server is enlisted or any of the above LDAP operations cannot be completed, then the server MUST return a failure.
    - Or, if based on the server's local state, the partition does not exist in the Application Directory Partition Table, create (but not enlist itself in) the partition object using LDAP

add commands and add the partition object to the Application Directory Partition Table, returning a failure if either of these operations fails to complete successfully.

- Poll the directory server for the partitions and enlistment status using LDAP search operations (to update the local state) and return success. If any LDAP operation cannot be completed, then the server **MUST** return a failure.
- If pszOperation is "EnlistDirectoryPartition" and the DNS\_DP\_OP\_ENLIST or DNS\_DP\_OP\_UNENLIST operations are specified by the contents of the input **DNS\_RPC\_ENLIST\_DP** structure, the server **SHOULD**:
  - Check whether the application directory partition specified is either the domain global partition or the forest global partition and whether the operation specified is not DNS\_DP\_OP\_ENLIST, and if the preceding conditions are true, the server **MUST** return a failure.
  - Check whether, instead, the operation specified is DNS\_DP\_OP\_ENLIST and the partition specified is the domain or forest global partition, and if so, the server **MUST** connect to the directory server that is the FSMO role owner of the Domain naming master FSMO role and perform an LDAP modify operation to add or remove the local server's name from the enlistment list (distinguished name "msDS-NC-Replica-Locations" (RODCs use "msDS-NC-RO-Replica-Locations" [<282>](#))) and return success.
  - Check whether the application directory partition specified is neither the domain global partition nor the forest global partition, and if so, the server **MUST**:
    - Poll the directory server for the partitions and enlistment status using LDAP search operations (to update the local Application Directory Partition Table), and then:
    - Check the local Application Directory Partition Table for the requested partition, and if the partition does not exist, return a failure.
    - Otherwise, the server **MUST** check whether the server is already enlisted and the operation is DNS\_DP\_OP\_ENLIST or the server is already unenlisted and the operation is DNS\_DO\_OP\_UNENLIST, and if so, return a failure.
    - Otherwise, the server **MUST** connect to the directory server that is the FSMO role owner of the Domain naming master FSMO role, and perform an LDAP modify operation to add or remove (for DNS\_DP\_OP\_ENLIST and DNS\_DP\_OP\_UNENLIST, respectively) the local server's name from the enlistment list (distinguished name "msDS-NC-Replica-Locations" (RODCs use "msDS-NC-RO-Replica-Locations" [<283>](#))), and return success.
  - If any of the above LDAP operations cannot be completed, then the server **MUST** return a failure.

If the zone was successfully loaded then the DNS server **MUST** set the zone's Shutdown flag to zero (section [2.2.5.2.2](#)).

- If pszOperation is "EnlistDirectoryPartition" and the DNS\_DP\_OP\_CREATE operation is specified by the contents of the input **DNS\_RPC\_ENLIST\_DP** structure, the server **SHOULD**:
  - Check whether the application directory partition specified is either the domain global partition or the forest global partition, and if so, perform the procedure described above for DNS\_DP\_OP\_CREATE\_DOMAIN or DNS\_DP\_OP\_CREATE\_FOREST, respectively.

- Otherwise, poll the directory server for the partitions and enlistment status using LDAP search operations (to update the local Application Directory Partition Table), and then check the local Application Directory Partition Table for the requested partition. If the partition already exists, the server MUST return a failure.
- Otherwise, the server MUST connect to the directory server and create the partition (an LDAP "domainDNS" object) using the LDAP add operation, poll the directory server for the partitions and enlistment status, update the local Application Directory Partition Table, attempt to create the "MicrosoftDNS" object using LDAP add operations, and return success.
- If any of the above LDAP operations other than the creation of the "MicrosoftDNS" object cannot be completed, then the server MUST return a failure.
- If pszOperation is "EnlistDirectoryPartition" and the DNS\_DP\_OP\_DELETE operation is specified by the contents of the input **DNS\_RPC\_ENLIST\_DP** structure, the server SHOULD:
  - Poll the directory server for the partitions and enlistment status using LDAP search operations (to update the local Application Directory Partition Table), and then check the local Application Directory Partition Table for the requested partition. If the partition does not exist, the server MUST return a failure.
  - Otherwise, the server MUST connect to the directory server that is the FSMO role owner of the Domain naming master FSMO role and perform an LDAP delete operation on the distinguished name of the crossRef object of the specified application directory partition, and if successful, again poll the directory server to update the local Application Directory Partition Table and return success.
  - If any of the above LDAP operations fails, then the server MUST return a failure. Any LDAP delete operation MUST have no client-side time limit. The server MUST NOT retry any failed LDAP operation.
- If pszOperation is "StartScavenging" the server MUST initiate a resource record scavenging cycle on the DNS server, and return success.
- If pszOperation is "AbortScavenging" the server MUST terminate a resource record scavenging cycle on the DNS server if one is currently in progress, and return success.
- If pszOperation is "AutoConfigure" the server SHOULD perform DNS server autoconfiguration as specified by the contents of the input DNS\_RPC\_AUTOCONFIGURE (section [2.2.8.2.1](#)) structure, and return success or failure based on the results of this operation. [<284>](#) To perform DNS server autoconfiguration, the server SHOULD:
  - Configure forwarders, if specified by the input flags and if the server does not currently have any forwarders configured. The list of forwarders is built by querying other servers that host the domain specified in the input arguments and by attempting to copy their forwarder list using the "ServerInfo" feature of the [R DnssrvQuery \(section 3.1.4.2\)](#) operation. If forwarders cannot be copied from another server, the list of forwarders is copied from the local machine's DNS client's list of DNS servers.
  - Configure root hints, if specified by the input flags. The list of root hints is built by querying the each DNS server on each local network adapter for the root DNS name.
  - Perform self-pointing, if either of the following is true:

- The DNS server is directory services-integrated and is the first DNS server in the directory services forest, and the DNS\_RPC\_AUTOCONFIG\_INTERNAL\_ZONES flag is set.
- Any of the DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT, DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT\_PREPEND, or DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT\_APPEND flags are set.
- To perform self-pointing, for each enabled network adapter and for each enabled IP stack (IPv4 or IPv6) on that adapter, the server SHOULD do one of the following:
  - If the DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT flag is set, replace the adapter's IP stack's DNS servers list with the loopback address.
  - If the DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT\_PREPEND flag is set and the loopback address is not already present, insert the loopback address at the start of the adapter's IP stack's DNS server list.
  - Otherwise, if the loopback address is not already present, insert the loopback address at the end of the adapter's IP stack's DNS server list.
  - Otherwise, make no change to the adapter's IP stack's DNS server list.
- Create the forest root domain and its "\_msdcs" subdomain, if specified by the input flags, and the directory service domain name of this DNS server matches the directory services forest root domain name, and the domains do not already exist. These two domains are created using LDAP add operations, followed by additional LDAP add operations to install the default records for those domains. If there is a failure in writing the records to the directory server, then the server MUST retry the write operation twice.

If any operation (including LDAP operations) fails, continue processing but record the fact that a failure occurred. At the end of processing, if the DNS\_RPC\_AUTOCONFIG\_INTERNAL\_RETURN\_ERRORS input flag is set, return an error; otherwise, return success.

- If pszOperation is "ExportSettings" the server SHOULD export DNS settings to a file on the DNS server, and return success.<285>
- If pszOperation is "PrepareForDemotion" the server SHOULD prepare the DNS server for demotion by removing references to this DNS server from all zones stored in the directory server, and return success.<286>
- If pszOperation is "PrepareForUninstall" the server SHOULD do nothing, and return success.<287>
- If pszOperation is "DeleteNode" the server MUST check whether the specified node is empty or does not currently exist and return ERROR\_SUCCESS if so. Otherwise it MUST delete all DNS records at the node pointed to by pszNodeName from the DNS server's cache. It MUST also delete all DNS records in the node's sub-tree if specified by the Boolean flag pointed to by dwParam field in pData, and return success. If the dwParam field in pData is set to FALSE and the node contains sub-trees, both the node and its sub-trees MUST NOT be deleted and a success status is returned. If the zone is directory server-integrated, the DNS server MUST set the node's DNS Node Tombstone State (section 3.1.1) to TRUE by setting the value of the dnsTombstoned attribute to "TRUE" and writing a DNS\_RPC\_RECORD\_TS (section 2.2.2.2.4.23) in the **dnsRecord** attribute.
- If pszOperation is "DeleteRecordSet" the server MUST check whether the specified node is empty or does not currently exist and return ERROR\_SUCCESS if so. Otherwise it MUST delete



all DNS records of the type specified by the dwParam field in pData from the node pointed to by pszNodeName in the DNS server's cache and return success. If this operation deletes the last record from the node and the zone is directory server-integrated, the DNS server MUST set the node's DNS Node Tombstone State (section [3.1.1](#)) to TRUE by setting the value of the dnsTombstoned attribute to "TRUE" and writing a DNS\_RPC\_RECORD\_TS (section [2.2.2.2.4.23](#)) in the **dnsRecord** attribute.

- If pszOperation is "WriteBackFile" the server SHOULD write the root hints to their permanent storage. If the root hints are stored on the directory server and DownlevelDCsInDomain is nonzero, the server MUST check whether the root hints are empty. If the root hints are empty, the root hints MUST NOT be written to their permanent storage. Otherwise, if the root hints are stored on the directory server, the server MUST use LDAP add, delete, and search operations to replace the root hints on the directory server and return success, regardless of the success or failure of these operations. If there is a failure in writing the root hints records to the directory server, then the server MUST retry the write operation twice, and still return success, even if the retries fail. [<288>](#)
- If pszOperation is "LogFilePath" the server MUST store the value passed in pData to be returned, unchanged, in future server information queries. Further, if pData is a NULL pointer or it points to an empty string, the server MUST replace pData's present value with the path to the default implementation specific log file. Finally, the server MUST attempt to create and/or open for write the file specified by the string. If the string is a filename or relative path, the server MUST attempt to create the file relative to the default implementation specific log file path. [<289>](#) If the file is opened successfully, then the server MUST commence logging to file and return success. Otherwise, it MUST disable logging to file and return a failure.
- If pszOperation is "ListenAddresses", the server MUST search the incoming array for loopback, multicast, or broadcast addresses, and if any are found, return a failure. Otherwise, the server MUST remove any addresses from the input that are not IPv4 addresses and create a backup copy of the current listen addresses. Then the server MUST attempt to listen on the network interfaces specified by the new listen addresses from the incoming array, and if this attempt fails, the server MUST restore the previous listen addresses and return a failure. Otherwise, the server MUST attempt to update the SOA records for its authoritative zones with the new listen addresses and MUST return success even if the attempt fails. If a zone is directory services-integrated, the server MUST use the LDAP search, add, and delete operations to update the SOA records and return success even if there were unsuccessful LDAP calls. If there is a failure in modifying the records on the directory server, then the server MUST retry the write operation twice, and still return success even if the retries fail.
- If pszOperation is "ServerLevelPluginDll", the server MUST store the value passed in pData and return SUCCESS, indicating only that the value was successfully received. The server MUST NOT validate the value passed, nor attempt to load the DLL, until the server is restarted. When the server restarts, if the value stored for ServerLevelPluginDll is not an empty string, the server MUST attempt to load the DLL specified. If the DLL fails to load for any reason, the server MUST fail to start. If the DLL has been loaded, then whenever the server is required to invoke the DLL query function, the server MUST invoke the query function of the DLL with a query name and type and add any resulting records to the server's cache. Whenever the server processes a query, if the DLL has been loaded, the server MUST invoke the query function of the DLL in the following conditions:
  - If a query cannot be answered with the information already present in the server's zone database and cache, then prior to recursing (if applicable), invoke the DLL query function and try again to answer the query from local data.



- If a response should have records in its additional section but no such records are in the server's cache or zone database, invoke the DLL query function and attempt again to find records for the additional section.
- If pszOperation is "ActiveRefreshAllTrustPoints", the server MUST schedule an RFC 5011 active refresh for all configured trust points (if any) and return success.

If pszZone is not NULL and is not a valid multizone operation string, then the server MUST return a failure if it does not contain a zone with the name matching the string pointed to by pszZone. If pszZone is not NULL or if the value of dwContext specifies a multizone operation, the server MUST execute the operation indicated by the value of pszOperation individually for each zone specified by the values of dwContext and pszZone. If a zone operation is performed with a filter in dwContext or a multizone operation string in pszZone, and no zones match the specified filter or satisfy the multi-zone operation string, the server MUST return ERROR\_SUCCESS (0x00000000). If the operation is executed for more than one zone, then for each matching zone, the specified operation MUST be executed with pszZone replaced with the name of the matching zone. If any of these multiple operations return an error, the server MUST return an error; otherwise, the server MUST return success. For any operation, if the specified zone is marked as "AutoCreated", the DNS server MUST return an error.

- If pszOperation is "ResetDwordProperty" the server MUST verify that the property name matches a property name listed in section [3.1.1.2.1](#) for which the server supports the "ResetDwordProperty" operation, and if not return an error. If the property is specified as "read-only" in section [3.1.1.2.1](#), the server MUST return an error. If the zone specified is a cache zone, the server MUST return an error. It MUST further verify that the value specified is within the property's allowable range (if specified) and if not return an error. When the property specified is Boolean and the value to be set is greater than 0x00000001, the server MUST replace the value with 0x00000001. If the property name is "AllowUpdate", the server MUST return an error when the zone specified is not Active Directory-integrated and the requested value is ZONE\_UPDATE\_SECURE, or when the zone specified is not a primary zone. If the property name is "AllowUpdate" and the value is not ZONE\_UPDATE\_OFF (section [2.2.6.1.1](#)), then the server SHOULD invoke the **NetlogonControl2Ex** method with function code **NETLOGON\_CONTROL\_FORCE\_DNS\_REG** on the "Netlogon" protocol implementation on the **local Domain Controller**. (See [\[MS-NRPC\]](#) section 3.5.4.9.1.)<290> If the property name is "Aging", the property value is TRUE, and the zone's Aging state is FALSE, then the server MUST reset the zone's Aging time by updating the dwAvailForScavengeTime (section [2.2.5.2.4.1](#)) value to the current time value, incremented by dwRefreshInterval (section [2.2.5.2.4.1](#)).<291> If the value is zero and zero is listed as a flag value for the default in section [3.1.1.2.1](#), the server MUST update the value of the property to be the default for that property, and return success. Otherwise, the server MUST update the value of the property for the zone to be the new value specified, and return success.<292>
- If pszOperation matches a property name listed in section [3.1.1.2.2](#) or section [3.1.1.2.3](#) for which the server supports this property reset operation, the server MUST update the value of the property for the zone to be the new value specified, and return success.<293> Otherwise, the server MUST return a nonzero error code.
- If pszOperation is "ZoneTypeReset" the server MUST:
  - Check whether the requested zone type, directory server integration, and (if applicable) application directory partition match the zone's present state, and if so, return success.
  - Check whether the zone is directory server integrated and currently in the process of loading, and if so, return a failure.

- Check whether the zone has scopes and whether the request is to convert the zone to either a non-primary zone or an AD integrated primary zone, and if so, return a failure.
- Verify, if the requested zone type is primary, that:
  - There is a complete copy of the zone on the server (that is, the zone is not a forwarder or stub zone).
  - If the fDSIntegrated flag is TRUE:
    - If the zone is a primary zone, that it is not empty.
    - If the zone is currently directory-server-integrated, it is already stored in the requested by pszDpFqdn, and is not a secondary or cache zone.
    - If the zone is currently not directory-server-integrated, it is either a cache zone or a primary zone.
  - If the fDSIntegrated flag is FALSE, that:
    - A copy of the zone is present in local persistent storage.
    - The zone is not shutdown, or empty.
  - If any of the verifications fails, return a failure.
- Check whether the requested zone type is secondary, and if so, reset the fDSIntegrated flag of the present zone to false.

Otherwise, the server MUST, in accordance with the operation specified:

- If the zone is directory server integrated, use LDAP search operations to find the zone's "dnsZone" and "dnsNode" objects (section [2.3](#)) and copy the appropriate zone properties and data to a file. Set the zone's "Zone GUID" property to NULL.
- If the zone is not directory server integrated, copy the zone's properties and data to the directory server using LDAP add operations (adding appropriate "dnsZone" and "dnsNode" objects to the directory server). If there is a failure in writing the zone records to the directory server, then the server MUST retry the write operation twice.
- Reset the type information to the requested type.
- Delete the original zone (using LDAP delete operations to remove the zone's "dnsZone" and "dnsNode" objects if necessary), and return success, or if any of the LDAP operations has failed, even after retries were specified, then return a failure. If deleting the zone from the directory server, the server MUST first attempt up to 4 times to rename the zone being deleted to a temporary name. Regardless of whether the rename is successful, the server then MUST attempt to delete the zone using LDAP delete operations, and retry up to 300 times if LDAP\_ADMIN\_LIMIT\_EXCEEDED is returned, never retrying if LDAP\_INSUFFICIENT\_RIGHTS is returned, and retrying up to 30 times for any other LDAP error.
- If pszOperation is "PauseZone" the server MUST set the zone's Paused flag to TRUE and return success. The same operation MUST be performed for all the existing zone scopes of the zone.
- If pszOperation is "ResumeZone" the server MUST set the zone's Paused flag to FALSE and return success. The same operation MUST be performed for all the existing zone scopes of the zone.

- If pszOperation is "DeleteZone" the server MUST delete the zone pointed to by pszZone from the server's local memory only and ensure that the zone will not be loaded at the next reboot, and return success. This operation does not modify the zone as it appears in the directory server, if it exists there.

The DNS server MUST also delete the associated scopes of that zone. The database file of the scopes MUST not be deleted.

- If pszOperation is "ReloadZone" the server MUST check whether the zone pointed to by pszZone is directory server-integrated and is already in the process of loading, and if so, return a failure. Otherwise, if the zone's **Dirty Flag** (section [3.1.1](#)) is set to TRUE, then the server MUST do the following:
  - If the zone is a cache zone, and the zone is directory server-integrated and there are root hints in the zone, the server SHOULD write root hints to their permanent storage and overwrite existing directory server root hints through the use of the DNS\_ZONE\_LOAD\_OVERWRITE\_DS flag (section [2.2.5.2.7.1](#)). If the root hints are stored on the directory server and DownlevelDCsInDomain is nonzero, the server MUST check whether the root hints are empty. If the root hints are empty, the root hints MUST NOT be written to their permanent storage. Otherwise, if the root hints are stored on the directory server, the server MUST search for and delete the root hints on the directory server using LDAP search and delete commands and write the new root hints to the cache zone on the directory server using LDAP add and modify commands. If there is a failure in writing the root hints records to the directory server, then the server MUST retry the write operation twice. [<294>](#)
  - Otherwise, if the zone is not a read-only zone, write a copy of the zone to a file and send a DNS notification to peer or secondary DNS servers, if any.

If this zone has scopes configured, then all the data of the scopes MUST be written one by one to the corresponding scope files, if the scope's **Dirty Flag** (section [3.1.1](#)) is set to TRUE. If writing to any of the scopes encounters any issue, an error event SHOULD be logged and the error code MUST be returned.

Then the server MUST do the following:

- If the zone is a secondary zone, check that it has been stored to a file, and if not, return a failure.
- If the zone is directory server-integrated, use the LDAP search operation to load the zone's "dnsZone" and "dnsNode" objects (section [2.3](#)) from the directory server into memory and return success.
- Otherwise, load the zone from the file, and return success. If the zone has scopes, then the DNS Server MUST try to load the scopes one by one. If any of the scope fails to load from the file, an event SHOULD be logged and the error MUST be returned.
- If any of the above LDAP operations fails, even after retries, where specified, then return a failure.
- If pszOperation is "RefreshZone" and the zone specified is a secondary zone, is not currently transferring from the primary server, and at least 15 seconds has elapsed since the **Time of Last SOA check** (section [3.1.1](#)) of the primary zone, then the server MUST force a refresh of the secondary zone pointed to by pszZone, from the primary DNS server, and return success. Otherwise, the server MUST return an error.

- If pszOperation is "ExpireZone" the server MUST force expiration of the secondary zone pointed to by pszZone, by invalidating the zone data locally and contacting the primary DNS server to refresh, and return success.
- If pszOperation is "IncrementVersion" the server MUST return a failure if the zone is not a primary or cache zone, and it MUST return success if the zone's **Dirty Flag** (section [3.1.1](#)) is set to FALSE or the zone is not stored in a file. Otherwise, the server MUST write the uncommitted information to the file, send DNS notify [\[RFC1996\]](#) messages to all other servers hosting the zones, if they exist, and return success. If the zone specified is the cache zone, the server SHOULD write the root hints to their permanent storage. If the root hints are stored on the directory server and DownlevelDCsInDomain is nonzero, the server MUST check whether the root hints are empty. If the root hints are empty, the root hints MUST NOT be written to their permanent storage. Otherwise, if the root hints are stored on the directory server, the server MUST use LDAP add, delete, and search operations to replace the root hints on the directory server, and return success, regardless of the success or failure of these operations. If there is a failure in writing the root hints records to the directory server, then the server MUST retry the write operation twice, and still return success even if the retries fail. [.<295>](#)
- If pszOperation is "WriteBackFile" the server MUST return a failure if the zone is not a primary or cache zone, and it MUST return success if the zone's **Dirty Flag** (section [3.1.1](#)) is set to FALSE and it does not have any scopes configured or the zone is not stored in a file. Otherwise, the server MUST write the uncommitted information to the file, send DNS notify [\[RFC1996\]](#) messages to all other servers hosting the zones, if they exist, and return success, regardless of the success or failure of these operations. If the zone has scopes, the data of the scopes MUST be written to their corresponding scope files if the scope's **Dirty Flag** (section [3.1.1](#)) is set to TRUE. If the server fails to write the data for any of the scopes it SHOULD log an error event and return the error code. If the operation results in updates only in the scope files and not the zone file, then DNS notify [\[RFC1996\]](#) messages SHOULD NOT be sent. If the zone specified is the cache zone, the server SHOULD write the root hints to their permanent storage. If the root hints are stored on the directory server and DownlevelDCsInDomain is nonzero, the server MUST check whether the root hints are empty. If the root hints are empty, the root hints MUST NOT be written to their permanent storage. Otherwise, if the root hints are stored on the directory server, the server MUST use LDAP add, delete, and search operations to replace the root hints on the directory server, and return success, regardless of the success or failure of these operations. If there is a failure in writing the root hints records to the directory server, then the server MUST retry the write operation twice, and still return success even if the retries fail. [.<296>](#)
- If pszOperation is "DeleteZoneFromDs", the server MUST leave the zone, represented by a "dnsZone" object (section [2.3](#)), intact and return a failure if the specified zone is not directory service-integrated. Otherwise, the server MUST locate the zone and its records, represented by "dnsNode" objects (section [2.3](#)) using LDAP search operations. The server MUST rename the dnsZone object by prepending "..Deleted-" (or "..Deleted.<random number>-" if "..Deleted-" already exists) to the zone's name, then attempt to delete the zone and its records using LDAP delete operations, and retry up to 300 times if LDAP\_ADMIN\_LIMIT\_EXCEEDED is returned, never retrying if LDAP\_INSUFFICIENT\_RIGHTS is returned, and retrying up to 30 times for any other LDAP error. If the deletion from the directory server was successful, then the server MUST delete the local memory copy of the zone, and return success. If any of these LDAP operations cannot be completed, even after retries where specified, then the server MUST return failure.
- If pszOperation is "UpdateZoneFromDs" the server MUST:

- Verify that the zone is not currently loading if it is a directory server integrated zone; otherwise, return a failure.
- Refresh data for the zone from the directory server, and return success or failure depending on the result of this operation.
- If pszOperation is "ZoneExport", the server SHOULD:
  - Verify that the zone is not currently loading if it is a directory server integrated zone; otherwise, return a failure.
  - Export zone data for the zone to a file on the DNS server specified by pszZoneExportFile in pData, and return success or failure depending on the result of this operation, but MAY<297> simply return a failure.
- If pszOperation is "ZoneChangeDirectoryPartition" the server MUST:
  - Verify that the specified zone is not currently loading; otherwise, return a failure.
  - Verify that the application directory partition specified by pszDstPartition in pData is already known to the server; otherwise, return a failure.
  - Verify that the specified zone is not already in the destination application directory partition; otherwise, return success.<298>
  - Create a backup copy of the zone properties (the "dnsZone" object) in local storage using LDAP search operations, and then create a temporary "dnsZone" object in the new application directory partition with the zone properties from the backup copy, using LDAP add operations. If a failure occurs, the server MUST delete the temporary zone using the LDAP search and delete operations, and return a failure.
  - Copy the records ("dnsNode" objects) of the old zone to the temporary zone by enumerating the old zone's records using an LDAP search operation and by writing the new records to the temporary zone using LDAP add operations. If there is a failure in writing the records to the directory server, then the server MUST retry the write operation twice. If the search fails or the new records cannot be written, then the server MUST attempt to delete the temporary zone using the LDAP search and delete operations, and retry up to 300 times if LDAP\_ADMIN\_LIMIT\_EXCEEDED is returned, never retrying if LDAP\_INSUFFICIENT\_RIGHTS is returned, and retrying up to 30 times for any other LDAP error, and return a failure.
  - Rename the temporary zone to the final zone name using the LDAP rename operation, and delete the original zone's "dnsZone" and "dnsNode" objects using the LDAP search and delete operations. If the LDAP rename operation fails, the server MUST attempt to delete the temporary zone, and retry up to 300 times if LDAP\_ADMIN\_LIMIT\_EXCEEDED is returned, never retrying if LDAP\_INSUFFICIENT\_RIGHTS is returned, and retrying up to 30 times for any other LDAP error and return a failure. Otherwise, the server MUST return a success.
- If pszOperation is "DeleteNode", then:
  - If the node does not exist, the server MUST return success.
  - Otherwise, if the node specified is not in either a primary zone or the cache zone or a zone scope of a primary file backed zone, or the node is the root node for a zone, the server MUST return failure.

- Otherwise, if the zone containing the specified node is not directory server-integrated, the server MUST delete the node pointed to by pszNodeName from the zone, MUST delete all DNS records in the node's sub-tree if the Boolean flag pointed to by the **dwParam** field in pData is set to TRUE, and return success.
- Otherwise,
  - If the Boolean flag pointed to by the **dwParam** field in pData is set to TRUE, the server MUST poll the directory server for zone changes, using LDAP search operations.
  - If the Boolean flag pointed to by the **dwParam** field in pData is set to FALSE and the node contains sub-trees, both the node and its sub-trees MUST NOT be deleted and a success status MUST be returned.
  - If the node to be deleted is not a cache node, then the server MUST locate the node's "dnsNode" object (section [2.3](#)) and its children using LDAP search operations, and then perform LDAP modify operations to set each node's dnsTombstoned attribute to "TRUE" and each node's dnsRecord (section [2.3.2.2](#)) attribute to contain a DNS\_RPC\_RECORD\_TS record (section [2.2.2.2.4.23](#)) with an EntombedTime value equal to the current time expressed as the number of 100-nanosecond intervals since 12:00 A.M. January 1, 1601 Coordinated Universal Time (UTC).
  - If any of these LDAP operations fails, the server MUST attempt to roll back the previous operations using LDAP modify operations and return failure. If, during an attempt to roll back the deletions, any LDAP operation fails, the server MUST retry up to 2 times, and return failure.
- If pszOperation is "DeleteRecordSet" and the node does not exist or the node exists but does not contain any records, represented as "dnsNode" objects (section [2.3](#)) of the type specified by the dwParam field in pData the server MUST return success. If the zone is directory server-integrated, the server MUST verify with an LDAP search that the node and/or records don't exist before returning success. Otherwise, the server MUST delete the record set of the type specified by **dwParam** field in pData (using LDAP modify operations when the zone is directory server-integrated), and return success. If deleting the record set would delete all records for the node and the zone is stored in the directory server, the DNS server MUST set the node's dnsTombstoned attribute to "TRUE" and each node's dnsRecord (section [2.3.2.2](#)) attribute to contain a DNS\_RPC\_RECORD\_TS record (section [2.2.2.2.4.23](#)) with an EntombedTime value equal to the current time expressed as the number of 100-nanosecond intervals since 12:00 A.M. January 1, 1601 Coordinated Universal Time (UTC). If any of these LDAP operations fails, the server MUST return failure.
- If pszOperation is "ForceAgingOnNode" and the zone's "Aging" (section [3.1.1.2.1](#)) property is TRUE and the node pointed to by pszNodeName exists, the server MUST enable aging on the node name pointed to by pszNodeName in the zone and MUST enable aging on the node's sub-tree if specified by the dwParam value in pData, and return success. Otherwise, the server MUST return a failure.
- If pszOperation is "AllowNSRecordsAutoCreation", the server MUST verify that the zone referenced is a primary zone and that it is a directory service-integrated zone and that it is not in the process of loading. If the verification fails, the server MUST return a failure. If the verification succeeds, the server MUST replace the list of IP addresses for which NS records will be automatically created with the list specified by pData, in the properties for the zone specified by pszZone, and use the LDAP modify operation to write all of the zone's properties to the copy of the zone on the directory server. Then, if any of the server's IP addresses are present in the new list of IP addresses and an NS record for the server is not present in the root of the zone, the server MUST add an NS record for the server to the zone, using the LDAP

add operation. If there is a failure in writing the record to the directory server, then the server MUST retry the write operation twice. If none of the server's IP addresses are present in the new list of IP addresses and an NS record for the server is present in the root of the zone, the server MUST delete the NS record for the server from the zone, using the LDAP delete operation. Finally, if any of these LDAP operations could not be completed, even after retries where specified, then the server MUST return a failure; otherwise, the server MUST return success.

- If pszOperation is "DatabaseFile", the server MUST:
  - Verify that the zone pointed to by pszZone exists; that if the fDsIntegrated field of the DNS\_RPC\_ZONE\_DATABASE structure is set to TRUE, the zone is directory services-integrated, and is not in the process of loading; and that if fDsIntegrated is set to FALSE, the zone is not directory services-integrated. If any of these verifications fail, the server MUST return a failure.
  - Check whether pszFilename is not NULL and fDsIntegrated is TRUE, and if so, return a failure.
  - Check whether pszFilename is NULL and fDsIntegrated is TRUE, and if so, return success.
  - Check whether pszFilename is NULL and fDsIntegrated is FALSE, and if so, create the default database file for the zone and configure the zone to use that file, and return success.
  - Check whether pszFilename is not NULL and fDsIntegrated is FALSE, and if so, create the database file specified by pszFilename for the zone and configure the zone to use that file, and return success, or return a failure if the file name could not be created as specified.
- If pszOperation is "SignZone", the server MUST:
  - Verify that the zone pointed to by pszZone exists, that the zone's IsSigned property is FALSE, that the zone is a Primary zone, and that the zone's Keymaster property is set to TRUE. If any of these verifications fails, the server MUST return a failure.
  - Check whether the EnableOnlineSigning server property is TRUE. If it is FALSE and the zone is directory services-integrated, the server MUST return a failure.
  - Add appropriate DNSSEC records to the zone according to the entries in the zone's **signing key descriptor list** and [\[RFC4035\]](#), and return failure if any operation fails.
  - If the zone has scopes, the server MUST initiate signing on the zone scopes. If any of the scope signing fails, the zone MUST be marked as unsigned, and an appropriate event SHOULD be logged and an error code MUST be returned.
  - Set the zone's IsSigned flag to TRUE, and return success.
- If pszOperation is "UnsignZone", the server MUST:
  - Verify that the zone pointed to by pszZone exists, that the zone's IsSigned property is TRUE, that the zone is a Primary zone, and that the zone's Keymaster property is set to TRUE. If any of these verifications fails, the server MUST return a failure.
  - Check whether the EnableOnlineSigning server property is TRUE. If it is FALSE and the zone is directory services-integrated, the server MUST return a failure.



- Remove all DNSSEC records of type DNSKEY, RRSIG, NSEC, and NSEC3 from the zone, and return failure if any operation fails.
- If the zone has scopes, the server MUST unsign all the scopes one by one. If any of the scopes fails to be unsigned the value of the zone IsSigned property MUST be kept unchanged, an appropriate event SHOULD be logged, and an error code MUST be returned.
- Set the zone's IsSigned flag to FALSE, and return success.
- If pszOperation is "ResignZone", the server MUST:
  - Verify that the zone pointed to by pszZone exists, that the zone's IsSigned property is TRUE, that the zone is a Primary zone, and that the zone's Keymaster property is set to TRUE. If any of these verifications fails, the server MUST return a failure.
  - Check whether the EnableOnlineSigning server property is TRUE. If it is FALSE and the zone is directory services-integrated, the server MUST return a failure.
  - Regenerate all DNSSEC records in the zone as done for the "SignZone" operation, and return failure if any operation fails.
  - If the zone has scopes, the server MUST start the re-sign operation on them. If any of the scopes fail to re-sign, the value of the zone IsSigned property MUST be kept unchanged, an appropriate event SHOULD be logged, and an error code MUST be returned.
  - Return success.
- If pszOperation is "PerformZoneKeyRollover", the server MUST:
  - Verify that the zone pointed to by pszZone exists, that the zone's IsSigned property is TRUE, that the zone is a Primary zone, and that the zone's Keymaster property is set to TRUE. If any of these verifications fails, the server MUST return a failure.
  - Find the signing key descriptor indicated by pData in the zone's signing key descriptor list. If the signing key descriptor is not in this list, the server MUST return a failure.
  - Put the specified signing key descriptor into the zone's **rollover queue**, and return success.
- If pszOperation is "PokeZoneKeyRollover", the server MUST:
  - Verify that the zone pointed to by pszZone exists, that the zone's IsSigned property is TRUE, that the zone is a Primary zone, and that that the zone's Keymaster property is set to TRUE. If any of these verifications fails, the server MUST return a failure.
  - Find the signing key descriptor indicated by pData in the zone's signing key descriptor list. If the signing key descriptor is not in this list, the server MUST return a failure.
  - If the dwCurrentRolloverStatus member of the DNS\_RPC\_SKD\_STATE structure for this signing key descriptor is not DNS\_SKD\_STATUS\_KSK\_WAITING\_FOR\_DS\_UPDATE, the server MUST return a failure.
  - Otherwise, the DNS server MUST advance the dwCurrentRolloverStatus member of the DNS\_RPC\_SKD\_STATE structure for this signing key descriptor to DNS\_SKD\_STATUS\_KSK\_WAITING\_FOR\_DS\_TTL and proceed with key rollover for this signing key descriptor as specified in [\[RFC4641\]](#), and return success.
- If pszOperation is "RetrieveRootTrustAnchors", the server MUST:



- Verify that:
  - The zone pointed to by pszZone is the Trust Anchors zone.
  - The server is not a read-only server.
  - The "EnableDnsSec" server property (section [3.1.1.1.1](#)) is TRUE.
  - The "RootTrustAnchorsURL" property (section [3.1.1.1.3](#)) is not NULL, that the scheme is "https", and that the host and url-path are nonempty (see [RFC1738](#)).
- If any of these verifications fails, return a failure. Otherwise, the server MUST:
  - Create the Trust Anchors zone if it does not exist. If the creation fails, return an error.
  - Open an HTTPS connection to the host and port defined by the "RootTrustAnchorsURL" property. If the connection fails or times out, return an error.
  - Retrieve the XML document specified by the url-path or return an error.
  - Parse the XML document strictly according to the schema defined in [draft-jabley-dnssec-trust-anchor-03]. If the XML document deviates from the schema, return an error.
  - For each DS trust anchor specified by the XML document, verify that the algorithm and digest type are supported by the server, and that the length of the digest is valid. If the DS trust anchor is unsupported, continue to the next DS trust anchor. If none of the trust anchors are supported, return an error. If the trust anchor is supported and the digest length is invalid, return an error. Otherwise, add the trust anchor to a list of pending trust anchors.
  - Add each trust anchor in the pending list of trust anchors to the root node of the Trust Anchors zone. If the add attempt fails, return an error.
  - Schedule an immediate RFC 5011 active refresh for all trust points, and return success.
- If pszOperation is "TransferKeymasterRole", the server MUST:
  - Verify that dwTypeId is DNSSRV\_TYPEID\_DWORD and that pData points to one of the following values. If any of these verifications fails, the server MUST return a failure.

Value	Meaning
DNS_ZONE_TRANSFER_KEYMASTER_ROLE 0x00000001	Assign the key master role to the current server only if the zone is currently signed and the current server has access to the private key material corresponding to the signing key descriptors for the zone.
DNS_ZONE_SEIZE_KEYMASTER_ROLE 0x00000002	Force assignment of the key master role to the current server, allowing assignment on unsigned zones or zones with no

Value	Meaning
	signing key descriptors. If the zone has signing key descriptors, this operation MUST return a failure if private key material for all signing key descriptors is not accessible by the current server.
DNS_ZONE_POLL_KEYMASTER_ROLE 0x00000003	Requests that the server refresh the key master zone flags by reading from the directory service.
DNS_ZONE_SEIZE_KEYMASTER_ROLE_WITHOUT_KEY_CHECKS 0x00000004	Force assignment of the key master role to the current server, without checking to see whether private key material is accessible by the current server.

- Verify that the zone pointed to by pszZone exists, that the zone is a Primary zone that is directory services-integrated, and that the zone is not read-only. If any of these verifications fails, the server MUST return a failure.
- If the value of pData is DNS\_ZONE\_TRANSFER\_KEYMASTER\_ROLE or DNS\_ZONE\_SEIZE\_KEYMASTER\_ROLE or DNS\_ZONE\_SEIZE\_KEYMASTER\_ROLE\_WITHOUT\_KEY\_CHECKS, the server MUST:
  - Verify that the "IsSigned" property of the zone (section [3.1.1.2.1](#)) is TRUE or that the value of pData is DNS\_ZONE\_SEIZE\_KEYMASTER\_ROLE. If not, return a failure.
  - For an AD-backed zone, refresh the list of key master servers from the directory service or, if this operation fails, return a failure.
  - Return success if the "IsKeymaster" property of the zone (section [3.1.1.2.1](#)) is TRUE and the value of pData is DNS\_ZONE\_TRANSFER\_KEYMASTER\_ROLE.
  - For an AD-backed zone, check whether the value of pData is DNS\_ZONE\_TRANSFER\_KEYMASTER\_ROLE or DNS\_ZONE\_SEIZE\_KEYMASTER\_ROLE and the zone has one or more signing key descriptors and, if so, verify that the private key material is available for each of the signing key descriptors.
  - For a file-backed zone, check whether the value of pData is DNS\_ZONE\_SEIZE\_KEYMASTER\_ROLE and the zone has one or more signing key descriptors and, if so, verify that the private key material is available for each of the signing key descriptors.
  - For an AD-backed zone, query the directory service for the value of the current key master. If this operation fails, and the value of pData is DNS\_ZONE\_TRANSFER\_KEYMASTER\_ROLE, return a failure.
  - For an AD-backed zone, check whether the value of pData is DNS\_ZONE\_TRANSFER\_KEYMASTER\_ROLE and, if so, verify that the query in the preceding step returned a value for the current key master, and that the current key

master is marked as a key master by invoking the [R\\_DnssrvQuery2](#) operation (section [3.1.4.7](#)) on the current key master. The `pszZone` value used MUST match the value of `pszZone` passed to this operation, and `pszOperation` MUST be set to "IsKeymaster". If the current key master returns an error or FALSE, return a failure.

- For an AD-backed zone, refresh the zone properties from the directory service. If this operation fails, the server MUST ignore the failure.
- For an AD-backed zone, perform an LDAP modify operation to add the DN of the zone to the "msDNS-KeymasterZones" attribute of the server settings object ("CN=DNS Settings,CN=<server\_name>,CN=Servers,CN=<site\_name>,CN=Sites,CN=Configuration,DC=<domain name>") or, if this operation fails, return a failure.
- For an AD-backed zone, perform an LDAP delete operation to delete the DN of the zone from the "msDNS-KeymasterZones" attribute of the previous key master server settings object ("CN=DNS Settings,CN=<previous\_keymaster\_server\_name>,CN=Servers,CN=<site\_name>,CN=Sites,CN=Configuration,DC=<domain name>") or, if this operation fails, return a failure.
- For an AD-backed zone, check whether the value of `pData` is `DNS_ZONE_TRANSFER_KEYMASTER_ROLE` and, if so, perform the `TransferKeymasterRole` operation on the previous key master, with all parameters matching those passed to the current operation except that `pData` MUST point to `DNS_ZONE_POLL_KEYMASTER_ROLE`. If this operation fails, the server MUST ignore the failure.
- Return success.
- If the value of `pData` is `DNS_ZONE_POLL_KEYMASTER_ROLE`, the server MUST:
  - Open an LDAP connection to the Domain Naming Master FSMO role holder or to the local directory service if an LDAP connection to the Domain Naming Master FSMO role holder cannot be opened.
  - Perform an LDAP search operation to retrieve the value of the "msDNS-KeymasterZones" attribute of the DNS server settings object ("CN=DNS Settings,CN=<server\_name>,CN=Servers,CN=<site\_name>,CN=Sites,CN=Configuration,DC=<domain name>") and, if the operation fails, return a failure.
  - The DNS server MUST iterate each zone in the DNS Zone Table (section [3.1.1.2](#)) and for each zone:
    - If the zone is not directory service-integrated, ignore it.
    - Otherwise, if the zone is present in the "msDNS-KeymasterZones" attribute and the zone's `IsKeymaster` property (see section [3.1.1.2.1](#)) is not TRUE or if the zone is not present in the "msDNS-KeymasterZones" attribute and the zone's `IsKeymaster` property is TRUE:
      - Set the zone's `IsKeymaster` property (section [3.1.1.2.1](#)) to TRUE if the zone is present in the "msDNS-KeymasterZones" attribute or to FALSE if the zone is not present in the "msDNS-KeymasterZones" attribute.

- Retrieve the fully qualified domain name of the zone's current key master: if the zone's IsKeymaster property is TRUE, set the zone's Keymaster property to the fully qualified domain name of the local host; otherwise, perform an LDAP search using the search filter "(&(msDNS-KeymasterZones=<zone name>)(objectCategory=msDNS-ServerSettings))" and set the zone's Keymaster property to the dnsHostName attribute value found on the parent object of the object returned by this search.
- If pszOperation is "CreateZonescope", the server MUST:
  - Verify that the zone pointed to by pszZone exists, that the zone is a primary file backed zone, that it is not an autogenerated zone, that it is a non-reverse lookup zone, and that a scope does not exist with the given scope name in the specified zone. If any of these verifications fails, the server MUST return a failure.
  - Create the zone scope with the given scope name and return success if the creation is successful, otherwise return a failure.
- If pszOperation is "DeleteZonescope", the server MUST:
  - Verify that the zone pointed to by pszZone exists, that the zone is a primary file backed zone, that it is not an autogenerated zone, that it is a non-reverse lookup zone, and that a scope does exist with the given scope name in the specified zone. If any of these verifications fails, the server MUST return a failure.
  - Return success if the deletion of the zone scope is successful, otherwise return a failure.

### 3.1.4.2 R\_DnssrvQuery (Opnum 1)

The R\_DnssrvQuery method queries the DNS server for information. The type of information queried for is specified by the client using the pszZone and pszOperation parameters. For the purpose of selecting an output structure type the server MUST consider the value of dwClientVersion (section [2.2.1.2.1](#)) to be 0x00000000 when responding to this method.

```
LONG R_DnssrvQuery(
    [in] handle_t hBindingHandle,
    [in, unique, string] LPCWSTR pwszServerName,
    [in, unique, string] LPCSTR pszZone,
    [in, unique, string] LPCSTR pszOperation,
    [out] PDWORD pdwTypeId,
    [out, switch_is(*pdwTypeId)] DNSSRV_RPC_UNION* ppData
);
```

**hBindingHandle:** An RPC binding handle to the server. Details concerning binding handles are specified in [\[C706\]](#) section 2.3.

**pwszServerName:** The client SHOULD pass a pointer to the fully qualified domain name of the target server as a null-terminated UTF-16LE character string. The server MUST ignore this value.

**pszZone:** A pointer to a null-terminated character string that contains name of the zone to be queried. For operations specific to a particular zone, this field MUST contain the name of the zone in UTF-8 format. For all other operations, this field MUST be NULL.

**pszOperation:** A pointer to a null-terminated character string that contains the name of the operation to be performed on the server. These are two sets of allowed values for pszOperation:

If pszZone is set to NULL, pszOperation MUST be either a property name listed in section [3.1.1.1](#), or the following.

Value	Meaning
"ServerInfo"	On output pdwTypeId SHOULD be set according to the value of the dwClientVersion field (section <a href="#">2.2.1.2.1</a> ). If dwClientVersion is 0x00000000, then pdwTypeId SHOULD be set to DNSSRV_TYPEID_SERVER_INFO_W2K. If dwClientVersion is 0x00060000, then pdwTypeId SHOULD be set to DNSSRV_TYPEID_SERVER_INFO_DOTNET. If dwClientVersion is 0x00070000, then pdwTypeId SHOULD be set to DNSSRV_TYPEID_SERVER_INFO< <a href="#">299</a> > ppData MUST point to a structure of one of the types specified in DNS_RPC_SERVER_INFO (section <a href="#">2.2.4.2.2</a> ), which SHOULD contain the configuration information for the DNS server, but MAY have some fields set to zero even when the related configuration value is nonzero.< <a href="#">300</a> >

If pszZone is not NULL, pszOperation MUST be either a property name listed in section [3.1.1.2](#), or one of the following.

Value	Meaning
"Zone"	On output the value pointed to by pdwTypeId SHOULD be set to DNSSRV_TYPEID_ZONE< <a href="#">301</a> > and ppData MUST point to a structure of one of the types specified in DNS_RPC_ZONE (section <a href="#">2.2.5.2.1</a> ), which contains abbreviated information about the zone pointed to by pszZone.
"ZoneInfo"	On output the value pointed to by pdwTypeId SHOULD be set to DNSSRV_TYPEID_ZONE_INFO< <a href="#">302</a> > and ppData MUST point to a structure of one of the types specified in DNS_RPC_ZONE_INFO (section <a href="#">2.2.5.2.4</a> ), which contains full information about the zone pointed to by pszZone.

**pdwTypeId:** A pointer to an integer that on success contains a value of type DNS\_RPC\_TYPEID (section [2.2.1.1.1](#)) that indicates the type of data pointed to by ppData.

**ppData:** A DNSSRV\_RPC\_UNION(section [2.2.1.2.6](#)) that contains a data-structure as indicated by dwTypeId.

**Return Values:** A Win32 error code indicating whether the operation completed successfully (0x00000000) or failed (any other value).

When processing this call, the server MUST do the following:

- If the Global Server State (section [3.1](#)) is not "Running", return a failure.
- Check that the input parameters conform to the syntax requirements above, and if not return a failure.
- If pszZone is not NULL, search the DNS Zone Table (section [3.1](#)) for the zone with name matching the value of pszZone. If a matching zone cannot be found return a failure.
- Validate, as specified in section [3.1.6.1](#), that the client has permissions to perform the attempted operation. If pszZone is NULL then the DNS server MUST perform the Phase 2 authorization test using the DNS Server Configuration Access Control List. If pszZone is not NULL then the DNS

server MUST perform the Phase 2 authorization test using the Zone Access Control List for the zone specified by pszZone. Read privilege MUST be tested for all operations. If the client does not have permission to perform the operation, the server MUST return a failure.

- If pszZone is NULL, execute the operation indicated by the value of pszOperation specified as follows:
  - If pszOperation is "ServerInfo" the server MUST return in ppData configuration information for the DNS server, and return success.
  - If pszOperation matches a property name listed in section [3.1.1.1.1](#), the server SHOULD<303> return DNSSRV\_TYPEID\_DWORD in pdwTypeId and return in ppData the value associated with that property, and return success. The server MAY return a nonzero error code.
  - If pszOperation matches a property name listed in section [3.1.1.1.2](#), the server SHOULD<304> return DNSSRV\_TYPEID\_ADDRARRAY in pdwTypeId, return in ppData the value associated with that property as a DNS\_ADDR\_ARRAY (section [2.2.3.2.1](#)), and return success; it MAY instead return DNSSRV\_TYPEID\_ADDRARRAY in the pdwTypeId, return in ppData the value associated with that property as an IP4\_ARRAY (section [2.2.3.2.1](#)), and return success; or it MAY instead return a nonzero error code.<305><306>
  - If pszOperation matches a property name listed in section [3.1.1.1.3](#) that the server supports, the server MUST return DNSSRV\_TYPEID\_LPSTR for UTF-8 string properties or DNSSRV\_TYPEID\_LPWSTR for Unicode string properties in pdwTypeId, and return in ppData a pointer to the UTF-8 or Unicode string associated with that property, and return success.<307> If the property name is not supported, the server MUST return a nonzero error code.
  - If pszOperation matches a property name listed in section [3.1.1.1.4](#) that the server supports, the server MUST return DNSSRV\_TYPEID\_UTF8\_STRING\_LIST, and return in ppData the DNS\_RPC\_UTF8\_STRING\_LIST (section [2.2.1.2.3](#)) associated with that property, and return success.
- If pszZone is not NULL, execute the operation indicated by the value of pszOperation, specified as follows:
  - If pszOperation is "Zone", the server SHOULD instead return information about the zone in DNS\_RPC\_ZONE (section [2.2.5.2.1](#)) format in ppData, and return success.
  - If pszOperation is "ZoneInfo", the server SHOULD instead return information about the zone in DNS\_RPC\_ZONE\_INFO (section [2.2.5.2.4](#)) format in ppData, and return success.
  - If pszOperation matches a property name listed in section [3.1.1.2.1](#), the server MUST return DNSSRV\_TYPEID\_DWORD in pdwTypeId, return in ppData the value associated with that property, and return success.
  - If pszOperation matches a property name listed in section [3.1.1.2.2](#) that the server supports, the server SHOULD<308> return DNSSRV\_TYPEID\_ADDRARRAY in pdwTypeId, return in ppData the value associated with that property as a DNS\_ADDR\_ARRAY (section [2.2.3.2.3](#)), and return success; or it MAY instead return DNSSRV\_TYPEID\_IPARRAY in pdwTypeId, return in ppData the value associated with that property as an IP4\_ARRAY (section [2.2.3.1.1](#)), and return success. If the property name is not supported, the server MUST return a nonzero error code.

### 3.1.4.3 R\_DnssrvComplexOperation (Opnum 2)

The R\_DnssrvComplexOperation method is used to invoke a set of server functions specified by the caller. These functions generally return more complex structures than simple 32-bit integer values, unlike the operations accessible through R\_DnssrvOperation. For the purpose of selecting an output structure type the server MUST consider the value of dwClientVersion (section 2.2.1.2.1) to be 0x00000000 when responding to this method.

```
LONG R_DnssrvComplexOperation(  
    [in] handle_t hBindingHandle,  
    [in, unique, string] LPCWSTR pwszServerName,  
    [in, unique, string] LPCSTR pszZone,  
    [in, unique, string] LPCSTR pszOperation,  
    [in] DWORD dwTypeIn,  
    [in, switch_is(dwTypeIn)] DNSSRV_RPC_UNION pDataIn,  
    [out] PDWORD pdwTypeOut,  
    [out, switch_is(*pdwTypeOut)] DNSSRV_RPC_UNION* ppDataOut  
);
```

**hBindingHandle:** An RPC binding handle to the server. Details concerning binding handles are specified in [\[C706\]](#) section 2.3.

**pwszServerName:** The client SHOULD pass a pointer to the fully qualified domain name of the target server as a null-terminated UTF-16LE character string. The server MUST ignore this value.

**pszZone:** The name of the zone that is being operated on. This MUST be set to NULL unless pszOperation is set to QueryDwordProperty, in which case this value MUST be set either to NULL (to indicate that DNS Server Configuration information is being requested) or to the name of the zone to be queried in UTF-8 format (to indicate that a DNS Zone integer property is being requested). This value will be used by certain operations as specified in the table below.

**pszOperation:** The operation to perform. The value of pszOperation MUST be one of the following:

Value	Meaning
EnumZones	Enumerate zones present on the DNS server qualifying for a specified simple zone filter value. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_DWORD and pDataIn MUST point to any combination of <a href="#">ZONE REQUEST FILTERS (section 2.2.5.1.4)</a> values. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_ZONE_LIST and ppDataOut MUST point to a structure of one of the types specified in <a href="#">DNS_RPC_ZONE_LIST (section 2.2.5.2.3)</a> .
EnumZones2	Enumerate zones present on the DNS server qualifying for a specified complex zone filter value. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_ENUM_ZONES_FILTER and pDataIn MUST point to a structure of type <a href="#">DNS_RPC_ENUM_ZONES_FILTER (section 2.2.5.2.9)</a> . Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_ZONE_LIST and MUST ppDataOut point to a structure of one of the types

Value	Meaning
	specified in DNS_RPC_ZONE_LIST.
EnumDirectoryPartitions	Enumerate the Application Directory Partition Table known to the DNS server. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_DWORD and pDataIn MUST be set to zero if all application directory partitions should be enumerated or to 0x00000001 if the DNS domain partition and DNS forest partition should be excluded from results. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_DP_LIST and ppDataOut MUST point to a structure of type <a href="#">DNS RPC DP LIST (section 2.2.7.2.4)</a> .
DirectoryPartitionInfo	Retrieve detailed information about a specified application directory partition. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_LPSTR and pDataIn MUST point to a null-terminated UTF-8 string specifying the distinguished name of an application directory partition. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be DNSSRV_TYPEID_DP_INFO and ppDataOut MUST point to a structure of type <a href="#">DNS RPC DP INFO (section 2.2.7.2.1)</a> .
Statistics	Retrieve statistics. On input dwTypeIn MUST be set to DNSSRV_TYPEID_DWORD and pDataIn MUST point to any combination of the <a href="#">DNSSRV_STATID TYPES (section 2.2.10.1.1)</a> values. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_BUFFER and ppDataOut MUST point to a <a href="#">DNS RPC BUFFER structure (section 2.2.1.2.2)</a> that contains a list of variable sized <a href="#">DNSSRV_STAT structures (section 2.2.10.2.2)</a> .
QueryDwordProperty	Retrieve the value of a 32-bit integer property. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_LPSTR and pDataIn MUST point to a null-terminated UTF-8 string specifying a zone property name listed in section <a href="#">3.1.1.2.1</a> (if pszZone is non-NULL) or server property name listed in section <a href="#">3.1.1.1.1</a> (if pszZone is NULL). Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_DWORD and ppDataOut MUST point to a DWORD value.
IpValidate	Validate a list of IP addresses. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_IP_VALIDATE and pDataIn MUST point to a <a href="#">DNS RPC IP_VALIDATE structure (section 2.2.3.2.4)</a> containing a list of IP addresses to be validated and the context information for validation as specified in section <a href="#">2.2.3.2.4</a> . Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_ADDRARRAY and ppDataOut MUST point to a structure of type <a href="#">DNS ADDR_ARRAY (section 2.2.3.2.3)</a> that contains IP validation results (section <a href="#">2.2.3.2.1</a> ).
ModifyZoneSigningKeyDescriptor	Creates, deletes, or modifies a signing key descriptor (SKD) for the specified zone. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_SKD and pDataIn MUST point to a structure



Value	Meaning
	of type <a href="#">DNS_RPC_SKD (section 2.2.6.2.1)</a> . If GUID inside DNS_RPC_SKD is set to zero, the server MUST create a new signing key descriptor. If GUID inside DNS_RPC_SKD is set to a nonzero value and if all other fields in the structure are NULL, the server MUST delete the signing key descriptor from the zone. Otherwise, the server MUST modify the signing key descriptor for the specified zone. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_SKD and ppDataOut MUST point to a structure of type <b>DNS_RPC_SKD</b> (section 2.2.6.2.1). <a href="#">&lt;309&gt;</a>
EnumZoneSigningKeyDescriptors	Retrieves the signing key descriptor found in the zone's signing key descriptor list for the specified zone. Input parameters (dwTypeIn and pDataIn) are ignored. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_SKD_LIST and ppDataOut MUST point to a structure of type <a href="#">DNS_RPC_SKD_LIST (section 2.2.6.2.2)</a> . <a href="#">&lt;310&gt;</a>
GetZoneSigningKeyDescriptorState	Retrieves the SKD state (section <a href="#">2.2.6.2.3</a> ) for the specified zone and the signing key descriptor GUID. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_LPWSTR and pDataIn MUST point to a string containing the GUID of the signing key descriptor. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_SKD_STATE and ppDataOut MUST point to a structure of type <b>DNS_RPC_SKD_STATE</b> (section 2.2.6.2.3). <a href="#">&lt;311&gt;</a>
SetZoneSigningKeyDescriptorState	Modifies the SKD state (section <a href="#">2.2.6.2.3</a> ) for the specified zone and the signing key descriptor GUID. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_SKD_STATE and pDataIn MUST point to a structure of type <b>DNS_RPC_SKD_STATE</b> (section 2.2.6.2.3). Note that only one key pointer string inside DNS_RPC_SKD_STATE will be set per a specific operation as described below. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_SKD_STATE and ppDataOut MUST point to a structure of type <b>DNS_RPC_SKD_STATE</b> (section 2.2.6.2.3) containing the modified SKD state. <a href="#">&lt;312&gt;</a>
ValidateZoneSigningParameters	Validates the zone signing parameters and returns a structure describing the invalid signing parameters. Input parameters (dwTypeIn and pDataIn) are ignored. Unless success is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_SIGNING_VALIDATION_ERROR and ppDataOut MUST point to a structure of type <a href="#">DNS_RPC_SIGNING_VALIDATION_ERROR (section 2.2.6.2.8)</a> containing invalid elements of the zone configuration. <a href="#">&lt;313&gt;</a>
EnumerateKeyStorageProviders	Enumerates key storage providers installed on the DNS server. On input, dwTypeIn and pDataIn are ignored. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to

Value	Meaning
	DNSSRV_TYPEID_UNICODE_STRING_LIST and ppDataOut MUST point to a structure of type <a href="#">DNS RPC UNICODE STRING LIST (section 2.2.1.2.4)</a> that contains a list of storage providers installed on the DNS server. <314>
EnumerateTrustPoints	Retrieve a list of trust points, containing either all trust points or only those at or below a given FQDN. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_LPSTR and pDataIn MUST point either to a null pointer or to a null-terminated UTF-8 string specifying an FQDN. If pDataIn points to a null pointer, the server MUST return all of the trust points. If pDataIn is an FQDN and there is a trust point or parent of a trust point at the FQDN, the server MUST return the trust point at the FQDN (or an empty trust-point structure for the FQDN if the FQDN is not a trust point) followed by empty trust-point structures for each immediate child of the FQDN, if any. An empty trust-point structure is a structure in which eTrustPointState is TRUST_POINT_STATE_INITIALIZED and all elements other than pszTrustPointName and dwRpcStructureVersion are zero. If pDataIn is an FQDN and there is neither a trust point nor the parent of a trust point at the FQDN, the server MUST return a nonzero error. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_TRUST_POINT_LIST and ppDataOut MUST point to a structure of type <a href="#">DNS RPC TRUST POINT LIST (section 2.2.6.2.5)</a> . <315>
EnumerateTrustAnchors	Retrieve a list of the trust anchors at a given FQDN. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_LPSTR and pDataIn MUST point to a null-terminated UTF-8 string specifying an FQDN. If the FQDN specified is not a trust point, the server MUST return a nonzero error. Unless an error is returned, on output the value pointed to by pdwTypeOut MUST be set to DNSSRV_TYPEID_TRUST_ANCHOR_LIST and ppDataOut MUST point to a structure of type <a href="#">DNS RPC TRUST ANCHOR LIST (section 2.2.6.2.7)</a> . <316>
ExportZoneSigningSettings	Exports all the Dnssec settings of a file-backed primary zone from a server. On input, dwTypeIn MUST be set to DNSSRV_TYPEID_DWORD and pDataIn SHOULD be 1 to get KSK details in the exported <b>DNS_RPC_ZONE_DNSSEC_SETTINGS</b> structure; otherwise, pDataIn SHOULD be zero. Unless an error is returned, on output pdwTypeOut is set to DNSSRV_TYPEID_ZONE_SIGNING_SETTINGS and ppDataOut points to a structure of type <b>PDNS_RPC_ZONE_DNSSEC_SETTINGS</b> . <317>
ImportZoneSigningSettings	Imports the Dnssec settings to a primary file-backed zone on a server and takes appropriate action based on the signing metadata imported.  On input, dwTypeIn MUST be set to DNSSRV_TYPEID_ZONE_SIGNING_SETTINGS and pDataIn

Value	Meaning
	<p>should be a structure of type PDNS_RPC_ZONE_DNSSEC_SETTINGS.</p> <p>If this operation is invoked on a server that hosts a primary unsigned copy of a file-backed zone and the <code>fIsSigned</code> Property of PDNS_RPC_ZONE_DNSSEC_SETTINGS is 1, then the server becomes a <b>nonkey master primary server</b> for that zone. Unless an error is returned, on output <code>pdwTypeOut</code> should be set to <code>DNSSRV_TYPEID_DWORD</code> and <code>ppDataOut</code> should point to a structure of type <code>ImportOpResult</code>. <a href="#">&lt;318&gt;</a></p>
EnumZoneScopes	<p>Enumerates all the zone scopes in a zone. On input, <code>dwTypeIn</code> MUST be set to <code>DNSSRV_TYPEID_NULL</code> and <code>pDataIn</code> SHOULD be NULL. The <code>pszZone</code> MUST be the zone name for which zone scopes are to be enumerated. Unless an error is returned, on output <code>pdwTypeOut</code> is set to <code>DNSSRV_TYPEID_ZONE_SCOPE_ENUM</code> and <code>ppDataOut</code> points to a structure of type <code>PDNS_RPC_ENUM_ZONE_SCOPE_LIST</code>. <a href="#">&lt;319&gt;</a></p>
ZoneStatistics	<p>Gets the zone statistics from the server. On input, <code>dwTypeIn</code> MUST be set to <code>DNSSRV_TYPEID_DWORD</code> and <code>pDataIn</code> SHOULD be either <code>DNS_RPC_ZONE_STATS_GET</code>, which gets the current zone statistics, or <code>DNS_RPC_ZONE_STATS_CLEAR</code>, which clears the zone statistics after getting them. The <code>pZone</code> MUST point to the zone information for which statistics are required. Unless an error is returned, on output <code>pdwTypeOut</code> SHOULD be set to <code>DNSSRV_TYPEID_ZONE_STATS</code> and <code>ppDataOut</code> SHOULD point to a structure of type <code>DNS_RPC_ZONE_STATS</code>. <a href="#">&lt;320&gt;</a></p>

**dwTypeIn:** A [DNS\\_RPC\\_TYPEID \(section 2.2.1.1.1\)](#) value indicating the type of input data pointed to by `pDataIn`.

**pDataIn:** Input data of type [DNSSRV\\_RPC\\_UNION \(section 2.2.1.2.6\)](#), which contains a data structure of the type indicated by `dwTypeIn`.

**pdwTypeOut:** A pointer to a DWORD that on success returns a `DNS_RPC_TYPEID` (section [2.2.1.1.1](#)) value indicating the type of output data pointed to by `ppDataOut`.

**ppDataOut:** A pointer to output data of type **DNSSRV\_RPC\_UNION**, which on success contains a data structure of the type indicated by `pdwTypeOut`.

**Return Values:** The method MUST return `ERROR_SUCCESS` (0x00000000) on success or a nonzero Win32 error code value if an error occurred. All error values MUST be treated the same.

When processing this call, the server MUST do the following:

- If the **Global Server State** (section [3.1.1](#)) is not "Running", return a failure.
- Check that the input parameters conform to the syntax requirements above, and if not, return a failure.
- If `pszZone` is not NULL, verify that `pszOperation` is set to `QueryDwordProperty`, `ModifyZoneSigningKeyDescriptor`, `EnumZoneSigningKeyDescriptors`,

GetZoneSigningKeyDescriptorState, SetZoneSigningKeyDescriptorState, or ValidateZoneSigningParameters. If it is not, return a failure.

- If pszOperation is set to QueryDwordProperty, verify that dwTypeIn is set to DNSSRV\_TYPEID\_LPSTR. If it is not, return a failure.
- If pszOperation is set to ModifyZoneSigningKeyDescriptor, verify that dwTypeIn is set to DNSSRV\_TYPEID\_SKD. If it is not, return a failure.
- If pszOperation is set to GetZoneSigningKeyDescriptorState, verify that dwTypeIn is set to DNSSRV\_TYPEID\_LPWSTR. If it is not, return a failure.
- If pszOperation is set to SetZoneSigningKeyDescriptorState, verify that dwTypeIn is set to DNSSRV\_TYPEID\_SKD\_STATE. If it is not, return a failure.
- If pszZone is not NULL, search the DNS Zone Table (section [3.1.1](#)) for the zone with a name matching the value of pszZone. If a matching zone cannot be found, return a failure.
- Validate, as specified in section [3.1.6.1](#), that the client has permissions to perform the attempted operation. If pszZone is NULL then the DNS server MUST perform the Phase 2 authorization test using the DNS Server Configuration Access Control List. If pszZone is not NULL then the DNS server MUST perform the Phase 2 authorization test using the Zone Access Control List for the zone specified by pszZone. Write privilege MUST be tested for the ModifyZoneSigningKeyDescriptor and SetZoneSigningKeyDescriptorState operations. Read privilege MUST be tested for all other operations. If the client does not have permission to perform the operation, the server MUST return a failure.
- If pszZone is not NULL, and pszOperation is QueryDwordProperty, and pDataIn matches a property name listed in section [3.1.6.1](#), and the server supports that property, the server MUST set pdwTypeOut to DNSSRV\_TYPEID\_DWORD, set ppDataOut to the DWORD value of the property, and return success. If the property name is not supported, the server MUST return a nonzero error code.
- If pszZone is not NULL, and pszOperation is ModifyZoneSigningKeyDescriptor:
  - If the **GUID** field of the **DNS\_RPC\_SKD** structure pointed to by pDataIn is set to zero, the server MUST create a new signing key descriptor for the specified zone and add it to the zone's signing key descriptor list. If this operation succeeds, the server MUST set pdwTypeOut to DNSSRV\_TYPEID\_SKD and MUST set ppDataOut to point to a structure of type DNS\_RPC\_SKD representing the newly created signing key descriptor, and return success. Otherwise, the server MUST return a nonzero error code.
  - If the **GUID** field of the **DNS\_RPC\_SKD** structure pointed to by pDataIn is nonzero, and all other fields are set to zero, the server MUST locate the specified SKD in the zone's signing key descriptor list. If the SKD is found, the server MUST delete the specified SKD from the zone's signing key descriptor list. If this operation succeeds, the server MUST set pdwTypeOut to DNSSRV\_TYPEID\_SKD and MUST set ppDataOut to point to a structure of type DNS\_RPC\_SKD representing the deleted signing key descriptor, and return success. Otherwise, the server MUST return a nonzero error code.
  - If the **GUID** field of the **DNS\_RPC\_SKD** structure pointed to by pDataIn is nonzero, and one or more of the other fields are nonzero, the server MUST locate the specified SKD in the zone's signing key descriptor list. If the SKD is found, the server MUST modify this SKD as follows:
    - If any of the dwDNSKEYSignatureValidityPeriod, "dwDSSignatureValidityPeriod", "dwStandardSignatureValidityPeriod", or "dwNextRolloverAction" fields of the

DNS\_RPC\_SKD pointed to by pDataIn are nonzero, modify the appropriate field in the SKD found in the zone's signing key descriptor list.

- If the **dwRolloverPeriod** field of the DNS\_RPC\_SKD structure pointed to by pDataIn is nonzero, modify the appropriate field of the SKD found in the zone's signing key descriptor list. If the previous value of this field in the SKD was 0xFFFFFFFF, and the new value is some other nonzero value, the server MUST also modify the signing key descriptor's associated **ftNextRolloverTime** to the current time plus the new **dwRolloverPeriod**. If the previous value of this field in the SKD was not 0xFFFFFFFF, and the new value is 0xFFFFFFFF, the server MUST also set the signing key descriptor's associated **ftNextRolloverTime** to zero.
- If any other fields of the DNS\_RPC\_SKD pointed to by pDataIn are nonzero, ignore these fields.

If this operation succeeds, the server MUST set pdwTypeOut to DNSSRV\_TYPEID\_SKD and MUST set ppDataOut to point to a structure of type DNS\_RPC\_SKD representing the modified signing key descriptor, and return success. Otherwise, the server MUST return a nonzero error code.

- If pszZone is not NULL, and pszOperation is SetZoneSigningKeyDescriptorState, the server MUST locate the SKD specified by the **GUID** field of the DNS\_RPC\_SKD\_STATE pointed to by pDataIn in the zone's signing key descriptor list. If the SKD is found, the server MUST modify exactly one of the signing key descriptor's **pwszActiveKey**, **pwszStandbyKey**, or **pwszNextKey** fields as follows:
  - If the **pwszActiveKey** field of the DNS\_RPC\_SKD\_STATE structure pointed to by pDataIn is nonzero, modify the signing key descriptor's **pwszActiveKey** field.
  - If the **pwszActiveKey** field of the DNS\_RPC\_SKD\_STATE structure pointed to by pDataIn is NULL, and the **pwszStandbyKey** field of this structure is nonzero, modify the signing key descriptor's **pwszStandbyKey**.
  - If both the **pwszActiveKey** and **pwszStandbyKey** fields of the DNS\_RPC\_SKD\_STATE structure pointed to by pDataIn are NULL, and the **pwszNextKey** field of this structure is nonzero, modify the signing key descriptor's **pwszNextKey**.

If this operation succeeds, the server MUST set pdwTypeOut to DNSSRV\_TYPEID\_SKD\_STATE and MUST set ppDataOut to point to a structure of type DNS\_RPC\_SKD\_STATE representing the modified signing key descriptor, and return success. Otherwise, the server MUST return a nonzero error code.

- If pszZone is not NULL, and pszOperation is EnumZoneSigningKeyDescriptors, GetZoneSigningKeyDescriptorState, or ValidateZoneSigningParameters, execute the operation indicated by the value of pszOperation as previously specified against the specified zone.
- If pszZone is NULL, execute the operation indicated by the value of pszOperation as specified above.
- If pszZone is not NULL, and pszOperation is ExportZoneSigningSettings:
  - On input, dwTypeIn MUST be set to DNSSRV\_TYPEID\_DWORD.
  - To get KSK details in exported DNS\_RPC\_ZONE\_DNSSEC\_SETTINGS, pDataIn is 1; otherwise, zero.

- Unless an error is returned, on output `pdwTypeOut` should be set to `DNSSRV_TYPEID_ZONE_SIGNING_SETTINGS` and `ppDataOut` should point to a structure of type `PDNS_RPC_ZONE_DNSSEC_SETTINGS`.
- If `pszZone` is not NULL, and `pszOperation` is `ImportZoneSigningSettings`:
  - On input, `dwTypeIn` MUST be set to `DNSSRV_TYPEID_ZONE_SIGNING_SETTINGS` and `pDataIn` should be a structure of type `PDNS_RPC_ZONE_DNSSEC_SETTINGS`.
  - Unless an error is returned, on output `pdwTypeOut` should be set to `DNSSRV_TYPEID_DWORD` and `ppDataOut` should point to a structure of type `ImportOpResult`.
  - If this operation is invoked on a server hosting a file-backed unsigned zone, and if the `flsSigned` property of `DNS_RPC_ZONE_DNSSEC_SETTINGS` is 1, then the server becomes a **nonkey master primary server** of that signed zone.
  - If this operation is invoked on a server hosting a file-backed signed zone, and if the `flsSigned` property of `DNS_RPC_ZONE_DNSSEC_SETTINGS` is 1, and if there is a difference in the imported and existing settings that requires re-signing, then the zone is marked for staggered signing with the new settings.
  - If this operation is invoked on a server hosting a file-backed signed zone, and if the `flsSigned` property of `DNS_RPC_ZONE_DNSSEC_SETTINGS` is 0, then the zone is unsigned on the target server.
- If `pszZone` is not NULL, and `pszOperation` is `EnumZoneScopes`:
  - On input, `dwTypeIn` MUST be set to `DNSSRV_TYPEID_NULL`.
  - `pDataIn` SHOULD be NULL.
  - The `pszZone` MUST be the zone name for which zone scopes are to be enumerated.
  - Unless an error is returned, on output `pdwTypeOut` should be set to `DNSSRV_TYPEID_ZONE_SCOPE_ENUM` and `ppDataOut` should point to a structure of type `PDNS_RPC_ENUM_ZONE_SCOPE_LIST`.
- If `pszZone` is not NULL, and `pszOperation` is `ZoneStatistics`:
  - On input, `dwTypeIn` MUST be set to `DNSSRV_TYPEID_DWORD`.
  - `pDataIn` SHOULD be either `DNS_RPC_ZONE_STATS_GET`, which gets the current Zone statistics or `DNS_RPC_ZONE_STATS_GETCLEAR` if one wants to clear the Zone Statistics after getting them.
  - The `pZone` MUST point to the zone information for which statistics are required.
  - Unless an error is returned, on output `pdwTypeOut` SHOULD be set to `DNSSRV_TYPEID_ZONE_STATS` and `ppDataOut` SHOULD point to a structure of type `DNS_RPC_ZONE_STATS`.

#### 3.1.4.4 R\_DnssrvEnumRecords (Opnum 3)

The `R_DnssrvEnumRecords` method enumerates DNS records on the server.

```
LONG R_DnssrvEnumRecords (
    [in] handle_t hBindingHandle,
```

```

[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in, unique, string] LPCSTR pszNodeName,
[in, unique, string] LPCSTR pszStartChild,
[in] WORD wRecordType,
[in] DWORD fSelectFlag,
[in, unique, string] LPCSTR pszFilterStart,
[in, unique, string] LPCSTR pszFilterStop,
[out] PDWORD pdwBufferLength,
[out, size_is(, *pdwBufferLength)]
    PBYTE* ppBuffer
);

```

**hBindingHandle:** An RPC binding handle to the server. Details concerning binding handles are specified in [\[C706\]](#) section 2.3.

**pwszServerName:** The client SHOULD pass a pointer to the fully qualified domain name of the target server as a null-terminated UTF-16LE character string. The server MUST ignore this value.

**pszZone:** A pointer to a null-terminated character string that contains the name of the zone to be queried. For operations specific to a particular zone, this field MUST contain the name of the zone in UTF-8 format. For all other operations, this field MUST be NULL.

**pszNodeName:** A pointer to a null-terminated character string that contains the node name at which to modify a record. A string that is not dot-terminated indicates a name relative to the zone root. A value of "@" indicates the zone root itself. A dot-terminated string indicates the name is an FQDN.

**pszStartChild:** A pointer to a null-terminated character string that contains the name of the child node at which to start enumeration. A NULL value indicates to start a new record enumeration. The client application can pass the last retrieved child node of pszNodeName to continue a previous enumeration.

**wRecordType:** An integer value that indicates the type of record to enumerate. Any value can be used, as specified in [DNS\\_RECORD\\_TYPE \(section 2.2.2.1.1\)](#). The query-only value DNS\_TYPE\_ALL indicates all types of records.

**fSelectFlag:** An integer value that specifies what records should be included in the response. Any combination of the values below MUST be supported. Values not listed below MUST be ignored.

Value	Meaning
DNS_RPC_VIEW_AUTHORITY_DATA 0x00000001	Include records from authoritative zones.
DNS_RPC_VIEW_CACHE_DATA 0x00000002	Include records from the DNS server's cache.
DNS_RPC_VIEW_GLUE_DATA 0x00000004	Include glue records.
DNS_RPC_VIEW_ROOT_HINT_DATA 0x00000008	Include root hint records.

Value	Meaning
DNS_RPC_VIEW_ADDITIONAL_DATA 0x00000010	Include additional records.
DNS_RPC_VIEW_NO_CHILDREN 0x00010000	Do not include any records from child nodes.
DNS_RPC_VIEW_ONLY_CHILDREN 0x00020000	Include only children nodes of the specified node in the results. For example: if a zone, "example.com", has child nodes, "a.example.com" and "b.example.com", calling R_DnsSrcEnumRecords(..., "example.com", "example.com", NULL, DNS_TYPE_ALL, DNS_RPC_VIEW_ONLY_CHILDREN, ..., ..., ...) will return DNS_RPC_NODES for "a" and "b".

**pszFilterStart:** Reserved for future use only. This MUST be set to NULL by clients and ignored by servers.

**pszFilterStop:** Reserved for future use only. This MUST be set to NULL by clients and ignored by servers.

**pdwBufferLength:** A pointer to an integer that on success contains the length of the buffer pointed to by ppBuffer.

**ppBuffer:** A pointer to a pointer that on success points to a buffer containing the enumerated records. The buffer is a series of structures beginning with a DNS\_RPC\_NODE structure (section 2.2.2.2.3). The records for the node will be represented by a series of DNS\_RPC\_RECORD (section 2.2.2.2.5) structures. The number of DNS\_RPC\_RECORD structures following a DNS\_RPC\_NODE structure is given by the wRecordCount member of DNS\_RPC\_NODE.

**Return Values:** The method MUST return ERROR\_SUCCESS (0x00000000) on success or a non-zero Win32 error code if an error occurred. All error values MUST be treated the same, except that if the return code is ERROR\_MORE\_DATA (0x000000EA) then the enumeration contains more results than can fit into a single RPC buffer. In this case the client application can call this method again passing the last retrieved child as the pszStartChild argument to retrieve the next set of results.

When processing this call, the server MUST do the following:

- If the Global Server State (section 3.1) is not "Running", return a failure.
- Check that the input parameters conform to the syntax requirements above, and if not return a failure.
- If pszZone is not NULL, search the DNS Zone Table for a zone with a name matching the value of pszZone. If a matching zone cannot be found, return a failure. If pszZone is NULL, assume for the operations below that pszZone specifies the cache zone.
- Validate, as specified in section 3.1.6.1, that the client has permissions to perform the attempted operation. The DNS server MUST perform the Phase 2 authorization test using the Zone Access Control List for the zone specified by pszZone. Read privilege MUST be tested for this operation. If the client does not have permission to perform the operation, the server MUST return a failure.
- Locate the node indicated by pszNodeName in the zone indicated by pszZoneName. If no such node is found, then return DNS\_ERROR\_NAME\_DOES\_NOT\_EXIST, and set the output buffer length to zero.



- If pStartChild is non-NULL it indicates that this call is in continuation of an earlier call to R\_DnssrvEnumRecords that returned ERROR\_MORE\_DATA (0x000000EA), hence the server MUST attempt to locate this node and return failure if it cannot be found. The server MUST then continue the enumeration from there.
- Return output records that meet the criteria specified by the value of fSelectFlag, in parameters pointed to by pdwBufferLength and ppBuffer, and return success. If these criteria are met--the zone is stored in a directory server; the DsMinimumBackgroundLoadThreads server property (section 3.1.1.1.1) is not 0x00000000; the zone is currently loading in the background; and the node indicated by pszNodeName has not yet been loaded into memory--the DNS server MUST behave as if the requested data does not exist in the zone.
- The server MUST return matching records for any **wType** value that is explicitly defined in the **DNS\_RECORD\_TYPE** structure. The server MUST also respond to type values exceeding 0x0031 that have matching records.

### 3.1.4.5 R\_DnssrvUpdateRecord (Opnum 4)

The R\_DnssrvUpdateRecord method is used to add a new DNS record or modify/delete an existing DNS record at the server. This operation SHOULD be supported. <321>

```
LONG R_DnssrvUpdateRecord(
    [in] handle_t hBindingHandle,
    [in, unique, string] LPCWSTR pwszServerName,
    [in, unique, string] LPCWSTR pszZone,
    [in, string] LPCWSTR pszNodeName,
    [in, unique] PDNS_RPC_RECORD pAddRecord,
    [in, unique] PDNS_RPC_RECORD pDeleteRecord
);
```

**hBindingHandle:** An RPC binding handle to the server. Details concerning binding handles are specified in [\[C706\]](#) section 2.3.

**pwszServerName:** The client SHOULD pass a pointer to the fully qualified domain name of the target server as a null-terminated UTF-16LE character string. The server MUST ignore this value.

**pszZone:** A pointer to a null-terminated character string that contains the name of the zone to be queried. For operations specific to a particular zone, this field MUST contain the name of the zone in UTF-8 format. For all other operations, this field MUST be NULL.

**pszNodeName:** A pointer to a null-terminated character string that contains the node name at which to modify a record. A string that is not dot-terminated indicates a name relative to the zone root. A value of "@" indicates the zone root itself. A dot-terminated string indicates the name is an FQDN.

**pAddRecord:** A pointer to a structure of type DNS\_RPC\_RECORD (section 2.2.2.5) that contains information based on the operation being performed as specified below.

**pDeleteRecord:** A pointer to a structure of type DNS\_RPC\_RECORD (section 2.2.2.5) that contains information based on the operation being performed as specified below.

To add a record:

- pAddRecord: The DNS RR data of the new record.

- pDeleteRecord: MUST be set to NULL.

To delete a record:

- pAddRecord: MUST be set to NULL.
- pDeleteRecord: Individual DNS RR data of the record to be deleted.

To replace a record:

- pAddRecord: New record data.
- pDeleteRecord: Old record data.

To add an empty node:

- pAddRecord: MUST be set to NULL.
- pDeleteRecord: MUST be set to NULL.

**Return Values:** The method MUST return ERROR\_SUCCESS (0x00000000) on success or a non-zero Win32 error code if an error occurred. All error values MUST be treated the same.

When processing this call, the server MUST do the following:

- If the Global Server State (section [3.1](#)) is not "Running", return a failure.
- Check that the input parameters conform to the syntax requirements above, and if not return a failure.
- The server SHOULD support **pAddRecord** and/or **pDeleteRecord** for the explicitly defined types in section [2.2.2.1.1.<322>](#). If any of the passed record types are not supported by the server, return a failure.
- If the **pAddRecord** is for an explicitly defined resource record type DNS\_TYPE\_CNAME (section [2.2.2.1.1](#)), then delete any existing DNS\_TYPE\_CNAME record for the node specified in **pszNodeName**, before adding the record.
- If pszZone is not NULL, search the DNS Zone Table for a zone with a name matching the value of pszZone. If a matching zone cannot be found return a failure.
- Validate, as specified in section [3.1.6.1](#), that the client has permissions to perform the attempted operation. The DNS server MUST perform the Phase 2 authorization test using the Zone Access Control List for the zone specified by pszZone. Read privilege MUST be tested for this operation. If the client does not have permission to perform the operation, the server MUST return a failure.
- Locate the node indicated by **pszNodeName** in the zone indicated by pszZoneName. If no such node is found then return ERROR\_SUCCESS.
- If both **pAddRecord** and pDeleteRecord are NULL, then the server MUST add an empty node to the database if the node does not already exist. If the node already exists, the server MUST return ERROR\_SUCCESS.
- If pszZoneName is NULL or points to "..Cache" then the operation SHOULD be performed on the DNS server's cache and MAY be performed on the DNS server's set of root hint records. [<323>](#) If pszZoneName points to "..RootHints" then the operation MUST be performed on the DNS server's set of root hint records.

- If `pszZoneName` points to a primary zone, attempt to perform addition/deletion/update of the record. If the operation is successful, increment the zone serial number using serial number arithmetic [RFC1982]. If the last record at the node is being deleted and the zone is stored in the directory server, the DNS server MUST set the node's `dnsTombstoned` attribute to "TRUE" and the node's `dnsRecord` (section 2.3.2.2) attribute to contain a `DNS_RPC_RECORD_TS` record (section 2.2.2.4.23) with an `EntombedTime` value equal to the current time expressed as the number seconds since 12:00 A.M. January 1, 1601 Coordinated Universal Time (UTC). If the zone is directory server-integrated and the update causes new or modified records to be committed to the directory, the new zone serial number MUST also be written to the **Serial** field of the **dnsRecord** attribute, as specified in 2.3.2.2. If this operation deletes the last record from the node and the zone is directory server-integrated, the DNS server MUST set the node's DNS Node Tombstone State (section 3.1.1) to TRUE by setting the value of the **dnsTombstoned** attribute to "TRUE" and writing a `DNS_RPC_RECORD_TS` (section 2.2.2.4.23) in the **dnsRecord** attribute.
- Return success or a failure to indicate the result of the attempted operation.

### 3.1.4.6 R\_DnssrvOperation2 (Opnum 5)

The `R_DnssrvOperation2` method is used to invoke a set of server functions specified by the caller. The DNS server SHOULD implement `R_DnssrvOperation2`.

All parameters are as specified by the `R_DnssrvOperation` method (section 3.1.4.1) with the following exceptions:

```
LONG R_DnssrvOperation2(
    [in] handle_t hBindingHandle,
    [in] DWORD dwClientVersion,
    [in] DWORD dwSettingFlags,
    [in, unique, string] LPCWSTR pwszServerName,
    [in, unique, string] LPCSTR pszZone,
    [in] DWORD dwContext,
    [in, unique, string] LPCSTR pszOperation,
    [in] DWORD dwTypeId,
    [in, switch_is(dwTypeId)] DNSSRV_RPC_UNION pData
);
```

**dwClientVersion:** The client version in `DNS_RPC_CURRENT_CLIENT_VER` (section 2.2.1.2.1) format.

**dwSettingFlags:** Reserved for future use. MUST be set to zero by clients and MUST be ignored by servers.

**Return Values:** The method MUST return `ERROR_SUCCESS` (0x00000000) on success or a nonzero Win32 error code value if an error occurred. All error values MUST be treated the same.

When processing this call, the server MUST perform the same actions as for the `R_DnssrvOperation` method (section 3.1.4.1) except in the event the `dwClientVersion` is greater than the server version, in which case the server MUST return the highest version number known to the server.

### 3.1.4.7 R\_DnssrvQuery2 (Opnum 6)

The R\_DnssrvQuery2 method queries the DNS server for information. The type of information queried for is specified by the client using the pszZone and pszOperation parameters. The DNS server SHOULD implement R\_DnssrvQuery2 [<324>](#).

All parameters are as specified by the R\_DnssrvQuery method (section [3.1.4.2](#)) with the following exceptions:

```
LONG R_DnssrvQuery2(  
    [in] handle_t hBindingHandle,  
    [in] DWORD dwClientVersion,  
    [in] DWORD dwSettingFlags,  
    [in, unique, string] LPCWSTR pwszServerName,  
    [in, unique, string] LPCSTR pszZone,  
    [in, unique, string] LPCSTR pszOperation,  
    [out] PDWORD pdwTypeId,  
    [out, switch_is(*pdwTypeId)] DNSSRV_RPC_UNION* ppData  
);
```

**dwClientVersion:** The client version in DNS\_RPC\_CURRENT\_CLIENT\_VER (section [2.2.1.2.1](#)) format.

**dwSettingFlags:** Reserved for future use only. This field MUST be set to zero by clients and ignored by servers.

**Return Values:** Return values behaviors and interpretations are same as they are for R\_DnssrvQuery method (section [3.1.4.2](#)).

When processing this call, the server MUST perform the same actions as for the R\_DnssrvQuery method (section [3.1.4.2](#)), except that for output structure types with multiple versions, the server MUST return the structure type selected by dwClientVersion except in the event the dwClientVersion is greater than the server version, in which case the server MUST return the highest version number known to itself.

### 3.1.4.8 R\_DnssrvComplexOperation2 (Opnum 7)

The R\_DnssrvComplexOperation2 method is used to invoke a set of server functions specified by the caller. These functions generally return more complex structures than simple 32-bit integer values, unlike the operations accessible through R\_DnssrvOperation. The DNS server SHOULD implement R\_DnssrvComplexOperation2.

All parameters are as specified by the R\_DnssrvComplexOperation method (section [3.1.4.3](#)) with the following exceptions:

```
LONG R_DnssrvComplexOperation2(  
    [in] handle_t hBindingHandle,  
    [in] DWORD dwClientVersion,  
    [in] DWORD dwSettingFlags,  
    [in, unique, string] LPCWSTR pwszServerName,  
    [in, unique, string] LPCSTR pszZone,  
    [in, unique, string] LPCSTR pszOperation,  
    [in] DWORD dwTypeIn,  
    [in, switch_is(dwTypeIn)] DNSSRV_RPC_UNION pDataIn,  
    [out] PDWORD pdwTypeOut,  
    [out, switch_is(*pdwTypeOut)] DNSSRV_RPC_UNION* ppDataOut
```

);

**dwClientVersion:** The client version in DNS\_RPC\_CURRENT\_CLIENT\_VER (section [2.2.1.2.1](#)) format.

**dwSettingFlags:** Reserved for future use only. This field MUST be set to zero by clients and ignored by servers.

**Return Values:** Return values and interpretations are the same as for R\_DnssrvComplexOperation (section [3.1.4.3](#)).

When processing this call, the server MUST perform the same actions as for the R\_DnssrvComplexOperation method (section [3.1.4.3](#)) except that for output structure types with multiple versions, the server MUST return the structure type selected by dwClientVersion except in the event the dwClientVersion is greater than the server version, in which case the server MUST return the highest version number known to itself.

### 3.1.4.9 R\_DnssrvEnumRecords2 (Opnum 8)

The R\_DnssrvEnumRecords2 method enumerates DNS records on the server. The DNS server SHOULD implement R\_DnssrvEnumRecords2 [<325>](#).

All parameters are as specified by the [R\\_DnssrvEnumRecords method \(section 3.1.4.4\)](#) with the following exceptions:

```
LONG R_DnssrvEnumRecords2(  
    [in] handle_t hBindingHandle,  
    [in] DWORD dwClientVersion,  
    [in] DWORD dwSettingFlags,  
    [in, unique, string] LPCWSTR pwszServerName,  
    [in, unique, string] LPCSTR pszZone,  
    [in, unique, string] LPCSTR pszNodeName,  
    [in, unique, string] LPCSTR pszStartChild,  
    [in] WORD wRecordType,  
    [in] DWORD fSelectFlag,  
    [in, unique, string] LPCSTR pszFilterStart,  
    [in, unique, string] LPCSTR pszFilterStop,  
    [out] PDWORD pdwBufferLength,  
    [out, size_is(*pdwBufferLength)]  
    PBYTE* ppBuffer  
);
```

**dwClientVersion:** The client version in DNS\_RPC\_CURRENT\_CLIENT\_VER (section [2.2.1.2.1](#)) format.

**dwSettingFlags:** Reserved for future use only. This field MUST be set to zero by clients and ignored by servers.

**Return Values:** Return values behaviors and interpretations are same as they are for **R\_DnssrvEnumRecords method**.

When processing this call, the server MUST perform the same actions as for the **R\_DnssrvEnumRecords method**.

### 3.1.4.10 R\_DnssrvUpdateRecord2 (Opnum 9)

The R\_DnssrvUpdateRecord2 method is used to add a new DNS record or modify/delete an existing DNS record at the server. The DNS server SHOULD implement R\_DnssrvEnumRecords2. [<326>](#)

All parameters are as specified by the [R\\_DnssrvUpdateRecord method \(section 3.1.4.5\)](#) with the following exceptions:

```
LONG R_DnssrvUpdateRecord2(  
    [in] handle_t hBindingHandle,  
    [in] DWORD dwClientVersion,  
    [in] DWORD dwSettingFlags,  
    [in, unique, string] LPCWSTR pwszServerName,  
    [in, unique, string] LPCSTR pszZone,  
    [in, string] LPCSTR pszNodeName,  
    [in, unique] PDNS_RPC_RECORD pAddRecord,  
    [in, unique] PDNS_RPC_RECORD pDeleteRecord  
);
```

**dwClientVersion:** The client version in DNS\_RPC\_CURRENT\_CLIENT\_VER (section [2.2.1.2.1](#)) format.

**dwSettingFlags:** Reserved for future use only. This field MUST be set to zero by clients and ignored by servers.

**Return Values:** The method MUST return ERROR\_SUCCESS (0x00000000) on success or a nonzero Win32 error code if an error occurred. All error values MUST be treated the same. All record types SHOULD be supported, but if an operation is attempted on an unsupported record type, the method MUST return a nonzero Win32 error code.

When processing this call, the server MUST perform the same actions as for the **R\_DnssrvUpdateRecord method**.

### 3.1.4.11 R\_DnssrvUpdateRecord3 (Opnum 10)

The R\_DnssrvUpdateRecord3 method [<327>](#) is used to add a new DNS record or modify or delete an existing DNS record in a zone or a zone scope if specified. The DNS server SHOULD [<328>](#) implement R\_DnssrvUpdateRecord3.

All parameters are as specified by the methods [R\\_DnssrvUpdateRecord \(section 3.1.4.5\)](#) and [R\\_DnssrvUpdateRecord2 \(section 3.1.4.10\)](#) with the following exceptions:

```
LONG R_DnssrvUpdateRecord3(  
    [in] handle_t hBindingHandle,  
    [in] DWORD dwClientVersion,  
    [in] DWORD dwSettingFlags,  
    [in, unique, string] LPCWSTR pwszServerName,  
    [in, unique, string] LPCSTR pszZone,  
    [in, unique, string] LPCWSTR pwszZonescope,  
    [in, string] LPCSTR pszNodeName,  
    [in, unique] PDNS_RPC_RECORD pAddRecord,  
    [in, unique] PDNS_RPC_RECORD pDeleteRecord  
);
```

**pwszZonescope:** A pointer to a null-terminated character string that contains the name of the zone scope inside the zone to be queried. For operations specific to a particular zone scope, this field MUST contain the name of the zone scope. If the value is NULL, the API gives the same behavior as **R\_DnssrvUpdateRecord2**.

When processing this call, the server MUST perform the same actions as for the **R\_DnssrvUpdateRecord2 method**.

### 3.1.4.12 R\_DnssrvEnumRecords3 (Opnum 11)

The **R\_DnssrvEnumRecords3** method<329> enumerates DNS records on a zone or a zone scope if specified. The DNS server SHOULD<330> implement **R\_DnssrvEnumRecords3**.

All parameters are as specified by the [R\\_DnssrvEnumRecords method \(section 3.1.4.4\)](#) and implement the [R\\_DnssrvEnumRecords2 method \(section 3.1.4.9\)](#) with the following exceptions:

```
LONG R_DnssrvEnumRecords3(  
    [in] handle_t hBindingHandle,  
    [in] DWORD dwClientVersion,  
    [in] DWORD dwSettingFlags,  
    [in, unique, string] LPCWSTR pwszServerName,  
    [in, unique, string] LPCSTR pszZone,  
    [in, unique, string] LPCWSTR pwszZoneScope,  
    [in, unique, string] LPCSTR pszNodeName,  
    [in, unique, string] LPCSTR pszStartChild,  
    [in] WORD wRecordType,  
    [in] DWORD fSelectFlag,  
    [in, unique, string] LPCSTR pszFilterStart,  
    [in, unique, string] LPCSTR pszFilterStop,  
    [out] PDWORD pdwBufferLength,  
    [out, size_is(, *pdwBufferLength)]  
    PBYTE* ppBuffer  
);
```

**pwszZoneScope:** A pointer to a null-terminated character string that contains the name of the zone scope inside the zone to be queried. For operations specific to a particular zone scope, this field MUST contain the name of the zone scope. If the value is NULL, the API gives the same behavior as the **R\_DnssrvEnumRecords2 method**.

When processing this call, the server MUST perform the same actions as for the **R\_DnssrvEnumRecords2 method**.

### 3.1.4.13 R\_DnssrvOperation3 (Opnum 12)

The **R\_DnssrvOperation3** method<331> is used to invoke a set of server functions specified by the caller on the zone scope if specified. The DNS server SHOULD<332> implement **R\_DnssrvOperation3**.

All parameters are as specified by the methods [R\\_DnssrvOperation \(section 3.1.4.1\)](#) and [R\\_DnssrvOperation2 \(section 3.1.4.6\)](#) with the following exceptions.

```
DNS_STATUS R_DnssrvOperation3(  
    [in] handle_t hBindingHandle,  
    [in] DWORD dwClientVersion,  
    [in] DWORD dwSettingFlags,
```

```

[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in, unique, string] LPCWSTR pwszZoneScopeName,
[in] DWORD dwContext,
[in, unique, string] LPCSTR pszOperation,
[in] DWORD dwTypeId,
[in, switch_is(dwTypeId)] DNSSRV_RPC_UNION pData
);

```

**pwszZoneScopeName:** A pointer to a null-terminated character string that contains the name of the zone scope inside the zone in which the operation is to be performed. For operations specific to a particular zone scope, this field **MUST** contain the name of the zone scope. If the value is NULL then the API gives the same behavior as **R\_DnssrvOperation2**. If the value is not NULL then pszZone **MUST** point to a null-terminated character string that contains the name of the zone in UTF-8 format. In this case the type of the zone pointed to by pszZone **MUST** be a primary zone. It **MUST** also be a non-autocreated zone, a non-reverse lookup zone, and a non-AD integrated zone.

If pszZone is not NULL and pwszZoneScopeName is not NULL, pszOperation **MUST** be set to one of the following values:

Value	Meaning
"DeleteNode"	Deletes a node from the given zone scope of a specified zone. On input, dwTypeId <b>MUST</b> be set to DNSSRV_TYPEID_NAME_AND_PARAM (section <a href="#">2.2.2.1.1</a> ), and pData <b>MUST</b> point to a structure of type <a href="#">DNS_RPC_NAME_AND_PARAM (section 2.2.1.2.5)</a> . That structure contains the name of the node to be deleted as pointed by pszNodeName, and a Boolean flag in dwParam that indicates if the node's sub-tree is to be deleted.
"DeleteRecordSet"	Deletes all the DNS records of a particular type at a particular node that is present in the given zone scope of a specified zone. On input, dwTypeId <b>MUST</b> be set to DNSSRV_TYPEID_NAME_AND_PARAM (section <a href="#">2.2.2.1.1</a> ). pData <b>MUST</b> point to a structure of type DNS_RPC_NAME_AND_PARAM (section <a href="#">2.2.1.2.5</a> ). That structure contains the name of the node to be deleted and the DNS record type in the dwParam member. The type <b>MUST</b> be of value of either <b>DNS_RECORD_TYPE</b> (section 2.2.2.1.1) or 0x00FF, which specifies all types.

When processing this call, the server **MUST** perform the same actions as for the **R\_DnssrvOperation2** method. Operations specific to this method are as described in the preceding table. These operations are processed as specified in **R\_DnssrvOperation**, in the given zone scope specified in pwszZoneScopeName of a zone specified in pszZone.

### 3.1.4.14 R\_DnssrvQuery3 (Opnum 13)

The **R\_DnssrvQuery3** method [<333>](#) queries the DNS server for information. The type of information queried for is specified by the client using the pszZone, pwszZoneScopeName, and pszOperation parameters. The DNS server **SHOULD** implement R\_DnssrvQuery3. [<334>](#)

All the parameters are as specified by the [R\\_DnssrvQuery2](#) method with the following exceptions:

```

LONG R_DnssrvQuery3(
[in] handle_t hBindingHandle,

```



```

[in] DWORD dwClientVersion,
[in] DWORD dwSettingFlags,
[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCWSTR pszZone,
[in, unique, string] LPCWSTR pwszZoneScopeName,
[in, unique, string] LPCWSTR pszOperation,
[out] PDWORD pdwTypeId,
[out, switch_is(*pdwTypeId)] DNSSRV_RPC_UNION* ppData
);

```

**pwszZoneScopeName:** A pointer to a null-terminated character string that contains the name of the zone scope inside the zone to be queried. For operations specific to a particular zone scope, this field **MUST** contain the name of the zone scope. If the value is NULL, then the API is as specified in **R\_DnssrvQuery2** (section 3.1.4.7).

If pszZone is not NULL and pwszZoneScopeName is not NULL, pszOperation **MUST** be set to one of the following values:

Value	Meaning
ScopeInfo	On output the value pointed to by pdwTypeId <b>SHOULD</b> be set to DNSSRV_TYPEID_ZONE_SCOPE_INFO and ppData <b>MUST</b> point to a structure of one of the types specified in <a href="#">DNS RPC ZONE SCOPE INFO V1 (section 2.2.13.2.3.1)</a> , which contains full information about the zone pointed to by pszZone.

- If pszZone is not NULL and pwszZoneScopeName is not NULL, and pszOperation value is ScopeInfo:
  - On input, pdwTypeId **MUST** be set to DNSSRV\_TYPEID\_ZONE\_SCOPE\_INFO.
  - pwszZoneScopeName **MUST** be the pointer to the name of the zone scope that is to be queried.
  - pszZone **MUST** be the name of the zone in which the zone scope information is queried.
  - ppData **MUST** be in the format of **DNS\_RPC\_ZONE\_SCOPE\_INFO\_V1**.

### 3.1.5 Timer Events

No protocol timer events are required on the server beyond the timers required in the underlying RPC protocol.

### 3.1.6 Other Local Events

#### 3.1.6.1 Three-phase authorization test

When a three-phase authorization test is performed, the following phases **MUST** be performed in order:

- **Phase 1:** If the DNS server is directory server integrated then the client's credentials **MUST** be tested for Read privilege against the DNS Server Configuration Access Control List (see section [3.1.1](#)). This tests whether or not the client should be granted access to any of the functionality of the DNS Server Management Protocol. If this test is passed, then the server **MUST** proceed to Phase 2. If the DNS server is not directory server integrated, and if the client is a member of either the Administrators group or the System Operators group, access **MUST** be granted and

further authorization testing MUST NOT be performed. Otherwise access MUST be denied and the server MUST return an error.

- **Phase 2:** If the authorization test in Phase 1 is passed and the DNS server is directory server integrated, then the DNS server MUST perform an explicit ACL check for either Read or Write privilege. The ACL used for this test MUST be one of the three listed in the following table, and, for either Read or Write privilege, as specified in the description of the request being processed.

Access Control List	Description
DNS Server Configuration Access Control List (see section <a href="#">3.1.1</a> )	This ACL is tested for Read privilege in Phase 1 to gate basic access to the protocol. It is also used to control access for any operation that is not performed against a specific zone or directory partition.
Application Directory Partition Access Control List (see section <a href="#">3.1.1</a> )	This ACL is used to control access for any operation that is performed against the directory partition. Operations that are performed against zones do not use this ACL.
Zone Access Control List (see section <a href="#">3.1.1</a> )	This ACL is used to control access for any operation that is performed against a zone that is stored in the directory server. If a zone is stored in the directory server inside a partition, any operation specific to the zone will use the Zone ACL.

- **Phase 3:** If the authorization test in Phase 2 is passed and the DNS server is directory server integrated, then the DNS server MUST impersonate the client for any actions performed against a directory server (for impersonation details, see [\[MS-RPCE\]](#) section 2.2.1.1.9), unless the target of the modification is a dnsNode object whose **Aging Time Stamp** attribute (section [3.1.1.2.4](#)) is older than the **Time Zone Secured** attribute of the zone (section [3.1.1](#)). If the operation against the directory server fails, the DNS server MUST return an error.

### 3.1.6.2 Directory server security descriptors reading and caching

If the server is directory server integrated, directory server **security descriptors** MUST be read from the directory server using LDAP every DsPollingInterval (section [3.1.1](#)) after DNS server boot. After each read, the server MUST cache the security descriptors.

Additionally, a Zone Access Control List (section [3.1.1](#)) security descriptor MUST be read from the directory server when the corresponding zone is loaded during server boot time. This security descriptor MUST also be read when the corresponding zone is created through the ZoneCreate operation (section [3.1.4.1](#)) and when the corresponding zone's directory partition encounters the EnlistDirectoryPartition operation (section [3.1.4.1](#)).

Additionally, an Application Directory Partition Access Control List (section [3.1.1](#)) security descriptor MUST be read from the directory server when the corresponding application directory partition is loaded during server boot time. This security descriptor MUST also be read when the corresponding application directory partition encounters the **EnlistDirectoryPartition** or ZoneChangeDirectoryPartition operation (section [3.1.4.1](#)).

### 3.1.6.3 dnsRecord in the Directory Server

If the server is directory server integrated, then whenever dnsRecord attribute values (section [2.3.2.2](#)) are written to the directory server by using LDAP, each string MUST be converted from type DNS\_RPC\_NAME (section [2.2.2.2.1](#)) to type DNS\_COUNT\_NAME (section [2.2.2.2.3](#)). Similarly, when reading dnsRecords, the DNS server MUST convert each string of type DNS\_COUNT\_NAME to type DNS\_RPC\_NAME.

### 3.1.6.4 Modifying Directory Server Security Descriptors

Wherever this document states that the security descriptor for a directory server object must be modified, the server MUST perform the following procedure:

- Invoke the "Performing an LDAP Operation on an ADConnection" task, as specified in [\[MS-ADSO\]](#) section 6.2.6.1.6, with the following parameters:
  - **TaskInputADConnection:** **DNS Server AD Connection**
  - **TaskInputRequestMessage:** protocolOp is set to **searchRequest** ([\[RFC4511\]](#) section 4.5).
  - The parameters of the **searchRequest** are set as follows:
    - **baseObject:** <The specified distinguished name of the object to be modified>
    - **scope:** base (0)
    - **derefAliases:** neverDerefAliases (0)
    - **sizeLimit:** 0
    - **timeLimit:** 360
    - **typesOnly:** FALSE
    - **filter:** "(objectCategory=\*)"
    - **attributes:** "ntSecurityDescriptor"
- If the search request was successful, modify the security descriptor returned, to grant or deny the specified rights to the specified local security group.
- If the security descriptor was successfully modified, invoke the "Performing an LDAP Operation on an ADConnection" task, as specified in [\[MS-ADSO\]](#) section 6.2.6.1.6, with the following parameters:
  - **TaskInputADConnection:** **DNS Server AD Connection**
  - **TaskInputRequestMessage:** protocolOp is set to **modifyRequest** ([\[RFC4511\]](#) section 4.6)
  - The parameters of the **modifyRequest** are set as follows:
    - **object:** <The specified distinguished name of the object to be modified>
    - **changes:**
      - **operation:** replace
      - **type:** "ntSecurityDescriptor"
      - **vals:** <modified security descriptor>

### 3.1.7 Key Processing Rules

#### 3.1.7.1 Constructing an Exported Key

The steps for constructing an exported key are as follows:

1. Obtain the Zone Signing Key. Encode it in a data structure appropriate for the signature algorithm, as specified in section [2.2.11.1.1](#).
2. If the zone is stored in the default DNS domain application directory partition or the default application directory partition, let ProtectionSid denote the DOMAIN\_DOMAIN\_CONTROLLERS SID specified in [\[MS-DTYP\]](#) section 2.4.2.4; otherwise, let ProtectionSid denote the ENTERPRISE\_DOMAIN\_CONTROLLERS SID specified in [\[MS-DTYP\]](#) section 2.4.2.4.
3. Construct a security descriptor in self-relative format, in the format specified in [\[MS-DTYP\]](#) section 2.4.6, using the following field values:
  1. **Sbz1** is set to zero.
  2. All **Control** bits except for SR are cleared.
  3. The **OwnerSid** field is set to ProtectionSid.
  4. The **GroupSid** field is set to the LOCAL\_SYSTEM SID specified in [\[MS-DTYP\]](#) section 2.4.2.4.
  5. The **Sacl** field is absent.
  6. The **Dacl** field contains two access control entries, as follows:
    1. The first entry contains an ACCESS\_ALLOWED\_ACE structure as specified in [\[MS-DTYP\]](#) section 2.4.4.2, with the **Sid** field set to ProtectionSid and the Mask value set to 0x3.
    2. The second entry contains an ACCESS\_ALLOWED\_ACE structure as specified in [\[MS-DTYP\]](#) section 2.4.4.2, with the **Sid** field set to the EVERYONE SID specified in [\[MS-DTYP\]](#) section 2.4.2.4 and the Mask value set to 0x2.
4. Obtain a group seed key for the security descriptor constructed in step 3, with NULL root key ID, using the procedure specified in [\[MS-GKDI\]](#) section 3.2.4.
5. Construct a Protection Key Attributes structure (section [2.2.11.2.5](#)) with the sidString element set to the string representation (specified in [\[MS-DTYP\]](#) section 2.4.2.1) of ProtectionSid (step 2).
6. Construct a Protection Key Identifier structure (section [2.2.11.2.4](#)) using the key identifiers obtained in step 2.
7. Derive an AES-256 key from the group seed key retrieved in step 4 as follows:
  1. Derive a 64-byte secret using the key derivation procedure specified in [\[SP800-108\]](#) section 5.1 with HMAC [\[FIPS198-1\]](#) using SHA-512 [\[FIPS180-3\]](#) as the underlying PRF, using the Unicode string L"KDS service" as the label, and using the Unicode string L"KDS private key" as the context.
  2. Compute the SHA-256 hash [\[FIPS180-3\]](#) of the secret computed in step 7a.
8. Generate a random 256-bit AES key for use as the content encryption key, and a 12-byte random nonce. Encrypt the structure created in step 1 using the AES algorithm [\[FIPS197\]](#) in GCM mode [\[SP800-38D\]](#) using this key and nonce, and using an Integrity Check Value (ICV) length of 16 bytes. Append the 16-byte ICV to the end of the cipher text.
9. Encrypt the content encryption key generated in step 8 with the key derived in step 7, using AES [\[FIPS197\]](#) in CBC mode [\[SP800-38A\]](#) with PKCS#7 padding ([\[RFC3852\]](#) section 6.3) with a random Initialization Vector (IV).

10. Construct a protected key blob using the CMS Enveloped-Data format specified in [\[RFC3852\]](#) section 6, as follows:
  1. Set the **version** field to 2.
  2. Omit the **originatorInfo** and **unprotectedAttrs** fields.
  3. In the **recipientInfos** structure, create a single **KEKRecipientInfo** member as follows:
    1. Set the **keyIdentifier** field of the KEK identifier to the result of step 6.
    2. Set the **other** field of the KEK identifier to the result of step 5.
    3. Omit the **date** field of the KEK identifier.
    4. Set the **keyEncryptionAlgorithm** field to AES256-CBC as specified in [\[RFC3565\]](#) section 4.1 and the **encryptedKey** field to the result of step 9.
  4. In the **EncryptedContentInfo** structure:
    1. Set the **contentType** field to Data, as specified in [\[RFC3852\]](#) section 4.
    2. Set the **contentEncryptionAlgorithm** field to AES256-GCM, as specified in [\[RFC5084\]](#) section 3.2, using the nonce and ICV length from step 8.
    3. Set the **encryptedContent** field to the result of step 8.
11. Construct an Exported Key Pair structure (section [2.2.11.2.6](#)) with the **ProtectedKeyBlob** field set to the result of step 10.

### 3.1.7.2 Importing an Exported Key

The steps for importing an exported key are as follows:

1. Validate that the input is in the format specified in section [2.2.11.2.6](#) by checking that the contents of the **Reserved1** and **Reserved2** fields match the expected values and that the **AlgorithmName** field has one of the values specified in section [2.2.11.1.1](#). If not, fail and return an error.
2. Extract the **ProtectedKeyBlob** field, and verify that it is a syntactically correct DER-encoded ASN.1 structure of the Enveloped-data content type specified in [\[RFC3852\]](#) section 6. If not, fail and return an error.
3. Extract the **keyIdentifier** field of the sole **KEKRecipientInfo** member of the **recipientInfos** structure in the protected key blob, and validate that it is in the format specified in section [2.2.11.2.6](#) by checking that the contents of the **Reserved1**, **Reserved2**, **Reserved3**, and **Reserved4** fields match their expected values. If not, fail and return an error.
4. Extract the **other** field of the KEK identifier in the aforementioned **KEKRecipientInfo** member, and validate that it is a syntactically correct DER-encoded ASN.1 structure conforming to the format specified in section [2.2.11.2.5](#). If not, fail and return an error.
5. Using the sidString value in the Protection Key Attributes structure obtained in step 4, construct a security descriptor as specified in section [3.1.7.1](#) step 3.
6. Using the information in the **Protection Key Identifier** structure obtained in step 3 and the security descriptor constructed in step 5, obtain a group seed key by using the procedure specified in [\[MS-GKDI\]](#) section 3.2.4. If unsuccessful, fail and return an error.

7. From the seed key obtained in step 6, derive the key encryption key as specified in section [3.1.7.1](#) step 7.
8. Use the key encryption key derived in step 7 to decrypt the contents of the **encryptedKey** field of the **KEKRecipientInfo** member obtained in step 3 by reversing the process in section [3.1.7.1](#) step 9. The result will be the content encryption key.
9. Extract the **encryptedContent** field of the protected key blob structure, and separate the ICV (contained in the last 16 bytes) from the cipher text (the rest of the field). Decrypt the cipher text using AES-GCM with the content encryption key obtained in step 8, and verify that the ICV computed during decryption matches the value in the blob. If not, fail and return an error.
10. Parse the result of step 9 based on the value of the **AlgorithmName** field, as specified in section [2.2.11.1.1](#).

## 4 Protocol Examples

### 4.1 Querying a DNS server DWORD property

The following example specifies how to query the value of a DWORD DNS server property. In this example the value of the "LogLevel" property will be read.

The client calls R\_DnssrvQuery2 and provides the following parameters:

- DNS\_RPC\_CURRENT\_CLIENT\_VER as the client version.
- Zero as the settings flag.
- A Unicode string containing the FQDN of the DNS server whose LogLevel property is to be read as the server name.
- NULL as the zone name.
- "LogLevel" as the operation.
- A pointer to a DWORD where the DNS RPC type of the output data will be stored.
- A pointer to a DNSSRV\_RPC\_UNION (section [2.2.1.2.6](#)) structure where the results of the DNS RPC operation will be stored.

The DNS server will return ERROR\_SUCCESS and additionally:

- The data type output value will be set to DNSSRV\_TYPEID\_DWORD.
- The DWORD member of the DNSSRV\_RPC\_UNION output structure will be set to the DNS server version in DNSSRV\_VERSION (section [2.2.4.2.1](#)) format.

### 4.2 Modifying a DNS server DWORD property

The following example specifies how to set the value of a DWORD DNS server property. In this example the value of the "LogLevel" property will be set.

The client formats a DNSSRV\_RPC\_UNION (section [2.2.1.2.6](#)) structure to represent the request by setting the "Dword" member of the union one of the ZONE\_REQUEST\_FILTER values. Use

- The **NameAndParam** member of the DNSSRV\_RPC\_UNION is set to point to a DNS\_RPC\_NAME\_AND\_PARAM (section [2.2.1.2.5](#)) structure (stored on the stack or elsewhere).
- The node name member of this DNS\_RPC\_NAME\_AND\_PARAM structure is set to "LogLevel".
- The **dwParam** member of this DNS\_RPC\_NAME\_AND\_PARAM structure is set to the desired combination of logging level bit flags, formatted as a DWORD. For example, to request logging of all incoming queries with full packet detail one would specify 0x0100E101 "(DNS\_LOG\_LEVEL\_QUERY|DNS\_LOG\_LEVEL\_QUESTIONS|DNS\_LOG\_LEVEL\_RECV|DNS\_LOG\_LEVEL\_UDP|DNS\_LOG\_LEVEL\_TCP|DNS\_LOG\_LEVEL\_FULL\_PACKETS)".

The client calls R\_DnssrvOperation2 and provides the following parameters:

- DNS\_RPC\_CURRENT\_CLIENT\_VER as the client version.
- Zero as the settings flag.

- A Unicode string containing the FQDN of the DNS server whose LogLevel property is to be read as the server name.
- NULL as the zone name.
- Zero as the context.
- "ResetDwordProperty" as the operation.
- DNSSRV\_TYPEID\_NAME\_AND\_PARAM as the type ID.
- A pointer to the DNSSRV\_RPC\_UNION structure created above as the RPC data.

The DNS server will return ERROR\_SUCCESS if the operation was successful or a Windows 32 error code if the operation failed.

### 4.3 Creating a New Zone

The following example specifies how to create a new primary zone named "example.com". This zone will be stored in a file (and not in the directory).

The client formats a DNSSRV\_RPC\_UNION structure to represent the request:

- The ZoneCreate member of the DNSSRV\_RPC\_UNION is set to point to a DNS\_RPC\_ZONE\_CREATE\_INFO (section [2.2.5.2.7](#)) structure (stored on the stack or elsewhere). The entire structure should be zeroed out prior to use.
- The **pszZoneName** of this DNS\_RPC\_ZONE\_CREATE\_INFO structure is set to "example.com".
- The **dwZoneType** member of this DNS\_RPC\_ZONE\_CREATE\_INFO structure is set to DNS\_ZONE\_TYPE\_PRIMARY.

The client calls R\_DnssrvOperation2 and provides the following parameters:

- DNS\_RPC\_CURRENT\_CLIENT\_VER as the client version.
- Zero as the settings flag.
- A Unicode string containing the FQDN of the DNS server on which the zone is to be created.
- NULL as the zone name.
- Zero as the context.
- "ZoneCreate" as the operation.
- DNSSRV\_TYPEID\_ZONE\_CREATE as the type ID.
- A pointer to the DNSSRV\_RPC\_UNION structure created above as the RPC data.

The DNS server will return ERROR\_SUCCESS if the operation was successful or a Windows 32 error code if the operation failed.

### 4.4 Enumerating Zones

The following example specifies how to enumerate all zones on the DNS server.



The client formats a `DNSSRV_RPC_UNION` structure to represent the request by setting the `Dword` member of the union to any `ZONE_REQUEST_FILTER` value. Use `ZONE_REQUEST_ALL_ZONES` to enumerate all DNS server zones (with the exception of autocreated zones and the cache zone).

The client calls `R_DnssrvComplexOperation2` and provides the following parameters:

- `DNS_RPC_CURRENT_CLIENT_VER` as the client version.
- Zero as the settings flag.
- A Unicode string containing the FQDN of the DNS server on which the zone is to be enumerated.
- `NULL` as the zone name.
- Zero as the context.
- "EnumZones" as the operation.
- `DNSSRV_TYPEID_DWORD` as in the input type ID.
- A pointer to the `DNSSRV_RPC_UNION` structure created above as the input RPC data.
- A pointer to a `DWORD` where the DNS RPC type of the output data will be stored.
- A pointer to a `DNSSRV_RPC_UNION` structure where the results of the RPC operation will be stored.

The DNS server will return `ERROR_SUCCESS` if the operation was successful or a Windows 32 error code if the operation failed. If the operation was successful then the output type ID will be set to `DNSSRV_TYPEID_ZONE_LIST` and the `ZoneList` member of the output `DNSSRV_RPC_UNION` structure will be set to the zone enumeration results. The client can iterate the elements of `ZoneList.ZoneArray` in the union to examine the zones returned by the enumeration.

Once the client is finished with the zone enumeration result it **MUST**:

- Call `MIDL_user_free` on the `pszZoneName` and `pszDpFqdn` member of each element of the `ZoneList.ZoneArray` in the `DNSSRV_RPC_UNION` structure.
- Call `MIDL_user_free` on each pointer in the `ZoneList.ZoneArray`.
- Call `MIDL_user_free` on the `ZoneList.ZoneArray` pointer itself.

## 4.5 Creating and Deleting a DNS Record

The following example specifies how to create a DNS record representing the IPv4 address "1.2.3.4" for the host named "host1" in the existing primary zone named "example.com". The new record will have a TTL of one hour.

The client formats a [DNS RPC RECORD structure \(section 2.2.2.2.5\)](#), stored on the stack or elsewhere using a buffer at least large enough to hold the `DNSSRV_RPC_RECORD` structure plus the [DNS RPC RECORD DATA](#) in the `Buffer` member, to represent the new record data as follows:

- **wDataLength** is set to the size of the data that will be stored in the `Buffer` member. In this case, because the record data will be a [DNS RPC RECORD A \(section 2.2.2.2.4.1\)](#) structure, the value of **wDataLength** is set to 4.
- **wType** is set to the desired record type (section [2.2.2.1.1](#)), in this case `DNS_TYPE_A`.

- **dwFlags**, **dwSerial**, **dwTimeStamp**, and **dwReserved** are set to zero.
- **dwTtlSeconds** is set to the desired TTL value in seconds, in this case 3600 for one hour.
- The client formats the Buffer member as a DNS\_RPC\_RECORD\_A structure. The byte values 0x01, 0x02, 0x03, and 0x04 are set in the four bytes of memory starting at the offset of Buffer.

The client calls [R\\_DnssrvUpdateRecord2](#) and provides the following parameters:

- [DNS\\_RPC\\_CURRENT\\_CLIENT\\_VER](#) as the client version.
- Zero as the settings flag.
- A Unicode string containing the FQDN of the DNS server on which the operation is to be performed.
- "example.com" as the zone name.
- "host1" as the node name.
- A pointer to the **DNS\_RPC\_RECORD** created above as the *pAddRecord* parameter.
- A NULL pointer as the *pDeleteRecord* pointer.

The DNS server will return ERROR\_SUCCESS if the record was successfully created or a Windows error code on failure.

To delete this DNS record, format a **DNS\_RPC\_RECORD** structure exactly as described above, and call [R\\_DnssrvUpdateRecord2](#) in exactly the same way but pass NULL as the *pAddRecord* parameter and the **DNS\_RPC\_RECORD** pointer as the *pDeleteRecord*.

## 4.6 Creating a Zone Scope

The following example specifies how to create a new zone scope named "example\_americas" in example.com created in section [4.3](#). This operation is allowed only on a file-backed zone.

The client calls [R\\_DnssrvOperation2 \(section 3.1.4.6\)](#) and provides the following parameters:

- [DNS\\_RPC\\_CURRENT\\_CLIENT\\_VER \(section 2.2.1.2.1\)](#) as the client version.
- Zero as the settings flag.
- A Unicode string containing the FQDN of the DNS server on which the zone is to be created.
- "example.com" as the zone name.
- Zero as the context.
- CreateZonescope as the operation.
- DNSSRV\_TYPEID\_ZONE\_SCOPE\_CREATE as the type ID.
- Create a [DNS\\_RPC\\_ZONE\\_SCOPE\\_CREATE\\_INFO\\_V1 \(section 2.2.13.2.2.1\)](#) structure with dwFlags as zero and pwszScopeName as "example\_americas". Pass this structure as RPC data.

The DNS server returns ERROR\_SUCCESS if the operation was successful or a Windows error code if the operation fails.

## 4.7 Deleting a Zone Scope

The following example specifies how to delete an existing **zone scope** named "example\_americas" in example.com created in section 4.6. This operation is allowed only on a file-backed zone.

The client calls [R\\_DnssrvOperation2 \(section 3.1.4.6\)](#) and provides the following parameters:

- [DNS\\_RPC\\_CURRENT\\_CLIENT\\_VER \(section 2.2.1.2.1\)](#) as the client version.
- Zero as the settings flag.
- A Unicode string containing the FQDN of the DNS server on which the zone is to be created.
- "example.com" as the zone name.
- Zero as the context.
- DeleteZonescope as the operation.
- DNSSRV\_TYPEID\_LPWSTR as the type ID.
- Unicode string "example\_americas" as RPC Data.

The DNS server returns ERROR\_SUCCESS if the operation was successful or a Windows error code if the operation failed.

## 4.8 Enumerating Zone Scopes

The following example specifies how to enumerate all zone scopes on a zone example.com on the DNS server.

The client calls [R\\_DnssrvComplexOperation2 \(section 3.1.4.8\)](#) and provides the following parameters:

- [DNS\\_RPC\\_CURRENT\\_CLIENT\\_VER \(section 2.2.1.2.1\)](#) as the client version.
- Zero as the settings flag.
- A Unicode string containing the FQDN of the DNS server on which the zone scopes are to be enumerated.
- "example.com" as the zone name.
- Zero as the context.
- EnumZonescopes as the operation.
- DNSSRV\_TYPEID\_NULL as the input type ID.
- NULL as the input RPC data.
- A pointer to a DWORD where the DNS RPC type of the output data will be stored.
- A pointer to a **DNSSRV\_RPC\_UNION** structure where the results of the RPC operation will be stored.

The DNS server returns ERROR\_SUCCESS if the operation is successful or a Windows error code if the operation fails. If the operation is successful, the output type ID will be set to DNSSRV\_TYPEID\_ZONE\_SCOPE\_ENUM and the ZonescopeList member of the output

**DNSSRV\_RPC\_UNION** structure will be set to the zone scope enumeration results. The client can iterate the elements of `ZonescopeList.ZoneArray` in the union to examine the zones returned by the enumeration.

After the client is finished with the zone scope enumeration result, it MUST call `MIDL_user_free` on each pointer in the `ZonescopeList.ZonescopeArray`.

## 4.9 Creating and Deleting a DNS Record in a Zone Scope

The following example specifies how to create a DNS record representing the IPv4 address "1.2.3.4" for the host named "host1" in the existing zone scope "example\_americas" on the primary zone named "example.com". The new record will have a TTL of one hour.

The client formats a [DNS\\_RPC\\_RECORD structure \(section 2.2.2.2.5\)](#), stored on the stack or elsewhere using a buffer at least large enough to hold the **DNS\_RPC\_RECORD** structure plus the [DNS\\_RPC\\_RECORD\\_DATA \(section 2.2.2.2.4\)](#) in the buffer member, to represent the new record data as follows:

- **wDataLength** is set to the size of the data that will be stored in the buffer member. In this case, because the record data will be a [DNS\\_RPC\\_RECORD\\_A structure \(section 2.2.2.2.4.1\)](#), the value of **wDataLength** is set to 4.
- **wType** is set to the desired record type (section [2.2.2.1.1](#)), in this case `DNS_TYPE_A`.
- **dwFlags**, **dwSerial**, **dwTimeStamp**, and **dwReserved** are set to zero.
- **dwTtlSeconds** is set to the desired TTL value in seconds, in this case 3600 for one hour.
- The client formats the buffer member as a **DNS\_RPC\_RECORD\_A** structure. The byte values 0x01, 0x02, 0x03, and 0x04 are set in the four bytes of memory starting at the offset of the buffer.

The client calls [R\\_DnssrvUpdateRecord3](#) and provides the following parameters:

- [DNS\\_RPC\\_CURRENT\\_CLIENT\\_VER \(section 2.2.1.2.1\)](#) as the client version.
- Zero as the settings flag.
- A Unicode string containing the FQDN of the DNS server on which the operation is to be performed.
- "example.com" as the zone name.
- "host1" as the node name.
- A pointer to the **DNS\_RPC\_RECORD** created above as the *pAddRecord* parameter.
- A NULL pointer as the *pDeleteRecord* pointer.
- "example\_americas" as the zone scope name. If this is provided as NULL, the record is added to the default zone scope maintaining the legacy behavior.

The DNS server returns `ERROR_SUCCESS` if the record is successfully created or a Windows error code on failure.

To delete this DNS record, format a **DNS\_RPC\_RECORD** structure exactly as described above, and call [R\\_DnssrvUpdateRecord3](#) (section 3.1.4.11) in exactly the same way, but pass NULL as the *pAddRecord* parameter and the **DNS\_RPC\_RECORD** pointer as the *pDeleteRecord*.

## 5 Security

### 5.1 Security Considerations for Implementers

This Protocol allows any user to establish a connection to the RPC server. The Protocol uses the underlying RPC Protocol to retrieve the identity of the caller that made the method call as specified in [\[MS-RPCE\]](#), section [3.3.3.4.3](#). Clients SHOULD create an authenticated RPC connection. Servers SHOULD use this identity to perform method specific access checks.

#### 5.1.1 Security considerations specific to the DNS Server Management Protocol

DNS data and DNS server operations exposed by the DNS Server Management Protocol SHOULD be protected by access checks based on the identity of the RPC client. DNS server settings, DNS zones, DNS records, and the Application Directory Partition Table MAY each be protected by a different access control list to allow for delegation of administrative control over the DNS server.

Servers implementing the DNS Server Management Protocol SHOULD NOT allow anonymous RPC connections and SHOULD protect DNS access to all data and operations with access control checks based on client identity.

RPC over named pipes SHOULD NOT be used by clients or servers as it is vulnerable to man-in-the-middle attacks. TCP/IP RPC SHOULD be used instead [<335>](#).

The DNS Server Management Protocol does not require clients to request `RPC_C_AUTHN_LEVEL_PKT_PRIVACY` or servers to enforce it. If privacy of DNS management traffic is important implementers should consider using IPsec or another technology to provide encryption of data at a lower layer.

### 5.2 Index of Security Parameters

Security Parameter	Section
<code>RPC_C_AUTHN_GSS_NEGOTIATE</code>	Section <a href="#">2.1.1</a>
<code>RPC_C_AUTHN_GSS_KERBEROS</code>	Section <a href="#">2.1.1</a>
<code>RPC_C_AUTHN_WINNT</code>	Section <a href="#">2.1.1</a>
<code>RPC_C_IMP_LEVEL_IMPERSONATE</code>	Section <a href="#">2.1.2</a>

## 6 Appendix A: Full IDL

For ease of implementation the full IDL is provided below, where "ms-rpce.idl" refers to the IDL found in [\[MS-RPCE\]](#) Appendix A. The syntax uses the IDL syntax extensions defined in [\[MS-RPCE\]](#) section 2.2.4. For example, as noted in [\[MS-RPCE\]](#) section 2.2.4.9, a pointer\_default declaration is not required and pointer\_default(unique) is assumed.

```
import "ms-dtyp.idl";

typedef struct _DnsStatHeader
{
    DWORD        StatId;
    WORD         wLength;
    BOOLEAN      fClear;
    UCHAR        fReserved;
}DNSSRV_STAT_HEADER, *PDNSSRV_STAT_HEADER;

typedef struct _DnsStat
{
    DNSSRV_STAT_HEADER Header;
    BYTE                Buffer[1];
}DNSSRV_STAT, *PDNSSRV_STAT, *PDNSSRV_STATS;

typedef struct _IP4_ARRAY
{
    DWORD        AddrCount;
    [size_is( AddrCount )]  DWORD  AddrArray[];
}
IP4_ARRAY, *PIP4_ARRAY;

typedef struct _DnsAddr
{
    CHAR MaxSa[32];
    DWORD DnsAddrUserDword[8];
}
DNS_ADDR, *PDNS_ADDR;

typedef struct _DnsAddrArray
{
    DWORD        MaxCount;
    DWORD        AddrCount;
    DWORD        Tag;
    WORD         Family;
    WORD         WordReserved;
    DWORD        Flags;
    DWORD        MatchFlag;
    DWORD        Reserved1;
    DWORD        Reserved2;

    [size_is( AddrCount )]  DNS_ADDR  AddrArray[];
}
DNS_ADDR_ARRAY, *PDNS_ADDR_ARRAY;

//
//  RPC buffer type for returned data
//
```

```

typedef struct _DnssrvRpcBuffer
{
    DWORD                dwLength;
    [size_is(dwLength)] BYTE    Buffer[];
}
DNS_RPC_BUFFER, *PDNS_RPC_BUFFER;

//
// Server data types
//
typedef struct _DnsRpcServerInfoW2K
{
    // version
    // basic configuration flags

    DWORD                dwVersion;
    UCHAR                fBootMethod;
    BOOLEAN              fAdminConfigured;
    BOOLEAN              fAllowUpdate;
    BOOLEAN              fDsAvailable;

    //
    // pointer section
    //

    [string] char *      pszServerName;

    // DS container

    [string] wchar_t *   pszDsContainer;

    // IP interfaces

    PIP4_ARRAY           aipServerAddr;
    PIP4_ARRAY           aipListenAddr;

    // forwarders

    PIP4_ARRAY           aipForwarders;

    // future extensions

    PDWORD               pExtension1;
    PDWORD               pExtension2;
    PDWORD               pExtension3;
    PDWORD               pExtension4;
    PDWORD               pExtension5;

    //
    // DWORD section
    //

    // logging

    DWORD                dwLogLevel;
    DWORD                dwDebugLevel;

    // configuration DWORDs

```

```

DWORD      dwForwardTimeout;
DWORD      dwRpcProtocol;
DWORD      dwNameCheckFlag;
DWORD      cAddressAnswerLimit;
DWORD      dwRecursionRetry;
DWORD      dwRecursionTimeout;
DWORD      dwMaxCacheTtl;
DWORD      dwDsPollingInterval;

// aging / scavenging

DWORD      dwScavengingInterval;
DWORD      dwDefaultRefreshInterval;
DWORD      dwDefaultNoRefreshInterval;

DWORD      dwReserveArray[10];

//
// BYTE section
//
// configuration flags

BOOLEAN    fAutoReverseZones;
BOOLEAN    fAutoCacheUpdate;

// recursion control

BOOLEAN    fRecurseAfterForwarding;
BOOLEAN    fForwardDelegations;
BOOLEAN    fNoRecursion;
BOOLEAN    fSecureResponses;

// lookup control

BOOLEAN    fRoundRobin;
BOOLEAN    fLocalNetPriority;

// BIND compatibility and mimicking

BOOLEAN    fBindSecondaries;
BOOLEAN    fWriteAuthorityNs;

// Bells and whistles

BOOLEAN    fStrictFileParsing;
BOOLEAN    fLooseWildcarding;

// aging / scavenging

BOOLEAN    fDefaultAgingState;
BOOLEAN    fReserveArray[15];
}
DNS_RPC_SERVER_INFO_W2K, *PDNS_RPC_SERVER_INFO_W2K;

typedef struct _DnsRpcServerInfoDotNet
{
    DWORD      dwRpcStructureVersion;
    DWORD      dwReserved0;

```



```

// basic configuration flags

DWORD      dwVersion;
UCHAR      fBootMethod;
BOOLEAN    fAdminConfigured;
BOOLEAN    fAllowUpdate;
BOOLEAN    fDsAvailable;

//
// pointer section
//

[string] char *      pszServerName;

// DS container

[string] wchar_t *  pszDsContainer;

// IP interfaces

PIP4_ARRAY  aipServerAddr;
PIP4_ARRAY  aipListenAddr;

// forwarders

PIP4_ARRAY  aipForwarders;

// logging

PIP4_ARRAY  aipLogFilter;
[string] wchar_t *  pwszLogFilePath;

// Server domain/forest

[string] char *      pszDomainName;          // UTF-8 FQDN
[string] char *      pszForestName;         // UTF-8 FQDN

// Built-in directory partitions

[string] char *      pszDomainDirectoryPartition; // UTF-8 FQDN
[string] char *      pszForestDirectoryPartition; // UTF-8 FQDN

// future extensions

[string] char *      pExtensions[ 6 ];

//
// DWORD section
//

// logging

DWORD      dwLogLevel;
DWORD      dwDebugLevel;

// configuration DWORDs

DWORD      dwForwardTimeout;

```

```

DWORD      dwRpcProtocol;
DWORD      dwNameCheckFlag;
DWORD      cAddressAnswerLimit;
DWORD      dwRecursionRetry;
DWORD      dwRecursionTimeout;
DWORD      dwMaxCacheTtl;
DWORD      dwDsPollingInterval;
DWORD      dwLocalNetPriorityNetMask;

// aging and scavenging

DWORD      dwScavengingInterval;
DWORD      dwDefaultRefreshInterval;
DWORD      dwDefaultNoRefreshInterval;
DWORD      dwLastScavengeTime;

// more logging

DWORD      dwEventLogLevel;
DWORD      dwLogFileMaxSize;

// Active Directory information

DWORD      dwDsForestVersion;
DWORD      dwDsDomainVersion;
DWORD      dwDsDsaVersion;

DWORD      dwReserveArray[ 4 ];

//
// BYTE section
//
// configuration flags

BOOLEAN    fAutoReverseZones;
BOOLEAN    fAutoCacheUpdate;

// recursion control

BOOLEAN    fRecurseAfterForwarding;
BOOLEAN    fForwardDelegations;
BOOLEAN    fNoRecursion;
BOOLEAN    fSecureResponses;

// lookup control

BOOLEAN    fRoundRobin;
BOOLEAN    fLocalNetPriority;

// BIND compatibility and mimicking

BOOLEAN    fBindSecondaries;
BOOLEAN    fWriteAuthorityNs;

// Bells and whistles

BOOLEAN    fStrictFileParsing;
BOOLEAN    fLooseWildcarding;

```

```

// aging \ scavenging

BOOLEAN      fDefaultAgingState;

BOOLEAN      fReserveArray[ 15 ];
}
DNS_RPC_SERVER_INFO_DOTNET, *PDNS_RPC_SERVER_INFO_DOTNET;

typedef struct _DnsRpcServerInfoLonghorn
{
    DWORD          dwRpcStructureVersion;
    DWORD          dwReserved0;

    // basic configuration flags

    DWORD          dwVersion;
    UCHAR          fBootMethod;
    BOOLEAN        fAdminConfigured;
    BOOLEAN        fAllowUpdate;
    BOOLEAN        fDsAvailable;

    //
    // pointer section
    //

    [string] char *      pszServerName;

    // DS container

    [string] wchar_t *   pszDsContainer;

    // IP interfaces

    PDNS_ADDR_ARRAY    aipServerAddrs;
    PDNS_ADDR_ARRAY    aipListenAddrs;

    // forwarders

    PDNS_ADDR_ARRAY    aipForwarders;

    // logging

    PDNS_ADDR_ARRAY    aipLogFilter;
    [string] wchar_t *  pwszLogFilePath;

    // Server domain/forest

    [string] char *     pszDomainName;           // UTF-8 FQDN
    [string] char *     pszForestName;          // UTF-8 FQDN

    // Built-in directory partitions

    [string] char *     pszDomainDirectoryPartition; // UTF-8 FQDN
    [string] char *     pszForestDirectoryPartition; // UTF-8 FQDN

    // future extensions

    [string] char *     pExtensions[ 6 ];
}

```

```

//
//  DWORD section
//

//  logging

DWORD      dwLogLevel;
DWORD      dwDebugLevel;

//  configuration DWORDs

DWORD      dwForwardTimeout;
DWORD      dwRpcProtocol;
DWORD      dwNameCheckFlag;
DWORD      cAddressAnswerLimit;
DWORD      dwRecursionRetry;
DWORD      dwRecursionTimeout;
DWORD      dwMaxCacheTtl;
DWORD      dwDsPollingInterval;
DWORD      dwLocalNetPriorityNetMask;

//  aging and scavenging

DWORD      dwScavengingInterval;
DWORD      dwDefaultRefreshInterval;
DWORD      dwDefaultNoRefreshInterval;
DWORD      dwLastScavengeTime;

//  more logging

DWORD      dwEventLogLevel;
DWORD      dwLogFileMaxSize;

//  Active Directory information

DWORD      dwDsForestVersion;
DWORD      dwDsDomainVersion;
DWORD      dwDsDsaVersion;
BOOLEAN    fReadOnlyDC;

DWORD      dwReserveArray[ 3 ];

//
//  BYTE section
//
//  configuration flags

BOOLEAN    fAutoReverseZones;
BOOLEAN    fAutoCacheUpdate;

//  recursion control

BOOLEAN    fRecurseAfterForwarding;
BOOLEAN    fForwardDelegations;
BOOLEAN    fNoRecursion;
BOOLEAN    fSecureResponses;

//  lookup control

```

```

    BOOLEAN    fRoundRobin;
    BOOLEAN    fLocalNetPriority;

    // BIND compatibility and mimicking

    BOOLEAN    fBindSecondaries;
    BOOLEAN    fWriteAuthorityNs;

    // Bells and whistles

    BOOLEAN    fStrictFileParsing;
    BOOLEAN    fLooseWildcarding;

    // aging \ scavenging

    BOOLEAN    fDefaultAgingState;

    BOOLEAN    fReserveArray[ 15 ];
}
DNS_RPC_SERVER_INFO_LONGHORN, *PDNS_RPC_SERVER_INFO_LONGHORN, DNS_RPC_SERVER_INFO,
*PDNS_RPC_SERVER_INFO;

typedef struct _DnssrvRpcForwardersW2K
{
    DWORD        fRecurseAfterForwarding;
    DWORD        dwForwardTimeout;
    PIP4_ARRAY   aipForwarders;
}
DNS_RPC_FORWARDERS_W2K, *PDNS_RPC_FORWARDERS_W2K;

typedef struct _DnssrvRpcForwardersDotNet
{
    DWORD        dwRpcStructureVersion;
    DWORD        dwReserved0;
    DWORD        fRecurseAfterForwarding;
    DWORD        dwForwardTimeout;
    PIP4_ARRAY   aipForwarders;
}
DNS_RPC_FORWARDERS_DOTNET, *PDNS_RPC_FORWARDERS_DOTNET;

typedef struct _DnssrvRpcForwardersLonghorn
{
    DWORD        dwRpcStructureVersion;
    DWORD        dwReserved0;
    DWORD        fRecurseAfterForwarding;
    DWORD        dwForwardTimeout;
    PDNS_ADDR_ARRAY   aipForwarders;
}
DNS_RPC_FORWARDERS_LONGHORN, *PDNS_RPC_FORWARDERS_LONGHORN, DNS_RPC_FORWARDERS,
*PDNS_RPC_FORWARDERS;

//
// Basic zone data
//

//typedef struct _DnssrvRpcZoneFlags
//{
//    DWORD    Paused            : 1;
//    DWORD    Shutdown         : 1;

```

```

//  DWORD  Reverse      : 1;
//  DWORD  AutoCreated  : 1;
//  DWORD  DsIntegrated : 1;
//  DWORD  Aging        : 1;
//  DWORD  Update       : 2;
//  DWORD  ReadOnly     : 1;
//  DWORD  Unused       : 23;
//}
//DNS_RPC_ZONE_FLAGS, *PDNS_RPC_ZONE_FLAGS;

typedef DWORD DNS_RPC_ZONE_FLAGS;

typedef struct _DnssrvRpcZoneW2K
{
    [string] wchar_t *    pszZoneName;
    DNS_RPC_ZONE_FLAGS   Flags;
    UCHAR               ZoneType;
    UCHAR               Version;
}
DNS_RPC_ZONE_W2K, *PDNS_RPC_ZONE_W2K;

typedef struct _DnssrvRpcZoneDotNet
{
    DWORD               dwRpcStructureVersion;
    DWORD               dwReserved0;
    [string] wchar_t *  pszZoneName;
    DNS_RPC_ZONE_FLAGS Flags;
    UCHAR               ZoneType;
    UCHAR               Version;
    DWORD               dwDpFlags;
    [string] char *     pszDpFqdn;
}
DNS_RPC_ZONE_DOTNET, *PDNS_RPC_ZONE_DOTNET, DNS_RPC_ZONE, *PDNS_RPC_ZONE;

//
// Zone enumeration
//

typedef struct _DnssrvRpcZoneListW2K
{
    [range(0,500000)]   DWORD               dwZoneCount;
    [size_is(dwZoneCount)] PDNS_RPC_ZONE_W2K ZoneArray[];
}
DNS_RPC_ZONE_LIST_W2K, *PDNS_RPC_ZONE_LIST_W2K;

typedef struct _DnssrvRpcZoneListDotNet
{
    DWORD               dwRpcStructureVersion;
    DWORD               dwReserved0;
    [range(0,500000)]   DWORD               dwZoneCount;
    [size_is(dwZoneCount)] PDNS_RPC_ZONE_DOTNET ZoneArray[];
}
DNS_RPC_ZONE_LIST_DOTNET, *PDNS_RPC_ZONE_LIST_DOTNET, DNS_RPC_ZONE_LIST, *PDNS_RPC_ZONE_LIST;

//
// DNSSEC trust point enumeration
//

```

```

typedef enum
{
    TRUSTPOINT_STATE_INITIALIZED,
    TRUSTPOINT_STATE_DSPENDING,
    TRUSTPOINT_STATE_ACTIVE,
    TRUSTPOINT_STATE_DELETE_PENDING,
    TRUSTPOINT_STATE_DELETED
}
TRUSTPOINT_STATE, *PTRUSTPOINT_STATE;

typedef enum
{
    TRUSTANCHOR_STATE_INITIALIZED,
    TRUSTANCHOR_STATE_DSPENDING,
    TRUSTANCHOR_STATE_DSINVALID,
    TRUSTANCHOR_STATE_ADDPEND,
    TRUSTANCHOR_STATE_VALID,
    TRUSTANCHOR_STATE_MISSING,
    TRUSTANCHOR_STATE_REVOKED,
    TRUSTANCHOR_STATE_DELETED
}
TRUSTANCHOR_STATE, *PTRUSTANCHOR_STATE;

typedef struct _DnssrvRpcTrustPoint
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;

    [string] char *      pszTrustPointName;

    TRUSTPOINT_STATE     eTrustPointState;

    __int64              i64LastActiveRefreshTime;
    __int64              i64NextActiveRefreshTime;
    __int64              i64LastSuccessfulActiveRefreshTime;
    DWORD                dwLastActiveRefreshResult;
    DWORD                dwReserved;
}
DNS_RPC_TRUST_POINT, *PDNS_RPC_TRUST_POINT;

typedef struct _DnssrvRpcTrustPointList
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;

    [range(0,500000)]    DWORD                dwTrustPointCount;
    [size_is(dwTrustPointCount)] PDNS_RPC_TRUST_POINT TrustPointArray[];
}
DNS_RPC_TRUST_POINT_LIST, *PDNS_RPC_TRUST_POINT_LIST;

//
// Signing Key Descriptor (SKD) definitions for online signing
//
// Each SKD has a set of static configuration represented by a DNS_RPC_SKD
// structure that defines how the SKD is used to generate keys and sign
// records in the zone. Each SKD is uniquely identified by its GUID field,
// which is dynamically generated by the DNS server when the SKD is created.
// Information on the dynamic state of the SKD, including identifiers for
// the current keys associated with the SKD, is represented using the

```

```

// DNS_RPC_SKD_STATE structure.
//
// Guid: A unique identifier for this SKD.
//
// pwszKeyStorageProvider: The KSP to use to generate keys.
//
// fStoreKeysInDirectory: If TRUE, keys will be stored on the zone object
//   in Active Directory. If FALSE, keys will be wrapped in self-signed
//   certificates in the local machine store.
//
// fIsKSK: TRUE if this SKD represents a Key Signing Key, FALSE for a Zone
//   Signing Key.
//
// bSigningAlgorithm: DNSSEC signature generation algorithm.
//
// dwKeyLength: Length, in bits, of keys. Ignored if the algorithm specified
//   by bSigningAlgorithm does not support variable-length keys.
//
// dwInitialRolloverOffset: Amount of time, in seconds, to delay the first
//   scheduled key rollover. This allows for key rollovers to be staggered.
//
// dwDNSKEYSignatureValidityPeriod: Amount of time, in seconds, that signatures
//   covering DNSKEY record sets should be valid.
//
// dwDSSignatureValidityPeriod: Amount of time, in seconds, that signatures
//   covering DS record sets should be valid.
//
// dwStandardSignatureValidityPeriod: Amount of time, in seconds, that signatures
//   covering all other record sets should be valid.
//
// dwRolloverType: Key rollover method (see [RFC4641]).
//
// dwRolloverPeriod: Amount of time, in seconds, between scheduled key rollovers
//   or 0xFFFFFFFF to disable automatic key rollover.
//
// dwNextRolloverAction: The action to take for this SKD on the next key
//   manual or scheduled key rollover event.
//
typedef struct _DnsSrvRpcSigningKeyDescriptor
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;

    GUID                Guid;
    [string] wchar_t*   pwszKeyStorageProvider;
    BOOL                fStoreKeysInDirectory;
    BOOL                fIsKSK;
    BYTE                bSigningAlgorithm;
    DWORD               dwKeyLength;
    DWORD               dwInitialRolloverOffset;
    DWORD               dwDNSKEYSignatureValidityPeriod;
    DWORD               dwDSSignatureValidityPeriod;
    DWORD               dwStandardSignatureValidityPeriod;
    DWORD               dwRolloverType;
    DWORD               dwRolloverPeriod;
    DWORD               dwNextRolloverAction;
    DWORD               dwReserved;
}

```



```

DNS_RPC_SKD, *PDNS_RPC_SKD;

//
// For dwActiveKeyScope, dwStandByKeyScope, dwNextKeyScope
// KeySignScope defines the scope of the key in a signed zone.
//

typedef enum _KeySignScope
{
    //
    // SIGN_SCOPE_DEFAULT specifies that the scope will be determined by the
    // key flags. That is, KSK is used to sign DNSKEY set only and ZSK is
    // used to sign all record sets.
    //
    SIGN_SCOPE_DEFAULT,

    //
    // A key with SIGN_SCOPE_DNSKEY_ONLY scope will only be used to sign DNSKEY
    // set. This gives the user a way to override the ZSK flag when signing.
    //
    SIGN_SCOPE_DNSKEY_ONLY,

    //
    // A key with SIGN_SCOPE_ALL_RECORDS scope will be used to sign all record
    // sets. This gives the user a way to override the KSK flag when signing.
    //
    SIGN_SCOPE_ALL_RECORDS,

    //
    // A key with SIGN_SCOPE_ADD_ONLY scope will be added to zone file, but
    // not used for signing. It is for user to pre-publish a key.
    //
    SIGN_SCOPE_ADD_ONLY,

    //
    // A key with SIGN_SCOPE_DO_NOT_PUBLISH will not be added to the zone file,
    // and will not be used for signing. It's used to remove a key.
    //
    SIGN_SCOPE_DO_NOT_PUBLISH,

    //
    // A key with SIGN_SCOPE_REVOKED will be published to the zone with its
    // "revoked" bit set to TRUE, and will be used to sign the DNSKEY set only.
    //
    SIGN_SCOPE_REVOKED
} KeySignScope;

//
// Signing Key Descriptor (SKD) dynamic state extended
//
//
// Guid: Unique identifier for the SKD
//

```

```

// dwCurrentRollState: Internal Status for Rollover.
//
// fManualTrigger: TRUE if Rollover was triggered manually.
//
// dwPreRollEventFired: Specifies which Pre Rollover Event has been fired
//
// ftNextKeyGenerationTime: Time at which Next Key was added to the zone.
//
// dwRevokedOrSwappedDnskeysLength: Length of the Revoked or Swaped DNSKEY RRSet
//
// pRevokedOrSwappedDnskeysBuffer: Pre-Signed DNSKEY RRSet for ZSK swap
//
// dwFinalDnskeysLength: Length of Pre-Signed DNSKEY RRSet
//
// pFinalDnskeys: Pre-Signed DNSKEY RRSet Post Rollover
//
// eActiveKeyScope: Signing key Scope for the SKD's active key.
//
// eStandbyKeyScope: Signing key Scope for the SKD's standby key.
//
// eNextKeyScope: Signing key Scope for the SKD's next key.
//
typedef struct _DnsRpcSigningKeyDescriptorStateEx
{
    DWORD dwRpcStructureVersion;
    DWORD dwReserved0;
    GUID Guid;
    DWORD dwCurrentRollState;
    DWORD fManualTrigger;
    DWORD dwPreRollEventFired;
    FILETIME ftNextKeyGenerationTime;
    DWORD dwRevokedOrSwappedDnskeysLength;
    PBYTE pRevokedOrSwappedDnskeysBuffer;
    DWORD dwFinalDnskeysLength;
    PBYTE pFinalDnskeys;
    KeySignScope eActiveKeyScope;
    KeySignScope eStandByKeyScope;
    KeySignScope eNextKeyScope;
}
DNS_RPC_SKD_STATE_EX, *PDNS_RPC_SKD_STATE_EX;

//
// Signing Key Descriptor (SKD) dynamic state
//
// The static configuration for each SKD is represented by a DNS_RPC_SKD
// structure. In addition to these static configuration parameters, each SKD
// also has dynamic configuration associated with it, including pointers to
// the current keys for the SKD and other information. The collection of
// dynamic configuration information for each SKD is represented by a
// DNS_RPC_SKD_STATE structure.
//
// A signing key pointer string contains a unique identifier for a signing
// key. For keys stored in the local machine certificate store, this
// identifier is the certificate serial number. For keys stored in Active
// Directory, this identifier is a GUID. The matching GUID for a key may be
// found in one of the values of the key attribute on the zone object along
// with a binary exported representation of the key. Note that the GUID for
// a key is not directly related to the GUID for the SKD.

```

```

//
// Guid: Unique identifier for the SKD
//
// ftLastRolloverTime: The time at which the last rollover event was performed.
//
// ftNextRolloverTime: The time at which the next rollover action must take place.
//
// dwState: The current state of this SKD.
//
// dwCurrentRolloverStatus: Current rollover status of this SKD.
//
// pwszActiveKey: Signing key pointer string for the SKD's active key.
//
// pwszStandbyKey: Signing key pointer string for the SKD's standby key.
//
// pwszNextKey: Signing key pointer string for the SKD's next key. This key
// will be used during the next key rollover event.
//

typedef struct _DnsRpcSigningKeyDescriptorState
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;

    GUID                Guid;
    FILETIME             ftLastRolloverTime;
    FILETIME             ftNextRolloverTime;
    DWORD                dwState;
    DWORD                dwCurrentRolloverStatus;
    [string] wchar_t *  pwszActiveKey;
    [string] wchar_t *  pwszStandbyKey;
    [string] wchar_t *  pwszNextKey;
    DWORD                dwReserved;
}
DNS_RPC_SKD_STATE, *PDNS_RPC_SKD_STATE;

//
// Signing Key Descriptor (SKD) Properties for a zone.
//
// pSkd      Static properties of a Zone SKD. This is a DNS_RPC_SKD structure
//
// psKdState Dynamic properties of a Zone SKD. This is a DNS_RPC_SKD_STATE structure
//
// psKdStateEx Extended Dynamic properties of a Zone SKD. This is a DNS_RPC_SKD_STATE_EX
// structure

typedef struct _DnssrvRpcZoneSKDSettings
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;
    PDNS_RPC_SKD        pSkd;
    PDNS_RPC_SKD_STATE  pSkdState;
    PDNS_RPC_SKD_STATE_EX pSkdStateEx;
} DNS_RPC_ZONE_SKD, *PDNS_RPC_ZONE_SKD;

//
// DNSSEC Settings of a Zone

```

```

typedef struct _DnssrvRpcZoneDnssecSettings
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;
    DWORD                fIsSigned;
    DWORD                fSignWithNSEC3;
    DWORD                fNSEC3OptOut;
    DWORD                dwMaintainTrustAnchor;
    DWORD                fParentHasSecureDelegation;
    DWORD                dwDSRecordAlgorithms;
    DWORD                fRFC5011KeyRollovers;
    BYTE                bNSEC3HashAlgorithm;
    BYTE                bNSEC3RandomSaltLength;
    WORD                wNSEC3IterationCount;
    LPWSTR              pwszNSEC3UserSalt;
    DWORD                dwDNSKEYRecordSetTtl;
    DWORD                dwDSRecordSetTtl;
    DWORD                dwSignatureInceptionOffset;
    DWORD                dwSecureDelegationPollingPeriod;
    DWORD                dwPropagationTime;
    DWORD                cbNSEC3CurrentSaltLength;
    PBYTE               pbNSEC3CurrentSalt;
    GUID                CurrentRollingSKDGuid;
    DWORD                dwBufferLength;
    PBYTE               pBuffer;
    DWORD                dwCount;
    PDNS_RPC_ZONE_SKD   pZoneSkdArray[1] ;
} DNS_RPC_ZONE_DNSSEC_SETTINGS, *PDNS_RPC_ZONE_DNSSEC_SETTINGS;

```

```

typedef struct _DnssrvRpcTrustAnchor
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;
    WORD                wTrustAnchorType;
    WORD                wKeyTag;
    WORD                wRRLength;
    TRUSTANCHOR_STATE   eTrustAnchorState;
    __int64             i64EnteredStateTime;
    __int64             i64NextStateTime;
    DWORD                dwReserved;
    [size_is(wRRLength)]
    BYTE                RRData[];
}
DNS_RPC_TRUST_ANCHOR, *PDNS_RPC_TRUST_ANCHOR;

```

```

typedef struct _DnssrvRpcTrustAnchorList
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;

    [range(0,500000)]    DWORD                dwTrustAnchorCount;
    [size_is(dwTrustAnchorCount)] PDNS_RPC_TRUST_ANCHOR TrustAnchorArray[];
}
DNS_RPC_TRUST_ANCHOR_LIST, *PDNS_RPC_TRUST_ANCHOR_LIST;

```

```

//
// Directory partition enumeration and info
//

```

```

typedef struct _DnssrvRpcDirectoryPartitionEnum
{
    DWORD          dwRpcStructureVersion;
    DWORD          dwReserved0;

    [string] char * pszDpFqdn;
    DWORD          dwFlags;
    DWORD          dwZoneCount;
}
DNS_RPC_DP_ENUM, *PDNS_RPC_DP_ENUM;

typedef struct _DnssrvRpcDirectoryPartitionList
{
    DWORD          dwRpcStructureVersion;
    DWORD          dwReserved0;
    [range(0,5000)] DWORD          dwDpCount;
    [size_is(dwDpCount)] PDNS_RPC_DP_ENUM DpArray[];
}
DNS_RPC_DP_LIST, *PDNS_RPC_DP_LIST;

typedef struct _DnssrvRpcDirectoryPartitionReplica
{
    [string] wchar_t * pszReplicaDn;
}
DNS_RPC_DP_REPLICA, *PDNS_RPC_DP_REPLICA;

typedef struct _DnssrvRpcDirectoryPartition
{
    DWORD          dwRpcStructureVersion;
    DWORD          dwReserved0;

    [string] char * pszDpFqdn;
    [string] wchar_t * pszDpDn;
    [string] wchar_t * pszCrDn;
    DWORD          dwFlags;
    DWORD          dwZoneCount;
    DWORD          dwState;

    DWORD          dwReserved[ 3 ];
    [string] wchar_t * pwszReserved[ 3 ];
    [range(0,10000)] DWORD          dwReplicaCount;
    [size_is(dwReplicaCount)] PDNS_RPC_DP_REPLICA ReplicaArray[];
}
DNS_RPC_DP_INFO, *PDNS_RPC_DP_INFO;

//
// Enlist (or create) directory partition
//

typedef struct _DnssrvRpcEnlistDirPart
{
    DWORD          dwRpcStructureVersion;
    DWORD          dwReserved0;
    [string] char * pszDpFqdn; // UTF8
    DWORD          dwOperation;
}
DNS_RPC_ENLIST_DP, *PDNS_RPC_ENLIST_DP;

```

```

//
// Zone export
//

typedef struct _DnssrvRpcZoneExport
{
    DWORD          dwRpcStructureVersion;
    DWORD          dwReserved0;

    [string] char * pszZoneExportFile;
}
DNS_RPC_ZONE_EXPORT_INFO, *PDNS_RPC_ZONE_EXPORT_INFO;

//
// Zone property data
//

typedef struct _DnssrvRpcZoneSecondariesW2K
{
    DWORD          fSecureSecondaries;
    DWORD          fNotifyLevel;
    PIP4_ARRAY     aipSecondaries;
    PIP4_ARRAY     aipNotify;
}
DNS_RPC_ZONE_SECONDARIES_W2K, *PDNS_RPC_ZONE_SECONDARIES_W2K;

typedef struct _DnssrvRpcZoneSecondariesDotNet
{
    DWORD          dwRpcStructureVersion;
    DWORD          dwReserved0;

    DWORD          fSecureSecondaries;
    DWORD          fNotifyLevel;
    PIP4_ARRAY     aipSecondaries;
    PIP4_ARRAY     aipNotify;
}
DNS_RPC_ZONE_SECONDARIES_DOTNET, *PDNS_RPC_ZONE_SECONDARIES_DOTNET;

typedef struct _DnssrvRpcZoneSecondariesLonghorn
{
    DWORD          dwRpcStructureVersion;
    DWORD          dwReserved0;

    DWORD          fSecureSecondaries;
    DWORD          fNotifyLevel;
    PDNS_ADDR_ARRAY aipSecondaries;
    PDNS_ADDR_ARRAY aipNotify;
}
DNS_RPC_ZONE_SECONDARIES_LONGHORN, *PDNS_RPC_ZONE_SECONDARIES_LONGHORN,
DNS_RPC_ZONE_SECONDARIES, *PDNS_RPC_ZONE_SECONDARIES;

typedef struct _DnssrvRpcZoneDatabaseW2K
{
    DWORD          fDsIntegrated;
    [string] char * pszFileName;
}
DNS_RPC_ZONE_DATABASE_W2K, *PDNS_RPC_ZONE_DATABASE_W2K;

typedef struct _DnssrvRpcZoneDatabaseDotNet

```

```

{
    DWORD        dwRpcStructureVersion;
    DWORD        dwReserved0;

    DWORD        fDsIntegrated;
    [string] char * pszFileName;
}
DNS_RPC_ZONE_DATABASE_DOTNET, *PDNS_RPC_ZONE_DATABASE_DOTNET, DNS_RPC_ZONE_DATABASE,
*PDNS_RPC_ZONE_DATABASE;

typedef struct _DnsSrvRpcZoneChangePartition
{
    DWORD        dwRpcStructureVersion;
    DWORD        dwReserved0;
    [string] char * pszDestPartition;
}
DNS_RPC_ZONE_CHANGE_DP, *PDNS_RPC_ZONE_CHANGE_DP;

typedef struct _DnsRpcZoneInfoW2K
{
    [string] char * pszZoneName;
    DWORD        dwZoneType;
    DWORD        fReverse;
    DWORD        fAllowUpdate;
    DWORD        fPaused;
    DWORD        fShutdown;
    DWORD        fAutoCreated;

    // Database info
    DWORD        fUseDatabase;
    [string] char * pszDataFile;

    // Masters
    PIP4_ARRAY   aipMasters;

    // Secondaries
    DWORD        fSecureSecondaries;
    DWORD        fNotifyLevel;
    PIP4_ARRAY   aipSecondaries;
    PIP4_ARRAY   aipNotify;

    // WINS or NetBIOS lookup
    DWORD        fUseWins;
    DWORD        fUseNbstat;

    // Aging
    DWORD        fAging;
    DWORD        dwNoRefreshInterval;
    DWORD        dwRefreshInterval;
    DWORD        dwAvailForScavengeTime;
    PIP4_ARRAY   aipScavengeServers;

    // save some space, just in case
    // avoid versioning issues if possible
    DWORD        pvReserved1;
    DWORD        pvReserved2;
    DWORD        pvReserved3;
    DWORD        pvReserved4;
}

```

```
DNS_RPC_ZONE_INFO_W2K, *PDNS_RPC_ZONE_INFO_W2K;
```

```
typedef struct _DnsRpcZoneInfoDotNet
{
    DWORD          dwRpcStructureVersion;
    DWORD          dwReserved0;

    [string] char * pszZoneName;
    DWORD          dwZoneType;
    DWORD          fReverse;
    DWORD          fAllowUpdate;
    DWORD          fPaused;
    DWORD          fShutdown;
    DWORD          fAutoCreated;

    // Database info
    DWORD          fUseDatabase;
    [string] char * pszDataFile;

    // Masters
    PIP4_ARRAY    aipMasters;

    // Secondaries
    DWORD          fSecureSecondaries;
    DWORD          fNotifyLevel;
    PIP4_ARRAY    aipSecondaries;
    PIP4_ARRAY    aipNotify;

    // WINS or NetBIOS lookup
    DWORD          fUseWins;
    DWORD          fUseNbstat;

    // Aging
    DWORD          fAging;
    DWORD          dwNoRefreshInterval;
    DWORD          dwRefreshInterval;
    DWORD          dwAvailForScavengeTime;
    PIP4_ARRAY    aipScavengeServers;

    // Forwarder zones
    DWORD          dwForwarderTimeout;
    DWORD          fForwarderSlave;

    // Stub zones
    PIP4_ARRAY    aipLocalMasters;

    // Directory partition
    DWORD          dwDpFlags;
    [string] char * pszDpFqdn;
    [string] wchar_t * pwszZoneDn;

    // Xfr time information
    DWORD          dwLastSuccessfulSoaCheck;
    DWORD          dwLastSuccessfulXfr;

    // save some space, just in case
    DWORD          dwReserved1;
    DWORD          dwReserved2;
    DWORD          dwReserved3;
}
```



```

        DWORD          dwReserved4;
        DWORD          dwReserved5;
        [string] char *    pReserved1;
        [string] char *    pReserved2;
        [string] char *    pReserved3;
        [string] char *    pReserved4;
    }
    DNS_RPC_ZONE_INFO_DOTNET, *PDNS_RPC_ZONE_INFO_DOTNET;

typedef struct _DnsRpcZoneInfoLonghorn
{
    DWORD          dwRpcStructureVersion;
    DWORD          dwReserved0;

    [string] char *    pszZoneName;
    DWORD          dwZoneType;
    DWORD          fReverse;
    DWORD          fAllowUpdate;
    DWORD          fPaused;
    DWORD          fShutdown;
    DWORD          fAutoCreated;

    // Database info
    DWORD          fUseDatabase;
    [string] char *    pszDataFile;

    // Masters
    PDNS_ADDR_ARRAY aipMasters;

    // Secondaries
    DWORD          fSecureSecondaries;
    DWORD          fNotifyLevel;
    PDNS_ADDR_ARRAY aipSecondaries;
    PDNS_ADDR_ARRAY aipNotify;

    // WINS or NetBIOS lookup
    DWORD          fUseWins;
    DWORD          fUseNbstat;

    // Aging
    DWORD          fAging;
    DWORD          dwNoRefreshInterval;
    DWORD          dwRefreshInterval;
    DWORD          dwAvailForScavengeTime;
    PDNS_ADDR_ARRAY aipScavengeServers;

    // Forwarder zones
    DWORD          dwForwarderTimeout;
    DWORD          fForwarderSlave;

    // Stub zones
    PDNS_ADDR_ARRAY aipLocalMasters;

    // Directory partition
    DWORD          dwDpFlags;
    [string] char *    pszDpFqdn;
    [string] wchar_t *    pwszZoneDn;

    // Xfr time information

```

```

        DWORD                dwLastSuccessfulSoaCheck;
        DWORD                dwLastSuccessfulXfr;

        DWORD                fQueuedForBackgroundLoad;
        DWORD                fBackgroundLoadInProgress;
        BOOL                 fReadOnlyZone;

        // Additional zone transfer information
        DWORD                dwLastXfrAttempt;
        DWORD                dwLastXfrResult;
    }
    DNS_RPC_ZONE_INFO_LONGHORN, *PDNS_RPC_ZONE_INFO_LONGHORN, DNS_RPC_ZONE_INFO,
    *PDNS_RPC_ZONE_INFO;

    //
    // Zone create data
    //

typedef struct _DnsRpcZoneCreateInfo
{
    [string] char *         pszZoneName;
    DWORD                 dwZoneType;
    DWORD                 fAllowUpdate;
    DWORD                 fAging;
    DWORD                 dwFlags;

    // Database info

    [string] char *         pszDataFile;
    DWORD                 fDsIntegrated;
    DWORD                 fLoadExisting;

    // Admin name (if auto-create SOA)

    [string] char *         pszAdmin;

    // Masters (if secondary)

    PIP4_ARRAY             aipMasters;

    // Secondaries

    PIP4_ARRAY             aipSecondaries;
    DWORD                 fSecureSecondaries;
    DWORD                 fNotifyLevel;

    // Reserve some space to avoid versioning issues

    [string] char *         pvReserved1;
    [string] char *         pvReserved2;
    [string] char *         pvReserved3;
    [string] char *         pvReserved4;
    [string] char *         pvReserved5;
    [string] char *         pvReserved6;
    [string] char *         pvReserved7;
    [string] char *         pvReserved8;

    DWORD                 dwReserved1;
    DWORD                 dwReserved2;

```

```

        DWORD        dwReserved3;
        DWORD        dwReserved4;
        DWORD        dwReserved5;
        DWORD        dwReserved6;
        DWORD        dwReserved7;
        DWORD        dwReserved8;
    }
    DNS_RPC_ZONE_CREATE_INFO_W2K, *PDNS_RPC_ZONE_CREATE_INFO_W2K;

typedef struct _DnsRpcZoneCreateInfoDotNet
{
    DWORD        dwRpcStructureVersion;
    DWORD        dwReserved0;

    [string] char *        pszZoneName;
    DWORD        dwZoneType;
    DWORD        fAllowUpdate;
    DWORD        fAging;
    DWORD        dwFlags;

    // Database info

    [string] char *        pszDataFile;
    DWORD        fDsIntegrated;
    DWORD        fLoadExisting;

    // Admin name (if auto-create SOA)

    [string] char *        pszAdmin;

    // Masters (if secondary)

    PIP4_ARRAY    aipMasters;

    // Secondaries

    PIP4_ARRAY    aipSecondaries;
    DWORD        fSecureSecondaries;
    DWORD        fNotifyLevel;

    // Forwarder zones

    DWORD        dwTimeout;
    DWORD        fRecurseAfterForwarding;

    // Directory partition

    DWORD        dwDpFlags;        // specify built-in DP or
    [string] char *        pszDpFqdn;        // UTF8 FQDN of partition

    DWORD        dwReserved[ 32 ];
}
DNS_RPC_ZONE_CREATE_INFO_DOTNET, *PDNS_RPC_ZONE_CREATE_INFO_DOTNET;

typedef struct _DnsRpcZoneCreateInfoLonghorn
{
    DWORD        dwRpcStructureVersion;
    DWORD        dwReserved0;

```

```

    [string] char *          pszZoneName;
    DWORD                dwZoneType;
    DWORD                fAllowUpdate;
    DWORD                fAging;
    DWORD                dwFlags;

    // Database info

    [string] char *          pszDataFile;
    DWORD                fDsIntegrated;
    DWORD                fLoadExisting;

    // Admin name (if auto-create SOA)

    [string] char *          pszAdmin;

    // Masters (if secondary)

    PDNS_ADDR_ARRAY      aipMasters;

    // Secondaries

    PDNS_ADDR_ARRAY      aipSecondaries;
    DWORD                fSecureSecondaries;
    DWORD                fNotifyLevel;

    // Forwarder zones

    DWORD                dwTimeout;
    DWORD                fRecurseAfterForwarding;

    // Directory partition

    DWORD                dwDpFlags;          // specify built-in DP or
    [string] char *          pszDpFqdn;      // UTF8 FQDN of partition

    DWORD                dwReserved[ 32 ];
}
DNS_RPC_ZONE_CREATE_INFO_LONGHORN, *PDNS_RPC_ZONE_CREATE_INFO_LONGHORN,
DNS_RPC_ZONE_CREATE_INFO, *PDNS_RPC_ZONE_CREATE_INFO;

//
// SKD collection used to enumerate the SKDs for a zone
//

typedef struct _DnssrvRpcZoneSigningKeyList
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;

    [range(0,1000)]      DWORD                dwCount;
    [size_is(dwCount)]  PDNS_RPC_SKD        SkdArray[];
}
DNS_RPC_SKD_LIST, *PDNS_RPC_SKD_LIST;

```

```

//
// Online signing validation output structure: this structure is an output
// of the validation operation. If the zone signing parameters are found to
// contain an error or a problem, this structure specifies the GUID of the SKD
// (if any) and the specific key (if any) and the error code.
//

typedef struct _DnssrvRpcSigningValidationError
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;

    GUID                guidSKD;
    [string] wchar_t *  pwszSigningKeyPointerString;
    DWORD                dwExtendedError;
    DWORD                dwReserved;
}
DNS_RPC_SIGNING_VALIDATION_ERROR, *PDNS_RPC_SIGNING_VALIDATION_ERROR;

typedef struct _DnsRpcAutoConfigureLonghorn
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;

    DWORD                dwAutoConfigFlags;
    DWORD                dwReserved1;
    [string] char *      pszNewDomainName;
}
DNS_RPC_AUTOCONFIGURE, *PDNS_RPC_AUTOCONFIGURE;

//
// EnumZones2 filter specification
//

typedef struct _DnsRpcEnumZonesFilter
{
    DWORD                dwRpcStructureVersion;
    DWORD                dwReserved0;

    DWORD                dwFilter;
    [string] char *      pszPartitionFqdn;
    [string] char *      pszQueryString;

    [string] char *      pszReserved[ 6 ];
}
DNS_RPC_ENUM_ZONES_FILTER, *PDNS_RPC_ENUM_ZONES_FILTER;

//
// RPC record structure
//

typedef struct _DnssrvRpcRecord
{
    WORD                wDataLength;
    WORD                wType;
    DWORD                dwFlags;
    DWORD                dwSerial;
}

```

```

        DWORD        dwTtlSeconds;
        DWORD        dwTimeStamp;
        DWORD        dwReserved;

        [size_is(wDataLength)] BYTE    Buffer[];
    }
    DNS_RPC_RECORD, *PDNS_RPC_RECORD,
    DNS_FLAT_RECORD, *PDNS_FLAT_RECORD;

//
// These RPC structures have no version because they are simple
// are they are explicitly defined by their names.
//

typedef struct _DnsSrvRpcNameAndParam
{
    DWORD        dwParam;
    [string] char *    pszNodeName;
}
DNS_RPC_NAME_AND_PARAM, *PDNS_RPC_NAME_AND_PARAM;

typedef struct _DnsRpcIPValidateLonghorn
{
    DWORD        dwRpcStructureVersion;
    DWORD        dwReserved0;

    DWORD        dwContext;
    DWORD        dwReserved1;
    [string] char *    pszContextName;
    PDNS_ADDR_ARRAY    aipValidateAddrs;
}
DNS_RPC_IP_VALIDATE, *PDNS_RPC_IP_VALIDATE;

//
// String array
//

typedef struct _DnsRpcUtf8StringList
{
    [range(0,10000)]    DWORD        dwCount;
    [size_is(dwCount),string]    char *    pszStrings[];
}
DNS_RPC_UTF8_STRING_LIST, *PDNS_RPC_UTF8_STRING_LIST;

typedef struct _DnsRpcUnicodeStringList
{
    [range(0,10000)]    DWORD        dwCount;
    [size_is(dwCount),string]    wchar_t *    pwszStrings[];
}
DNS_RPC_UNICODE_STRING_LIST, *PDNS_RPC_UNICODE_STRING_LIST;

//
// Union of RPC types
//

typedef enum _DnsSrvRpcTypeId
{
    DNSSRV_TYPEID_NULL = 0,
    DNSSRV_TYPEID_DWORD,

```

```

DNSSRV_TYPEID_LPSTR,
DNSSRV_TYPEID_LPWSTR,
DNSSRV_TYPEID_IPARRAY,
DNSSRV_TYPEID_BUFFER, // 5
DNSSRV_TYPEID_SERVER_INFO_W2K,
DNSSRV_TYPEID_STATS,
DNSSRV_TYPEID_FORWARDERS_W2K,
DNSSRV_TYPEID_ZONE_W2K,
DNSSRV_TYPEID_ZONE_INFO_W2K, // 10
DNSSRV_TYPEID_ZONE_SECONDARIES_W2K,
DNSSRV_TYPEID_ZONE_DATABASE_W2K,
DNSSRV_TYPEID_ZONE_TYPE_RESET_W2K,
DNSSRV_TYPEID_ZONE_CREATE_W2K,
DNSSRV_TYPEID_NAME_AND_PARAM, // 15
DNSSRV_TYPEID_ZONE_LIST_W2K,
DNSSRV_TYPEID_ZONE_RENAME,
DNSSRV_TYPEID_ZONE_EXPORT,
DNSSRV_TYPEID_SERVER_INFO_DOTNET,
DNSSRV_TYPEID_FORWARDERS_DOTNET, // 20
DNSSRV_TYPEID_ZONE,
DNSSRV_TYPEID_ZONE_INFO_DOTNET,
DNSSRV_TYPEID_ZONE_SECONDARIES_DOTNET,
DNSSRV_TYPEID_ZONE_DATABASE,
DNSSRV_TYPEID_ZONE_TYPE_RESET_DOTNET, // 25
DNSSRV_TYPEID_ZONE_CREATE_DOTNET,
DNSSRV_TYPEID_ZONE_LIST,
DNSSRV_TYPEID_DP_ENUM,
DNSSRV_TYPEID_DP_INFO,
DNSSRV_TYPEID_DP_LIST, // 30
DNSSRV_TYPEID_ENLIST_DP,
DNSSRV_TYPEID_ZONE_CHANGE_DP,
DNSSRV_TYPEID_ENUM_ZONES_FILTER,
DNSSRV_TYPEID_ADDRARRAY,
DNSSRV_TYPEID_SERVER_INFO, // 35
DNSSRV_TYPEID_ZONE_INFO,
DNSSRV_TYPEID_FORWARDERS,
DNSSRV_TYPEID_ZONE_SECONDARIES,
DNSSRV_TYPEID_ZONE_TYPE_RESET,
DNSSRV_TYPEID_ZONE_CREATE, // 40
DNSSRV_TYPEID_IP_VALIDATE,
DNSSRV_TYPEID_AUTOCONFIGURE,
DNSSRV_TYPEID_UTF8_STRING_LIST,
DNSSRV_TYPEID_UNICODE_STRING_LIST,
DNSSRV_TYPEID_SKD, // 45
DNSSRV_TYPEID_SKD_LIST,
DNSSRV_TYPEID_SKD_STATE,
DNSSRV_TYPEID_SIGNING_VALIDATION_ERROR,
DNSSRV_TYPEID_TRUST_POINT_LIST,
DNSSRV_TYPEID_TRUST_ANCHOR_LIST, // 50
DNSSRV_TYPEID_ZONE_SIGNING_SETTINGS,
DNSSRV_TYPEID_ZONE_SCOPE_ENUM,
DNSSRV_TYPEID_ZONE_STATS,
DNSSRV_TYPEID_ZONE_SCOPE_CREATE,
DNSSRV_TYPEID_ZONE_SCOPE_INFO // 55
}
DNS_RPC_TYPEID, *PDNS_RPC_TYPEID;

#define DNS_RPC_ZONE_STATS_VER 1
#define DNS_RPC_ZONE_STATS DNS_RPC_ZONE_STATS_V1

```

```

#define PDNS_RPC_ZONE_STATS                PDNS_RPC_ZONE_STATS_V1

#define DNS_RPC_ZONE_SCOPE_CREATE_INFO_VER 1
#define DNS_RPC_ZONE_SCOPE_CREATE_INFO    DNS_RPC_ZONE_SCOPE_CREATE_INFO_V1
#define PDNS_RPC_ZONE_SCOPE_CREATE_INFO    PDNS_RPC_ZONE_SCOPE_CREATE_INFO_V1

#define DNS_RPC_ZONE_SCOPE_INFO_VER        1
#define DNS_RPC_ZONE_SCOPE_INFO            DNS_RPC_ZONE_SCOPE_INFO_V1
#define PDNS_RPC_ZONE_SCOPE_INFO            PDNS_RPC_ZONE_SCOPE_INFO_V1

#define DNS_RPC_ENUM_ZONE_SCOPE_LIST_VER   1

typedef struct _DnsRpcEnumZoneScopeList
{
    DWORD        dwRpcStructureVersion;

    DWORD        dwZoneScopeCount;
    // array of dwZoneScopeCount zones
    LPWSTR       ZoneScopeArray[1];
} DNS_RPC_ENUM_ZONE_SCOPE_LIST, * PDNS_RPC_ENUM_ZONE_SCOPE_LIST;

//
// DNS server statistics
//

typedef struct _DnsSystemTime
{
    WORD         wYear;
    WORD         wMonth;
    WORD         wDayOfWeek;
    WORD         wDay;
    WORD         wHour;
    WORD         wMinute;
    WORD         wSecond;
    WORD         wMilliseconds;
}
DNS_SYSTEMTIME;

//
// Zone time statistics info.
//
typedef struct _DnsZoneTimeStats
{
    DNS_SYSTEMTIME StatsCollectionStartTime;
}
DNSSRV_ZONE_TIME_STATS, *PDNSSRV_ZONE_TIME_STATS;

#define MAX_RECORD_TYPES                31
#define MAX_ZONE_TRANSFER_TYPES         2

//
// Enum for all zone statistics types
//
typedef enum _DNS_ZONE_STATS_TYPE {
    //
    // Record types. If new zone stats record type is added/deleted
    // then update MAX_RECORD_TYPES accordingly.
    //
}

```



```

ZONE_STATS_TYPE_RECORD_A = 0,
ZONE_STATS_TYPE_RECORD_AAAA,
ZONE_STATS_TYPE_RECORD_PTR,
ZONE_STATS_TYPE_RECORD_CNAME,
ZONE_STATS_TYPE_RECORD_MX,
ZONE_STATS_TYPE_RECORD_AFSDB,
ZONE_STATS_TYPE_RECORD_ATMA,
ZONE_STATS_TYPE_RECORD_DHCID,
ZONE_STATS_TYPE_RECORD_DNAME,
ZONE_STATS_TYPE_RECORD_HINFO,
ZONE_STATS_TYPE_RECORD_ISDN,
ZONE_STATS_TYPE_RECORD_MG,
ZONE_STATS_TYPE_RECORD_MB,
ZONE_STATS_TYPE_RECORD_MINFO,
ZONE_STATS_TYPE_RECORD_NAPTR,
ZONE_STATS_TYPE_RECORD_NXT,
ZONE_STATS_TYPE_RECORD_KEY,
ZONE_STATS_TYPE_RECORD_MR,
ZONE_STATS_TYPE_RECORD_RP,
ZONE_STATS_TYPE_RECORD_RT,
ZONE_STATS_TYPE_RECORD_SRV,
ZONE_STATS_TYPE_RECORD_SIG,
ZONE_STATS_TYPE_RECORD_TEXT,
ZONE_STATS_TYPE_RECORD_WKS,
ZONE_STATS_TYPE_RECORD_X25,
ZONE_STATS_TYPE_RECORD_DNSKEY,
ZONE_STATS_TYPE_RECORD_DS,
ZONE_STATS_TYPE_RECORD_NS,
ZONE_STATS_TYPE_RECORD_SOA,
ZONE_STATS_TYPE_RECORD_ALL,
ZONE_STATS_TYPE_RECORD_OTHERS,

//
// Zone transfer types. If new zone transfer type is added/deleted
// then update MAX_ZONE_TRANSFER_TYPES accordingly.
//
ZONE_STATS_TYPE_TRANSFER_AXFR,
ZONE_STATS_TYPE_TRANSFER_IXFR,

//
// Zone update.
//
ZONE_STATS_TYPE_UPDATE,

MAX_ZONE_STATS_TYPES
} DNS_ZONE_STATS_TYPE, *PDNS_DONE_STATS_TYPE;

//
// Per zone per record type statistics.
//
typedef struct _DnsZoneQueryStats {
    DNS_ZONE_STATS_TYPE RecordType;
    ULONG64 QueriesResponded;
    ULONG64 QueriesReceived;
    ULONG64 QueriesFailure;
    ULONG64 QueriesNameError;
}
DNSSRV_ZONE_QUERY_STATS, *PDNSSRV_ZONE_QUERY_STATS;

```

```

//
// Zone transfer statistics.
//
typedef struct _DnsZoneTransferStats {
    DNS_ZONE_STATS_TYPE TransferType;
    ULONG64 RequestReceived;
    ULONG64 RequestSent;
    ULONG64 ResponseReceived;
    ULONG64 SuccessReceived;
    ULONG64 SuccessSent;
}
DNSSRV_ZONE_TRANSFER_STATS, *PDNSSRV_ZONE_TRANSFER_STATS;

//
// Zone update statistics.
//
typedef struct _DnsZoneUpdateStats {
    DNS_ZONE_STATS_TYPE Type;
    ULONG64 DynamicUpdateReceived;
    ULONG64 DynamicUpdateRejected;
}
DNSSRV_ZONE_UPDATE_STATS, *PDNSSRV_ZONE_UPDATE_STATS;

typedef struct _DnsRpcZoneStats {
    DWORD dwRpcStructureVersion;
    DNSSRV_ZONE_TIME_STATS ZoneTimeStats;
    DNSSRV_ZONE_QUERY_STATS ZoneQueryStats[MAX_RECORD_TYPES];
    DNSSRV_ZONE_TRANSFER_STATS ZoneTransferStats[MAX_ZONE_TRANSFER_TYPES];
    DNSSRV_ZONE_UPDATE_STATS ZoneUpdateStats;
}DNS_RPC_ZONE_STATS_V1, *PDNS_RPC_ZONE_STATS_V1;

typedef struct _DnsRpcScopeCreate
{
    DWORD dwFlags ;
    LPWSTR pwszScopeName; // Scope name that is to be created
}DNS_RPC_ZONE_SCOPE_CREATE_INFO_V1,*PDNS_RPC_ZONE_SCOPE_CREATE_INFO_V1;

typedef struct _DnsRpcZoneScopeInfo
{
    DWORD dwRpcStructureVersion;
    LPWSTR pwszScopeName;
    LPWSTR pwszDataFile;
} DNS_RPC_ZONE_SCOPE_INFO_V1, *PDNS_RPC_ZONE_SCOPE_INFO_V1;

typedef [switch_type(DWORD)] union _DnssrvSrvRpcUnion
{
    [case(DNSSRV_TYPEID_NULL)] PBYTE Null;

    [case(DNSSRV_TYPEID_DWORD)] DWORD Dword;

    [case(DNSSRV_TYPEID_LPSTR)] [string] char * String;

    [case(DNSSRV_TYPEID_LPWSTR)] [string] wchar_t * WideString;

    [case(DNSSRV_TYPEID_IPARRAY)] PIP4_ARRAY IpArray;

    [case(DNSSRV_TYPEID_BUFFER)] PDNS_RPC_BUFFER Buffer;
}

```

```

[case (DNSSRV_TYPEID_SERVER_INFO_W2K) ]
    PDNS_RPC_SERVER_INFO_W2K                ServerInfoW2K;

[case (DNSSRV_TYPEID_STATS) ]
    PDNSRPC_STATS                            Stats;

[case (DNSSRV_TYPEID_FORWARDERS_W2K) ]
    PDNS_RPC_FORWARDERS_W2K                 ForwardersW2K;

[case (DNSSRV_TYPEID_ZONE_W2K) ]
    PDNS_RPC_ZONE_W2K                       ZoneW2K;

[case (DNSSRV_TYPEID_ZONE_INFO_W2K) ]
    PDNS_RPC_ZONE_INFO_W2K                  ZoneInfoW2K;

[case (DNSSRV_TYPEID_ZONE_SECONDARIES_W2K) ]
    PDNS_RPC_ZONE_SECONDARIES_W2K          SecondariesW2K;

[case (DNSSRV_TYPEID_ZONE_DATABASE_W2K) ]
    PDNS_RPC_ZONE_DATABASE_W2K             DatabaseW2K;

[case (DNSSRV_TYPEID_ZONE_CREATE_W2K) ]
    PDNS_RPC_ZONE_CREATE_INFO_W2K          ZoneCreateW2K;

[case (DNSSRV_TYPEID_NAME_AND_PARAM) ]
    PDNS_RPC_NAME_AND_PARAM                 NameAndParam;

[case (DNSSRV_TYPEID_ZONE_LIST_W2K) ]
    PDNS_RPC_ZONE_LIST_W2K                 ZoneListW2K;

[case (DNSSRV_TYPEID_SERVER_INFO_DOTNET) ]
    PDNS_RPC_SERVER_INFO_DOTNET            ServerInfoDotNet;

[case (DNSSRV_TYPEID_FORWARDERS_DOTNET) ]
    PDNS_RPC_FORWARDERS_DOTNET             ForwardersDotNet;

[case (DNSSRV_TYPEID_ZONE) ]
    PDNS_RPC_ZONE                           Zone;

[case (DNSSRV_TYPEID_ZONE_INFO_DOTNET) ]
    PDNS_RPC_ZONE_INFO_DOTNET              ZoneInfoDotNet;
[case (DNSSRV_TYPEID_ZONE_SECONDARIES_DOTNET) ]
    PDNS_RPC_ZONE_SECONDARIES_DOTNET       SecondariesDotNet;

[case (DNSSRV_TYPEID_ZONE_DATABASE) ]
    PDNS_RPC_ZONE_DATABASE                  Database;

[case (DNSSRV_TYPEID_ZONE_CREATE_DOTNET) ]
    PDNS_RPC_ZONE_CREATE_INFO_DOTNET       ZoneCreateDotNet;

[case (DNSSRV_TYPEID_ZONE_LIST) ]
    PDNS_RPC_ZONE_LIST                      ZoneList;

[case (DNSSRV_TYPEID_ZONE_EXPORT) ]
    PDNS_RPC_ZONE_EXPORT_INFO              ZoneExport;

[case (DNSSRV_TYPEID_DP_INFO) ]
    PDNS_RPC_DP_INFO                        DirectoryPartition;

```

```

[case (DNSSRV_TYPEID_DP_ENUM) ]
    PDNS_RPC_DP_ENUM                                DirectoryPartitionEnum;

[case (DNSSRV_TYPEID_DP_LIST) ]
    PDNS_RPC_DP_LIST                                DirectoryPartitionList;

[case (DNSSRV_TYPEID_ENLIST_DP) ]
    PDNS_RPC_ENLIST_DP                              EnlistDirectoryPartition;

[case (DNSSRV_TYPEID_ZONE_CHANGE_DP) ]
    PDNS_RPC_ZONE_CHANGE_DP                        ZoneChangeDirectoryPartition;

[case (DNSSRV_TYPEID_ENUM_ZONES_FILTER) ]
    PDNS_RPC_ENUM_ZONES_FILTER                    EnumZonesFilter;

[case (DNSSRV_TYPEID_ADDRARRAY) ]
    PDNS_ADDR_ARRAY                                AddrArray;

[case (DNSSRV_TYPEID_SERVER_INFO) ]
    PDNS_RPC_SERVER_INFO                          ServerInfo;

[case (DNSSRV_TYPEID_ZONE_CREATE) ]
    PDNS_RPC_ZONE_CREATE_INFO                    ZoneCreate;

[case (DNSSRV_TYPEID_FORWARDERS) ]
    PDNS_RPC_FORWARDERS                          Forwarders;

[case (DNSSRV_TYPEID_ZONE_SECONDARIES) ]
    PDNS_RPC_ZONE_SECONDARIES                    Secondaries;

[case (DNSSRV_TYPEID_IP_VALIDATE) ]
    PDNS_RPC_IP_VALIDATE                          IpValidate;

[case (DNSSRV_TYPEID_ZONE_INFO) ]
    PDNS_RPC_ZONE_INFO                            ZoneInfo;

[case (DNSSRV_TYPEID_AUTOCONFIGURE) ]
    PDNS_RPC_AUTOCONFIGURE                        AutoConfigure;

[case (DNSSRV_TYPEID_UTF8_STRING_LIST) ]
    PDNS_RPC_UTF8_STRING_LIST                    Utf8StringList;

[case (DNSSRV_TYPEID_UNICODE_STRING_LIST) ]
    PDNS_RPC_UNICODE_STRING_LIST                UnicodeStringList;

[case (DNSSRV_TYPEID_SKD) ]
    PDNS_RPC_SKD                                  Skd;

[case (DNSSRV_TYPEID_SKD_LIST) ]
    PDNS_RPC_SKD_LIST                            SkdList;

[case (DNSSRV_TYPEID_SKD_STATE) ]
    PDNS_RPC_SKD_STATE                          SkdState;

[case (DNSSRV_TYPEID_SIGNING_VALIDATION_ERROR) ]
    PDNS_RPC_SIGNING_VALIDATION_ERROR            SigningValidationError;

[case (DNSSRV_TYPEID_TRUST_POINT_LIST) ]
    PDNS_RPC_TRUST_POINT_LIST                    TrustPointList;

```

```

    [case (DNSSRV_TYPEID_TRUST_ANCHOR_LIST)]
        PDNS_RPC_TRUST_ANCHOR_LIST                TrustAnchorList;

    [case (DNSSRV_TYPEID_ZONE_SIGNING_SETTINGS)]
        PDNS_RPC_ZONE_DNSSEC_SETTINGS              ZoneDnsSecSettings;

    [case (DNSSRV_TYPEID_ZONE_SCOPE_ENUM)]
        PDNS_RPC_ENUM_ZONE_SCOPE_LIST             ZoneScopeList;

    [case (DNSSRV_TYPEID_ZONE_STATS)]
        PDNS_RPC_ZONE_STATS                       ZoneStats;

    [case (DNSSRV_TYPEID_ZONE_SCOPE_CREATE)]
        PDNS_RPC_ZONE_SCOPE_CREATE_INFO          ScopeCreate;

    [case (DNSSRV_TYPEID_ZONE_SCOPE_INFO)]
        PDNS_RPC_ZONE_SCOPE_INFO                 ScopeInfo;
}
DNSSRV_RPC_UNION;

typedef enum _ImportOperationResult
{
    // No change was detected in Signing Settings imported and that of the Zone
    IMPORT_STATUS_NOOP,

    // Zone is marked for Re-Sign
    IMPORT_STATUS_SIGNING_READY,

    // Zone is marked for Unsign
    IMPORT_STATUS_UNSIGNING_READY,

    // Change was detected but no re-signing\unsigning required
    IMPORT_STATUS_CHANGED

} ImportOpResult;

[
    uuid(50abc2a4-574d-40b3-9d66-ee4fd5fba076),
    version(5.0),
    pointer_default(unique)
]
interface DnsServer
{
    LONG
    R_DnssrvOperation(
        [in]                handle_t                hBindingHandle,
        [in, unique, string] LPCWSTR                pwszServerName,
        [in, unique, string] LPCSTR                 pszZone,
        [in]                 DWORD                  dwContext,
        [in, unique, string] LPCSTR                 pszOperation,
        [in]                 DWORD                  dwTypeId,
        [in, switch_is(dwTypeId)] DNSSRV_RPC_UNION pData
    );

    LONG
    R_DnssrvQuery(

```

```

[in] handle_t hBindingHandle,
[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in, unique, string] LPCSTR pszOperation,
[out] PDWORD pdwTypeId,
[out, switch_is(*pdwTypeId)] DNSSRV_RPC_UNION * ppData
);

LONG
R_DnssrvComplexOperation(
[in] handle_t hBindingHandle,
[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in, unique, string] LPCSTR pszOperation,
[in] DWORD dwTypeIn,
[in, switch_is(dwTypeIn)] DNSSRV_RPC_UNION pDataIn,
[out] PDWORD pdwTypeOut,
[out, switch_is(*pdwTypeOut)] DNSSRV_RPC_UNION * ppDataOut
);

LONG
R_DnssrvEnumRecords(
[in] handle_t hBindingHandle,
[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in, unique, string] LPCSTR pszNodeName,
[in, unique, string] LPCSTR pszStartChild,
[in] WORD wRecordType,
[in] DWORD fSelectFlag,
[in, unique, string] LPCSTR pszFilterStart,
[in, unique, string] LPCSTR pszFilterStop,
[out] PDWORD pdwBufferLength,
[out, size_is(, *pdwBufferLength)] PBYTE * ppBuffer
);

LONG
R_DnssrvUpdateRecord(
[in] handle_t hBindingHandle,
[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in, string] LPCSTR pszNodeName,
[in, unique] PDNS_RPC_RECORD pAddRecord,
[in, unique] PDNS_RPC_RECORD pDeleteRecord
);

LONG
R_DnssrvOperation2(
[in] handle_t hBindingHandle,
[in] DWORD dwClientVersion,
[in] DWORD dwSettingFlags,
[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in] DWORD dwContext,
[in, unique, string] LPCSTR pszOperation,
[in] DWORD dwTypeId,
[in, switch_is(dwTypeId)] DNSSRV_RPC_UNION pData
);

LONG

```

```

R_DnssrvQuery2 (
    [in] handle_t hBindingHandle,
    [in] DWORD dwClientVersion,
    [in] DWORD dwSettingFlags,
    [in, unique, string] LPCWSTR pwszServerName,
    [in, unique, string] LPCSTR pszZone,
    [in, unique, string] LPCSTR pszOperation,
    [out] PDWORD pdwTypeId,
    [out, switch_is(*pdwTypeId)] DNSSRV_RPC_UNION * ppData
);

LONG
R_DnssrvComplexOperation2 (
    [in] handle_t hBindingHandle,
    [in] DWORD dwClientVersion,
    [in] DWORD dwSettingFlags,
    [in, unique, string] LPCWSTR pwszServerName,
    [in, unique, string] LPCSTR pszZone,
    [in, unique, string] LPCSTR pszOperation,
    [in] DWORD dwTypeIn,
    [in, switch_is(dwTypeIn)] DNSSRV_RPC_UNION pDataIn,
    [out] PDWORD pdwTypeOut,
    [out, switch_is(*pdwTypeOut)] DNSSRV_RPC_UNION * ppDataOut
);

LONG
R_DnssrvEnumRecords2 (
    [in] handle_t hBindingHandle,
    [in] DWORD dwClientVersion,
    [in] DWORD dwSettingFlags,
    [in, unique, string] LPCWSTR pwszServerName,
    [in, unique, string] LPCSTR pszZone,
    [in, unique, string] LPCSTR pszNodeName,
    [in, unique, string] LPCSTR pszStartChild,
    [in] WORD wRecordType,
    [in] DWORD fSelectFlag,
    [in, unique, string] LPCSTR pszFilterStart,
    [in, unique, string] LPCSTR pszFilterStop,
    [out] PDWORD pdwBufferLength,
    [out, size_is(, *pdwBufferLength)] PBYTE * ppBuffer
);

LONG
R_DnssrvUpdateRecord2 (
    [in] handle_t hBindingHandle,
    [in] DWORD dwClientVersion,
    [in] DWORD dwSettingFlags,
    [in, unique, string] LPCWSTR pwszServerName,
    [in, unique, string] LPCSTR pszZone,
    [in, string] LPCSTR pszNodeName,
    [in, unique] PDNS_RPC_RECORD pAddRecord,
    [in, unique] PDNS_RPC_RECORD pDeleteRecord
);

LONG
R_DnssrvUpdateRecord3 (
    [in] handle_t hBindingHandle,
    [in] DWORD dwClientVersion,
    [in] DWORD dwSettingFlags,

```

```

[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in, unique, string] LPCWSTR pwszZoneScope,
[in, string] LPCSTR pszNodeName,
[in, unique] PDNS_RPC_RECORD pAddRecord,
[in, unique] PDNS_RPC_RECORD pDeleteRecord
);

LONG
R_DnssrvEnumRecords3(
[in] handle_t hBindingHandle,
[in] DWORD dwClientVersion,
[in] DWORD dwSettingFlags,
[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in, unique, string] LPCWSTR pwszZoneScope,
[in, unique, string] LPCSTR pszNodeName,
[in, unique, string] LPCSTR pszStartChild,
[in] WORD wRecordType,
[in] DWORD fSelectFlag,
[in, unique, string] LPCSTR pszFilterStart,
[in, unique, string] LPCSTR pszFilterStop,
[out] PDWORD pdwBufferLength,
[out, size_is(, *pdwBufferLength)] PBYTE * ppBuffer
);

LONG
R_DnssrvOperation3(
[in] handle_t hBindingHandle,
[in] DWORD dwClientVersion,
[in] DWORD dwSettingFlags,
[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in, unique, string] LPCWSTR pwszZoneScope,
[in] DWORD dwContext,
[in, unique, string] LPCSTR pszOperation,
[in] DWORD dwTypeId,
[in, switch_is(dwTypeId)] DNSSRV_RPC_UNION pData
);

LONG
R_DnssrvQuery3(
[in] DWORD dwClientVersion,
[in] DWORD dwSettingFlags,
[in, unique, string] LPCWSTR pwszServerName,
[in, unique, string] LPCSTR pszZone,
[in, unique, string] LPCWSTR pszZoneScopeName,
[in, unique, string] LPCSTR pszOperation,
[out] PDWORD pdwTypeId,
[out, switch_is(*pdwTypeId)] DNSSRV_RPC_UNION * ppData
);
}

```



## 7 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:

- Windows NT operating system
- Windows 2000 operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 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 1.1:](#) Windows Server 2012 and Windows Server 2012 R2 support automatic DNSSEC signing of zones.

[<2> Section 2.1:](#) The DNS server management protocol client on Windows Server 2008 and Windows Server 2008 R2 does not support using RPC over Named-Pipes.

[<3> Section 2.1.2:](#) Windows 2000 and Windows Server 2003 clients always request `RPC_C_QOS_CAPABILITIES_MUTUAL_AUTH`. Windows Server 2008 and Windows Server 2008 R2 clients additionally request `RPC_C_QOS_CAPABILITIES_IGNORE_DELEGATE_FAILURE` during [R DnssrvOperation \(section 3.1.4.1\)](#) or [R DnssrvOperation2 \(section 3.1.4.6\)](#) when `pszOperation` is "EnlistDirectoryPartition".

[<4> Section 2.1.2:](#) Windows 2000 and Windows Server 2003 clients always request `RPC_C_IMP_LEVEL_DELEGATE`. Windows Server 2008 and Windows Server 2008 R2 clients request `RPC_C_IMP_LEVEL_DELEGATE` during [R DnssrvOperation](#) or [R DnssrvOperation2](#) when `pszOperation` is "EnlistDirectoryPartition".

[<5> Section 2.2.1.1.1:](#) zone scopes are available only in Windows Server 2012 R2.

<6> [Section 2.2.1.1.1](#): Windows Server 2008 and Windows Server 2008 R2 support type IDs up to and including `DNSSRV_TYPEID_UTF8_STRING_LIST`. The Windows Server 2003 DNS server supports type IDs up to and including `DNSSRV_TYPEID_ZONE_LIST`, as enumerated in section [2.2.1.1.1](#). The Windows 2000 DNS server supports type IDs up to and including `DNSSRV_TYPEID_ZONE_LIST_W2K`.

<7> [Section 2.2.1.1.2](#): Windows clients and servers use this value to indicate use of LPC [[MSDN-RPC](#)].

<8> [Section 2.2.1.2.6](#): Windows Server 2003 supports elements of this union up to and including `ZoneCreateDotNet`. Windows 2000 supports elements of this union up to and including `ZoneListW2K`.

<9> [Section 2.2.2.1.1](#): Windows 2000 does not support the following types: `DNS_TYPE_SIG`, `DNS_TYPE_KEY`, `DNS_TYPE_NXT`, `DNS_TYPE_NAPTR`, and `DNS_TYPE_DNAME`. Windows Server 2003 does not support `DNS_TYPE_NAPTR` and `DNS_TYPE_DNAME`. The types `DNS_TYPE_DS`, `DNS_TYPE_RRSIG`, `DNS_TYPE_NSEC`, `DNS_TYPE_DNSKEY`, and `DNS_TYPE_DHCID` are supported only on Windows Server 2008 R2. `DNS_TYPE_NSEC3` and `DNS_TYPE_NSEC3PARAM` are supported only on Windows Server 2012 and Windows Server 2012 R2.

<10> [Section 2.2.2.1.2](#): Windows 2000 does not support the `DNS_RPC_FLAG_OPEN_ACL` record flag.

<11> [Section 2.2.2.4.2](#): Windows 2000 and Windows Server 2003 do not support `DNS_TYPE_DNAME`.

<12> [Section 2.2.2.4.9](#): This record type is not supported in Windows 2000.

<13> [Section 2.2.2.4.10](#): This record type is only supported by Windows 7 and Windows Server 2008 R2.

<14> [Section 2.2.2.4.11](#): This record type is only supported by Windows 7 and Windows Server 2008 R2.

<15> [Section 2.2.2.4.12](#): This record type is only supported by Windows 7 and Windows Server 2008 R2.

<16> [Section 2.2.2.4.13](#): This record type is not supported in Windows 2000.

<17> [Section 2.2.2.4.14](#): This record type is only supported by Windows 7 and Windows Server 2008 R2.

<18> [Section 2.2.2.4.15](#): This record type is only supported by Windows 7 and Windows Server 2008 R2.

<19> [Section 2.2.2.4.17](#): This record type is not supported in Windows 2000.

<20> [Section 2.2.2.4.20](#): This record type is not supported in Windows 2000.

<21> [Section 2.2.2.4.24](#): This record type is supported only by Windows Server 2012 and Windows Server 2012 R2.

<22> [Section 2.2.2.4.25](#): This record type is supported only by Windows Server 2012 and Windows Server 2012 R2.

<23> [Section 2.2.2.5](#): The records `DNS_TYPE_DS`, `DNS_TYPE_RRSIG`, `DNS_TYPE_NSEC`, `DNS_TYPE_DNSKEY` and `DNS_TYPE_DHCID` are only supported in Windows 7 and Windows Server 2008 R2.

<24> [Section 2.2.3.1.1](#): Windows 2000 and Windows Server 2003 do not support IP validation.

<25> [Section 2.2.3.1.2](#): Windows 2000 and Windows Server 2003 do not support IP validation.

<26> [Section 2.2.4.1.1](#): Windows NT 4.0 populates its database in the following order, until successful: from a file-based persistent storage or from the persistent copy of the DNS Zone Table.

<27> [Section 2.2.4.2.1](#): Windows uses the build number as the OS Revision.

<28> [Section 2.2.4.2.2.1](#): All versions of Windows Server listed in the supported products list in [Appendix B: Product Behavior](#) except Windows NT Server 4.0 (which predates Active Directory) use "cn=MicrosoftDNS,cn=System" as the constant container relative distinguished name. A complete DS Container string could, for example, be "cn=MicrosoftDNS,cn=System,DC=corp,DC=contoso,DC=com".

<29> [Section 2.2.4.2.2.2](#): This version of the structure is for use with Windows Server 2003.

<30> [Section 2.2.4.2.2.3](#): This version of the structure is for use with Windows Server 2008 and Windows Server 2008 R2.

<31> [Section 2.2.5.1.1](#): Windows 2000 does not support the forwarder or stub zone types. Windows 2000 and Windows Server 2003 do not support the secondary cache zone type.

<32> [Section 2.2.5.1.4](#): Windows 2000 does not support any zone request filter values that involve application directory partitions. Windows 2000 does not support stub or forwarder zone request filters.

<33> [Section 2.2.5.1.5](#): Only Windows Server 2012 and Windows Server 2012 R2 support DNSSEC key rollover. Windows Server 2012 and Windows Server 2012 R2 support only DNS\_ROLLOVER\_TYPE\_PREPUBLISH for signing key descriptors where fIsKSK is FALSE and DNS\_ROLLOVER\_TYPE\_DOUBLE\_SIGNATURE for signing key descriptors where fIsKSK is TRUE.

<34> [Section 2.2.5.1.6](#): Only Windows Server 2012 and Windows Server 2012 R2 support DNSSEC key rollover.

<35> [Section 2.2.5.2.1](#): Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2 support DNS\_RPC\_ZONE\_DOTNET and DNS\_RPC\_ZONE\_W2K, and which one is used depends on **dwClientVersion**. Windows 2000 only supports DNS\_RPC\_ZONE\_W2K.

<36> [Section 2.2.5.2.2](#): The Windows DNS server auto-creates the 0.in-addr.arpa, 127.in-addr.arpa, and 255.in-addr.arpa zones as a performance optimization to avoid unnecessary recursions to the root server for queries for standard IP addresses such as 0.0.0.0, 127.0.0.1 (loopback), and 255.255.255.255 (**broadcast**).

<37> [Section 2.2.5.2.2](#): Windows 2000 and Windows Server 2003 do not support the ReadOnly bit.

<38> [Section 2.2.5.2.3](#): Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2 support DNS\_RPC\_ZONE\_LIST\_DOTNET and DNS\_RPC\_ZONE\_LIST\_W2K, and which one is used depends on **dwClientVersion**. Windows 2000 only supports DNS\_RPC\_ZONE\_LIST\_W2K.

<39> [Section 2.2.5.2.4.2](#): All versions of Windows Server listed in the supported products list in [Appendix B: Product Behavior](#) incorrectly set this to 0x00000000.

<40> [Section 2.2.5.2.4.3](#): All versions of Windows Server listed in the supported products list in [Appendix B: Product Behavior](#) incorrectly set this to 0x00000000.

<41> [Section 2.2.5.2.8](#): This structure is not implemented in Windows 2000 Server.

- <42> [Section 2.2.5.2.10.1](#): Windows 2000 Server uses 5 minutes (300 seconds).
- <43> [Section 2.2.6.1.2](#): KeySignScope is available in Windows Server 2012 R2 only.
- <44> [Section 2.2.6.1.3](#): ImportOpResult is available Windows Server 2012 R2 only.
- <45> [Section 2.2.6.2.1](#): Windows Server 2012 and Windows Server 2012 R2 support RSASHA1, RSASHA1-NSEC3-SHA1, RSASHA256, and RSASHA512.
- <46> [Section 2.2.6.2.9](#): The **DNS\_RPC\_ZONE\_DNSSEC\_SETTINGS** structure is available in Windows Server 2012 R2 only.
- <47> [Section 2.2.6.2.10](#): The **DNS\_RPC\_ZONE\_SKD** structure is available in Windows Server 2012 R2 only.
- <48> [Section 2.2.6.2.11](#): The **DNS\_RPC\_SKD\_STATE\_EX** structure is available in Windows Server 2012 R2 only.
- <49> [Section 2.2.7.1.1](#): This enumeration is supported only by Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2.
- <50> [Section 2.2.7.1.1](#): Windows 2000 Server and Windows Server 2003 do not support read-only DCs and do not process the **msDS-NC-RO-Replica-Locations**.
- <51> [Section 2.2.7.2.1](#): This structure is only supported by Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2.
- <52> [Section 2.2.7.2.1](#): Windows 2000 Server and Windows Server 2003 do not support read-only DCs and do not process the **msDS-NC-RO-Replica-Locations**.
- <53> [Section 2.2.7.2.1](#): Windows 2000 Server and Windows Server 2003 do not support read-only DCs and do not process the **msDS-NC-RO-Replica-Locations**.
- <54> [Section 2.2.7.2.2](#): This structure and its associated operations are only supported by Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2.
- <55> [Section 2.2.7.2.3](#): This structure is only supported by Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2.
- <56> [Section 2.2.7.2.4](#): This structure and its associated operations are only supported by Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2.
- <57> [Section 2.2.7.2.5](#): This structure and its associated operations are only supported by Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2.
- <58> [Section 2.2.7.2.6](#): This structure and its associated operations are only supported by Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2.
- <59> [Section 2.2.8.1.1](#): Windows NT 4.0 and Windows 2000 do not support this structure. Windows Server 2003 does not support the following values:  
DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT\_PREPEND,  
DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT\_APPEND, and  
DNS\_RPC\_AUTOCONFIG\_INTERNAL\_RETURN\_ERRORS.
- <60> [Section 2.2.8.1.1](#): Windows Server 2003 does not support the  
DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT\_APPEND constant. Use  
DNS\_RPC\_AUTOCONFIG\_INTERNAL\_SELFPOINT instead.

<61> [Section 2.2.8.1.1](#): Windows Server 2003 does not support the `DNS_RPC_AUTOCONFIG_INTERNAL_SELFPOINT_PREPEND` constant. Use `DNS_RPC_AUTOCONFIG_INTERNAL_SELFPOINT` instead.

<62> [Section 2.2.8.1.1](#): The Windows Server 2003 server and client use the values in the table below to obtain a constant. The Windows Server 2003 server interprets each value in the right-hand column as the corresponding constant in the left-hand column, regardless of the version of the client connecting to it. The Windows Server 2003 client uses the corresponding values in the table to indicate each constant, regardless of the version of the server it is connecting to. No Windows implementation checks the version of the other communicating host when determining how to select or interpret these values.

Constant	Value used by Windows Server 2003
<code>DNS_RPC_AUTOCONFIG_INTERNAL_ROOTHINTS</code>	0x00000001
<code>DNS_RPC_AUTOCONFIG_INTERNAL_FORWARDERS</code>	0x00000002
<code>DNS_RPC_AUTOCONFIG_ZONES</code>	0x00000008
<code>DNS_RPC_AUTOCONFIG_INTERNAL_SELFPOINT</code>	0x00000004
<code>DNS_RPC_AUTOCONFIG_ALL</code>	0xFFFFFFFF

<63> [Section 2.2.9.1.1](#): This value is only supported by Windows 7 and Windows Server 2008 R2.

<64> [Section 2.2.10.1.1](#): This value is only supported by Windows 7 and Windows Server 2008 R2.

<65> [Section 2.2.10.2.6](#): Windows 2000 DNS servers do not include this field.

<66> [Section 2.2.10.2.7](#): Windows 2000 DNS servers do not include this field.

<67> [Section 2.2.10.2.7](#): Windows 2000 and Windows Server 2003 DNS servers do not include this field.

<68> [Section 2.2.10.2.7](#): This field is only supported by Windows 7 and Windows Server 2008 R2.

<69> [Section 2.2.10.2.8](#): This record type is only supported by Windows 7 and Windows Server 2008 R2.

<70> [Section 2.2.10.2.9](#): Windows 2000 and Windows Server 2003 DNS servers do not include this field.

<71> [Section 2.2.10.2.9](#): Windows 2000 DNS servers do not include this field.

<72> [Section 2.2.10.2.10](#): Windows 2000 DNS servers do not include this field.

<73> [Section 2.2.10.2.14](#): Windows 2000 does not include this field.

<74> [Section 2.2.10.2.20](#): The following elements were added in Windows Server 2003: **PacketsForNsListUsed**, **PacketsForNsListReturned** and **PacketsForNsListInUse**. The Windows 2000 version of this structure does not contain these elements.

<75> [Section 2.2.10.2.20](#): Windows 2000 DNS servers do not include this field.

<76> [Section 2.2.12.1.1](#): The **DNS\_ZONE\_STATS\_TYPE** enumerator is available in Windows Server 2012 R2 only.

<77> [Section 2.2.12.2.1](#): The **DNSSRV\_ZONE\_TIME\_STATS** structure is available in Windows 8.1 and Windows Server 2012 R2 only.

<78> [Section 2.2.12.2.2](#): The **DNSSRV\_ZONE\_QUERY\_STATS** structure is available in Windows 8.1 and Windows Server 2012 R2 only.

<79> [Section 2.2.12.2.3](#): The **DNSSRV\_ZONE\_TRANSFER\_STATS** structure is available in Windows 8.1 and Windows Server 2012 R2 only.

<80> [Section 2.2.12.2.4](#): The **DNSSRV\_ZONE\_UPDATE\_STATS** structure is available in Windows 8.1 and Windows Server 2012 R2 only.

<81> [Section 2.2.12.2.5](#): The **DNS\_RPC\_ZONE\_STATS\_V1** structure is available in Windows 8.1 and Windows Server 2012 R2 only.

<82> [Section 2.2.13.2.1](#): The **DNS\_RPC\_ENUM\_ZONE\_SCOPE\_LIST** structure is available in Windows 8.1 and Windows Server 2012 R2 only.

<83> [Section 2.2.13.2.2.1](#): The **DNS\_RPC\_ZONE\_SCOPE\_CREATE\_INFO\_V1** structure is available in Windows 8.1 and Windows Server 2012 R2 only.

<84> [Section 2.2.13.2.3.1](#): The **DNS\_RPC\_ZONE\_SCOPE\_INFO\_V1** structure is available in Windows 8.1 and Windows Server 2012 R2 only.

<85> [Section 2.3](#): The dnsProperty and dnsRecord attributes, and their associated properties, are not supported on Windows NT 4.0.

<86> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<87> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<88> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<89> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<90> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<91> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<92> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<93> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<94> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<95> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<96> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<97> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<98> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<99> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<100> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<101> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<102> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<103> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<104> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<105> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<106> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<107> [Section 2.3](#): Supported only on Windows Server 2012 and Windows Server 2012 R2.

<108> [Section 2.3.1.1](#): This object class is supported only by Windows Server 2012 and Windows Server 2012 R2.

<109> [Section 2.3.2.1.1](#): The following table lists dnsProperty Ids that are supported under different versions of Windows Server.

Property Name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
DSPROPERTY_ZONE_TYPE		X	X	X	X	X	X
DSPROPERTY_ZONE_ALLOW_UPDATE		X	X	X	X	X	X
DSPROPERTY_ZONE_SECURE_TIME		X	X	X	X	X	X
DSPROPERTY_ZONE_NONREFRESH_INTERVAL		X	X	X	X	X	X
DSPROPERTY_ZONE_REFRESH_INTERVAL		X	X	X	X	X	X
DSPROPERTY_ZONE_AGING_STATE		X	X	X	X	X	X
DSPROPERTY_ZONE_SCAVENGING_SERVERS		X	X	X	X	X	X
DSPROPERTY_ZONE_DELETED_FROM_HOSTNAME		X	X				
DSPROPERTY_ZONE_AGING_ENABLED_TIME				X	X	X	X
DSPROPERTY_ZONE_MASTER_SERVERS				X	X	X	X
DSPROPERTY_ZONE_AUTO_NAMES_SERVERS			X	X	X	X	X
DSPROPERTY_ZONE_DCPROMO_CONVERT			X	X	X	X	X



Property Name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
DSPROPERTY_ZONE_SCAVENGING_SERVERS_DA				X	X	X	X
DSPROPERTY_ZONE_MASTER_SERVERS_DA				X	X	X	X
DSPROPERTY_ZONE_AUTO_NS_SERVERS_DA				X	X	X	X
DSPROPERTY_ZONE_NODE_D_BFLAGS		X	X	X	X	X	X

<110> [Section 2.3.2.1.1:](#) The DNS Server does not write the DSPROPERTY\_ZONE\_SCAVENGING\_SERVERS [propertyId](#) if ForceForestBehaviorVersion (section [3.1.1.1.1](#)) indicates a forest behavior version corresponding to Windows NT, Windows 2000 Server, or Windows Server 2003.

<111> [Section 2.3.2.1.1:](#) Windows 2000 Server and Windows Server 2003 initialize this value with the hostname of the server when the zone is being deleted and preserve the value at all other times. Windows Server 2008 and Windows Server 2008 R2 ignore this value. Windows NT Server 4.0 does not support this structure. The hostname written is the FQDN of the local machine, as determined by the GetComputerNameExW system call.

<112> [Section 2.3.2.1.1:](#) The DNS Server does not write the DSPROPERTY\_ZONE\_MASTER\_SERVERS [propertyId](#) if ForceForestBehaviorVersion (section [3.1.1.1.1](#)) indicates a forest behavior version corresponding to Windows NT, Windows 2000 Server, or Windows Server 2003.

<113> [Section 2.3.2.1.1:](#) The DNS Server does not write the DSPROPERTY\_ZONE\_AUTO\_NS\_SERVERS [propertyId](#) if ForceForestBehaviorVersion (section [3.1.1.1.1](#)) indicates a forest behavior version corresponding to Windows NT, Windows 2000 Server, or Windows Server 2003.

<114> [Section 2.3.2.1.1:](#) Windows 2000 and Windows Server 2003 do not read or write [Property Id](#) DSPROPERTY\_ZONE\_SCAVENGING\_SERVERS\_DA.

<115> [Section 2.3.2.1.1:](#) Windows 2000 and Windows Server 2003 do not read or write [Property Id](#) DSPROPERTY\_ZONE\_MASTER\_SERVERS\_DA.

<116> [Section 2.3.2.1.1:](#) Windows 2000 and Windows Server 2003 do not read or write [Property Id](#) DSPROPERTY\_ZONE\_AUTO\_NS\_SERVERS\_DA.

<117> [Section 2.3.2.1.2:](#) The DcPromo flags are supported only on Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2.

<118> [Section 2.3.2.1.2:](#) The RODC mode check is supported only on Windows Server 2008 and Windows Server 2008 R2.

<119> [Section 3.1.1:](#) All versions of Windows Server listed in the supported products list in [Appendix B: Product Behavior \(section 7\)](#) except Windows NT Server 4.0 (which predates Active



Directory) use "cn=MicrosoftDNS,cn=System" as the constant container relative distinguished name. A complete DS Container string could, for example, be "cn=MicrosoftDNS,cn=System,DC=corp,DC=contoso,DC=com". The access control list is stored in the `ntSecurityDescriptor` attribute of this container and can be modified using standard LDAP modify operations (see [\[MS-ADTS\]](#) section 3.1.1.5.3).

[<120> Section 3.1.1:](#) In all versions of Windows, this access control list by default grants Full Control to the Domain Administrators group, Full Control to members of the "DnsAdmins" group, and Full Control to members of the Enterprise Domain Controllers group if the DNS server is active directory integrated, and Full control to the Administrators group and the System Operators group otherwise.

[<121> Section 3.1.1:](#) All versions of Windows Server listed in the supported products list in [Appendix B: Product Behavior](#) except Windows NT Server 4.0 (which predates Active Directory) use the **dnsTombstoned** attribute to store DNS Record Tombstone State in the directory server. A value of "TRUE" indicates that the node is a tombstone. Any other value indicates that the node is not a tombstone. No version of the Windows Server supports DNS Record Tombstone state for zones that are not stored in the directory server.

[<122> Section 3.1.1:](#) In Windows 2000, Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2, this Access Control List by default grants Full Control to the Domain Administrators Group, Create All Child Objects privilege to Authenticated Users, and Read privilege to Everyone. If the zone is not stored in the DNS Forest Partition in the directory server, Full Control is also granted to the "DnsAdmins" group. In Windows Server 2008 and Windows Server 2008 R2, Full Control is also granted to members of the Enterprise Domain Controllers group, and Container Inheritance is enabled. In all other versions of Windows, Container Inheritance is not enabled.

[<123> Section 3.1.1:](#) Only Windows Server 2012 and Windows Server 2012 R2 support this element.

[<124> Section 3.1.1:](#) Only Windows Server 2012 and Windows Server 2012 R2 support this element.

[<125> Section 3.1.1:](#) The Windows 2000 DNS server does not implement an Application Directory Partition Table and does not support any operations related to application directory partitions.

[<126> Section 3.1.1:](#) All versions of Windows Server listed in the supported products list in [Appendix B: Product Behavior](#) except Windows NT Server 4.0 (which predates Active Directory) and Windows 2000 (which predates Application Directory Partitions) use "CN=MicrosoftDNS,CN=PartitionName" as the container relative distinguished name, where PartitionName is ForestDnsZones, DomainDnsZones, or a custom label specified by the administrator. A complete distinguished name for the object where this Access Control List is stored could be, for example, "CN=MicrosoftDNS,CN=DomainDnsZones,DC=corp,DC=contoso,DC=com". The access control list is stored in the **ntSecurityDescriptor** attribute of this container and can be modified using standard LDAP modify operations (see [\[MS-ADTS\]](#) section 3.1.1.5.3).

By default this Access Control List grants Full Control to members of the "DnsAdmins" group, Full Control to members of the Enterprise Domain Controllers group, and if the name of this partition is not ForestDnsZones Full Control to members of the Domain Administrators group.

[<127> Section 3.1.1:](#) All versions of Windows Server listed in the supported products list in [Appendix B: Product Behavior](#) use the LocalSystem account as the default **DNS Server Credentials**.

[<128> Section 3.1.1:](#) All versions of Windows Server listed in the supported products list in [Appendix B: Product Behavior](#) except Windows NT 4.0 and Windows 2000 Server support

DownlevelDCsInDomain. All the supporting versions of the DNS servers acquire the value using the LDAP filter "&(objectCategory=ntdsDsa)(!(msDS-Behavior-Version>=x))(!(msDS-HasMasterNCs=y)(hasMasterNCs=y))", where x is the forest functional level value ("ms-DS-Behavior-Version: Forest Functional Level", [\[MS-ADTS\]](#) section 6.1.4.4) that corresponds to that of Windows Server 2003, and y is the domain partition value ("nTDSDSA Object", [\[MS-ADTS\]](#) section 6.1.1.2.2.1.2.1.1).

[<129> Section 3.1.1.1.1:](#) Range verification is only supported on Windows Server 2008 and Windows Server 2008 R2. In Windows NT 4.0, Windows 2000, and Windows Server 2003, the range is unlimited, unless otherwise specified for a property. On upgrade, from Windows NT 4.0, Windows 2000, or Windows Server 2003 to Windows Server 2008 or Windows Server 2008 R2, range verification is enforced on values set under the previous version. On upgrade, if the value is zero, but not in the new version's range, and the zero value is disallowed, then the default value is used.

[<130> Section 3.1.1.1.1:](#) In Windows NT 4.0 this property does not exist.

[<131> Section 3.1.1.1.1:](#) In Windows NT 4.0 this property does not exist.

[<132> Section 3.1.1.1.1:](#) In Windows NT 4.0 and Windows 2000 Server this property does not exist.

[<133> Section 3.1.1.1.1:](#) In Windows NT 4.0, Windows 2000 Server, and Windows Server 2003, the default value is 0x00000001.

[<134> Section 3.1.1.1.1:](#) Windows NT 4.0 does not implement this property.

[<135> Section 3.1.1.1.1:](#) Windows NT 4.0 does not support this property.

[<136> Section 3.1.1.1.1:](#) Windows NT 4.0 does not support this property.

[<137> Section 3.1.1.1.1:](#) Windows NT 4.0 and Windows 2000 Server do not support this property.

[<138> Section 3.1.1.1.1:](#) In Windows NT 4.0 and Windows 2000 Server, the default value is 0x0000012C (5 minutes).

[<139> Section 3.1.1.1.1:](#) In Windows NT 4.0, Windows 2000 Server, and Windows Server 2003, the default value is 0x00093A80 (7 days).

[<140> Section 3.1.1.1.1:](#) In Windows NT 4.0, Windows 2000, and Windows Server 2003, the default value is 0x00000005.

[<141> Section 3.1.1.1.1:](#) Windows NT 4.0 uses zero as the default value.

[<142> Section 3.1.1.1.1:](#) In Windows NT 4.0 this parameter is not implemented. In Windows 2000 Server, the default value is 0x04000000 (4 MB).

[<143> Section 3.1.1.1.1:](#) The following table lists DNS\_LOG\_LEVELS flags that are supported for different versions of Windows Server. Where a flag is unsupported, the flag will be stored but ignored.

Property Name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
DNS_LOG_LEVEL_ANSWERS	X	X	X	X	X	X	X
DNS_LOG_LEVEL_DS_UPDATE	X	X					
DNS_LOG_LEVEL_DS_WRITE	X	X					
DNS_LOG_LEVEL_FULL_PACKETS	X	X	X	X	X	X	X
DNS_LOG_LEVEL_NOTIFY	X	X	X	X	X	X	X
DNS_LOG_LEVEL_QUERY	X	X	X	X	X	X	X
DNS_LOG_LEVEL_QUESTIONS	X	X	X	X	X	X	X
DNS_LOG_LEVEL_RECV	X	X	X	X	X	X	X
DNS_LOG_LEVEL_SEND	X	X	X	X	X	X	X
DNS_LOG_LEVEL_TCP	X	X	X	X	X	X	X
DNS_LOG_LEVEL_UDP	X	X	X	X	X	X	X
DNS_LOG_LEVEL_UNMATCHED_RESPONSE					X	X	X
DNS_LOG_LEVEL_UPDATE	X	X	X	X	X	X	X
DNS_LOG_LEVEL_WRITE_THROUGH	X	X	X	X	X	X	X

<144> [Section 3.1.1.1.1](#): Windows NT 4.0 does not implement this property.

<145> [Section 3.1.1.1.1](#): This parameter is implemented only on Windows Server 2012 and Windows Server 2012 R2.

<146> [Section 3.1.1.1.1](#): In Windows NT 4.0 the default value is 0x00000003.

<147> [Section 3.1.1.1.1](#): Windows NT 4.0 does not implement this property.

<148> [Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

<149> [Section 3.1.1.1.1](#): In Windows NT 4.0, this property is not implemented.

<150> [Section 3.1.1.1.1](#): In Windows NT 4.0, this property is not supported.

[<151> Section 3.1.1.1.1](#): In Windows NT 4.0, Windows 2000 Server and Windows Server 2003, the default value is 0x0000000F.

[<152> Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented. In Windows Server 2003, the default value is 0x00000001.

[<153> Section 3.1.1.1.1](#): In Windows NT 4.0, Windows 2000 Server, Windows Server 2003, and Windows Server 2008, the default value is 0xFFFFFFFF.

[<154> Section 3.1.1.1.1](#): In Windows NT 4.0 the default value is 0.

[<155> Section 3.1.1.1.1](#): In Windows NT 4.0, this property is not implemented. In Windows 2000 and Windows Server 2003, the value's range MUST be unlimited, and the value zero MUST be treated as a flag value for 0xFFFFFFFF.

[<156> Section 3.1.1.1.1](#): Windows NT 4.0 does not implement this property.

[<157> Section 3.1.1.1.1](#): In Windows NT 4.0, this property is not implemented. In Windows 2000 Server, the default value is 0x00000001. In Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2, the default value is 0x00000002.

[<158> Section 3.1.1.1.1](#): In Windows NT 4.0, Windows 2000, Windows Server 2003, and Windows Server 2008, this value is not implemented.

[<159> Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<160> Section 3.1.1.1.1](#): Windows NT 4.0 does not implement this property.

[<161> Section 3.1.1.1.1](#): Windows NT Server 4.0 and Windows 2000 Server do not limit this value.

[<162> Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented. In Windows Server 2003, the default value is 0x0000000F, the minimum value is 0x00000003, the maximum value is 0x00000078, and values greater than the maximum or less than the minimum are treated as flag values for the maximum and minimum respectively.

[<163> Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented. In Windows Server 2003, the default value is 0x00000001.

[<164> Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<165> Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<166> Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<167> Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<168> Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented. In Windows Server 2003 and Windows Server 2008, the default value is 0x00015180 (1 day), and the allowed range is 0x00000E10 (1 hour) to 0x00EFF100 (182 days).

[<169> Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

<170> [Section 3.1.1.1.1](#): Windows Server 2003 and Windows Server 2008 processes DNSSEC based on [\[RFC2535\]](#). In Windows NT 4.0 and Windows 2000 Server, this property is not implemented. In Windows Server 2003 and Windows Server 2008, the value is an enumerated DWORD, with the permitted range between 0x00000000 to 0x00000002, inclusive. In Windows Server 2003 and Windows Server 2008, the meaning of the allowed values are indicated in the table that follows.

Name/Value	Meaning
DNS_DNSSEC_DISABLED 0x00000000	The server will not include DNSSEC information in responses.
DNS_DNSSEC_ENABLED_IF_EDNS 0x00000001	The server will include DNSSEC information in a response only if the client request had EDNS <a href="#">[RFC2671]</a> enabled.
DNS_DNSSEC_ENABLED_ALWAYS 0x00000002	The server will include DNSSEC information in a response whenever such information is available.

<171> [Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented. In Windows Server 2008 the default value is zero.

<172> [Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

<173> [Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented. Default values are as follows:

- Windows Server 2003: zero (FALSE)
- Windows Server 2008: nonzero value (TRUE)
- Windows Server 2008 R2: nonzero value (TRUE)

<174> [Section 3.1.1.1.1](#): This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

<175> [Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

<176> [Section 3.1.1.1.1](#): This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

<177> [Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

<178> [Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

<179> [Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

<180> [Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

<181> [Section 3.1.1.1.1](#): In Windows NT 4.0 and Windows 2000 Server, this property is not implemented. In Windows Server 2003 and Windows Server 2008, the default value is 0x00015180 (1 day).

[<182> Section 3.1.1.1.1.1:](#) In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<183> Section 3.1.1.1.1.1:](#) In Windows NT 4.0 and Windows 2000 Server, this property is not implemented. In Windows Server 2003, the default value is 0xFFFFFFFF (DNS\_SERVER\_UNLIMITED\_CACHE\_SIZE).

[<184> Section 3.1.1.1.1.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<185> Section 3.1.1.1.1.1:](#) In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<186> Section 3.1.1.1.1.1:](#) In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<187> Section 3.1.1.1.1.1:](#) In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<188> Section 3.1.1.1.1.1:](#) In Windows, the default value varies with the operating system version. In Windows Server 2003, and Windows Server 2008, the default value is 0x00000500. In Windows Server 2008 R2, the default value is 0x00000FA0.

[<189> Section 3.1.1.1.1.1:](#) This property is not supported in Windows NT 4.0 or Windows 2000 Server.

[<190> Section 3.1.1.1.1.1:](#) In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<191> Section 3.1.1.1.1.1:](#) Windows NT 4.0 and Windows 2000 do not support this property.

[<192> Section 3.1.1.1.1.1:](#) In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<193> Section 3.1.1.1.1.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<194> Section 3.1.1.1.1.1:](#) Windows NT 4.0, Windows 2000, and Windows Server 2003 do not support these properties.

[<195> Section 3.1.1.1.1.1:](#) In Windows NT 4.0, Windows 2000, Windows Server 2003, and Windows Server 2008 R2, this property is not implemented.

[<196> Section 3.1.1.1.1.1:](#) This property is not supported on Windows NT 4.0, Windows 2000 Server, or Windows Server 2003.

[<197> Section 3.1.1.1.1.1:](#) On Windows Server 2008 and Windows Server 2008 R2, the default value is 0x00000001. On Windows Server 2012 and Windows Server 2012 R2, the default value is 0x00000000.

[<198> Section 3.1.1.1.1.1:](#) This property is not supported in Windows NT 4.0, Windows 2000 Server, or on Windows Server 2003

[<199> Section 3.1.1.1.1.1:](#) The EnableGlobalQueryBlockList property is supported in Windows Server 2008 and Windows Server 2008 R2.

[<200> Section 3.1.1.1.1.1:](#) OpenACLOnProxyUpdates and CacheLockingPercentproperties are only supported in Windows Server 2008 R2.

[<201> Section 3.1.1.1.1:](#) ZoneWritebackInterval is only available in Windows Server 2012 R2.

[<202> Section 3.1.1.1.2:](#) Windows NT Server 4.0, Windows 2000 Server, and Windows Server 2003 use DNSSRV\_TYPEID\_IPARRAY for input and return values. Windows Server 2008 and Windows Server 2008 R2 accept DNSSRV\_TYPEID\_IPARRAY and DNSSRV\_TYPEID\_ADDRARRAY as input and output DNSSRV\_TYPEID\_ADDRARRAY unless dwClientVersion is used to request a previous format.

[<203> Section 3.1.1.1.2:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<204> Section 3.1.1.1.2:](#) Windows 2000 does not support these properties.

[<205> Section 3.1.1.1.3:](#) Windows 2000 does not support these properties.

[<206> Section 3.1.1.1.3:](#) All versions of Windows Server listed in the supported products list in [Appendix B: Product Behavior](#) store the log file relative to the "%SystemRoot%\System32" directory, if the path or filename given is not absolute.

[<207> Section 3.1.1.1.3:](#) Windows 2000 and Windows 2003 do not support these properties.

[<208> Section 3.1.1.1.3:](#) In Windows NT 4.0 and Windows 2000 Server, this property is not implemented.

[<209> Section 3.1.1.1.3:](#) This parameter is implemented only on Windows Server 2012 and Windows Server 2012 R2.

[<210> Section 3.1.1.1.4:](#) Windows 2000 and Windows Server 2003 do not support these properties.

[<211> Section 3.1.1.2.1:](#) On Windows, it is written into the system32\dns directory.

[<212> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<213> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<214> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<215> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<216> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<217> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<218> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<219> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<220> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.



[<221> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<222> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<223> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<224> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<225> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<226> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<227> Section 3.1.1.2.1:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<228> Section 3.1.1.2.1:](#) This property is not supported on Windows NT 4.0.

[<229> Section 3.1.1.2.1:](#) This property is not supported on Windows NT 4.0 and Windows 2000 Server. Furthermore, it is only supported on zones configured for forwarding.

[<230> Section 3.1.1.2.1:](#) This property is not supported on Windows NT 4.0 and Windows 2000 Server. Furthermore, it is only supported on zones configured for forwarding.

[<231> Section 3.1.1.2.1:](#) This property is supported only on Windows NT 4.0.

[<232> Section 3.1.1.2.1:](#) The PluginEnabled property of a zone is available in Windows Server 2012 R2 only.

[<233> Section 3.1.1.2.2:](#) Windows NT Server 4.0, Windows 2000 Server, and Windows Server 2003 use DNS\_SRV\_TYPEID\_IPARRAY for input and return values. Windows Server 2008 and Windows Server 2008 R2 accept DNS\_SRV\_TYPEID\_IPARRAY and DNS\_SRV\_TYPEID\_ADDRARRAY as input and output DNS\_SRV\_TYPEID\_ADDRARRAY unless dwClientVersion is used to request a previous format.

[<234> Section 3.1.1.2.2:](#) Windows 2000 does not support these properties.

[<235> Section 3.1.1.2.3:](#) Windows 2000 does not support these properties.

[<236> Section 3.1.1.2.3:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<237> Section 3.1.1.2.3:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<238> Section 3.1.1.2.3:](#) This property is supported only on Windows Server 2012 and Windows Server 2012 R2.

[<239> Section 3.1.1.2.5:](#) DNS Zone Scope Configuration Information is available in Windows 8.1 and Windows Server 2012 R2 only.

[<240> Section 3.1.1.2.5:](#) In Windows Server 2012 R2, the zone scopes inherit all the properties and configurations of the zone in which they are created unless overridden explicitly.



<241> [Section 3.1.3](#): Windows NT 4.0 does not support invocation of the "Netlogon" protocol implementation.

<242> [Section 3.1.4](#): Windows 2000 supports only opnums 0 through 4.

<243> [Section 3.1.4.1](#): In Windows 2000, Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2 the DNS server process might fail on the "Restart" command.

<244> [Section 3.1.4.1](#): Windows 2000 Server uses DNS\_SRV\_TYPEID\_ZONE\_CREATE\_W2K. Windows Server 2003 uses DNS\_SRV\_TYPEID\_ZONE\_CREATE\_DOTNET.

<245> [Section 3.1.4.1](#): Windows NT Server 4.0 and Windows 2000 Server return error 9611 ("invalid DNS zone type") for ZoneCreate operations with DNS\_ZONE\_TYPE\_STUB or DNS\_ZONE\_TYPE\_CACHE record types. All versions listed in the supported products list in [Appendix B: Product Behavior](#) return error 9611 for DNS\_ZONE\_TYPE\_CACHE and DNS\_ZONE\_TYPE\_SECONDARY\_CACHE.

<246> [Section 3.1.4.1](#): The Windows 2000 DNS server returns a failure for this value of pszOperation. The Windows Server 2003 DNS server accepts DWORD input only. The Windows Server 2003 DNS client sends DWORD input.

<247> [Section 3.1.4.1](#): Windows XP and Windows Server 2003 accept DNS\_SRV\_TYPEID\_IPARRAY and IP4\_ARRAY and do not accept DNS\_SRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY. Windows Server 2008 and Windows Server 2008 R2 accept DNS\_SRV\_TYPEID\_ADDR\_ARRAY and DNS\_ADDR\_ARRAY and do not accept DNS\_SRV\_TYPEID\_IPARRAY and IP4\_ARRAY.

<248> [Section 3.1.4.1](#): Windows 2000 Server uses DNS\_SRV\_TYPEID\_FORWARDERS\_W2K. Windows Server 2003 uses DNS\_SRV\_TYPEID\_FORWARDERS\_DOTNET.

<249> [Section 3.1.4.1](#): Windows NT 4.0 and Windows 2000 do not support this value. Windows XP and Windows Server 2003 accept DNS\_SRV\_TYPEID\_IPARRAY and IP4\_ARRAY and do not accept DNS\_SRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY. Windows Server 2008 and Windows Server 2008 R2 accept DNS\_SRV\_TYPEID\_ADDR\_ARRAY and DNS\_ADDR\_ARRAY and do not accept DNS\_SRV\_TYPEID\_IPARRAY and IP4\_ARRAY.

<250> [Section 3.1.4.1](#): Windows NT 4.0 and Windows 2000 do not support this value. Windows XP and Windows Server 2003 accept DNS\_SRV\_TYPEID\_IPARRAY and IP4\_ARRAY and do not accept DNS\_SRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY. Windows Server 2008 and Windows Server 2008 R2 accept DNS\_SRV\_TYPEID\_ADDR\_ARRAY and DNS\_ADDR\_ARRAY and do not accept DNS\_SRV\_TYPEID\_IPARRAY and IP4\_ARRAY.

<251> [Section 3.1.4.1](#): Windows NT 4.0 and Windows 2000 do not support this value. Windows XP and Windows Server 2003 accept DNS\_SRV\_TYPEID\_IPARRAY and IP4\_ARRAY and do not accept DNS\_SRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY. Windows Server 2008 and Windows Server 2008 R2 accept DNS\_SRV\_TYPEID\_ADDR\_ARRAY and DNS\_ADDR\_ARRAY and do not accept DNS\_SRV\_TYPEID\_IPARRAY and IP4\_ARRAY.

<252> [Section 3.1.4.1](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<253> [Section 3.1.4.1](#): A Windows implementation allows DNS\_ZONE\_NOTIFY\_LEVEL to be set only to values 0x0 through 0x1. Attempts to set this property to a higher value result in the effective value 0x1.

<254> [Section 3.1.4.1](#): Windows 2000 Server uses DNS\_SRV\_TYPEID\_ZONE\_CREATE\_W2K, and cannot convert from other types. Windows Server 2003 uses DNS\_SRV\_TYPEID\_ZONE\_CREATE\_DOTNET, and can convert from

DNSSRV\_TYPEID\_ZONE\_CREATE\_W2K. Windows Server 2008 and Windows Server 2008 R2 use DNSSRV\_TYPEID\_ZONE\_CREATE and can convert from DNSSRV\_TYPEID\_ZONE\_CREATE\_W2K and DNSSRV\_TYPEID\_ZONE\_CREATE\_DOTNET.

<255> [Section 3.1.4.1](#): Windows 2000 Server uses DNSSRV\_TYPEID\_ZONE\_DATABASE\_W2K. Windows Server 2003 uses DNSSRV\_TYPEID\_ZONE\_DATABASE.

<256> [Section 3.1.4.1](#): On Windows Server 2008 and Windows Server 2008 R2, if ForceForestBehaviorVersion (section [3.1.1.1.1](#)) indicates a forest behavior version of Windows Server 2008 or Windows Server 2008 R2, the server writes only DNS\_ADDR\_ARRAY values to the directory server. Otherwise, the server writes both IP4\_ARRAY and DNS\_ADDR\_ARRAY values. Windows NT 4.0, Windows 2000 Server, and Windows Server 2003 do not support this forest version check, and write only IP4\_ARRAY values to the directory server.

<257> [Section 3.1.4.1](#): Windows 2000 and Windows Server 2003 accept DNSSRV\_TYPEID\_IPARRAY and IP4\_ARRAY and silently disregard DNSSRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY. They also write only the IP4\_ARRAY value to the directory server if the server is DS-integrated; when reading from **DS**, only the IP4\_ARRAY value is read, and any DNS\_ADDR\_ARRAY values are ignored. Windows Server 2008 and Windows Server 2008 R2 accept DNSSRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY and do not accept DNSSRV\_TYPEID\_IPARRAY and IP4\_ARRAY, and write both formats to the directory server if the server is DS-integrated; when reading from DS, the DNS\_ADDR\_ARRAY value is read if it exists; otherwise the IP4\_ARRAY value is read.

<258> [Section 3.1.4.1](#): Windows 2000 does not support this operation. Windows Server 2003 accepts DNSSRV\_TYPEID\_IPARRAY and IP4\_ARRAY and does not accept DNSSRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY. Windows Server 2008 and Windows Server 2008 R2 accept DNSSRV\_TYPEID\_ADDR\_ARRAY and DNS\_ADDR\_ARRAY and do not accept DNSSRV\_TYPEID\_IPARRAY and IP4\_ARRAY.

<259> [Section 3.1.4.1](#): Windows 2000 Server uses DNSSRV\_TYPEID\_ZONE\_SECONDARIES\_W2K. Windows Server 2003 uses DNSSRV\_TYPEID\_ZONE\_SECONDARIES\_DOTNET.

<260> [Section 3.1.4.1](#): On Windows Server 2008 and Windows Server 2008 R2, if ForceForestBehaviorVersion (section [3.1.1.1.1](#)) indicates a forest behavior version of Windows Server 2008 or Windows Server 2008 R2, the server writes only DNS\_ADDR\_ARRAY values to the directory server. Otherwise, the server writes both IP4\_ARRAY and DNS\_ADDR\_ARRAY values. Windows NT 4.0, Windows 2000 Server, and Windows Server 2003 do not support this forest version check, and write only IP4\_ARRAY values to the directory server.

<261> [Section 3.1.4.1](#): Windows 2000 and Windows Server 2003 accept DNSSRV\_TYPEID\_IPARRAY and IP4\_ARRAY and silently disregard DNSSRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY. They also write only the IP4\_ARRAY value to the directory server if the server is DS-integrated; when reading from DS, only the IP4\_ARRAY value is read, and any DNS\_ADDR\_ARRAY values are ignored. Windows Server 2008 and Windows Server 2008 R2 accept DNSSRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY and do not accept DNSSRV\_TYPEID\_IPARRAY and IP4\_ARRAY, and write both formats to the directory server if the server is DS-integrated; when reading from DS, the DNS\_ADDR\_ARRAY value is read if it exists; otherwise the IP4\_ARRAY value is read.

<262> [Section 3.1.4.1](#): On Windows Server 2008 and Windows Server 2008 R2, if ForceForestBehaviorVersion (section [3.1.1.1.1](#)) indicates a forest behavior version of Windows Server 2008 or Windows Server 2008 R2, the server writes only DNS\_ADDR\_ARRAY values to the directory server. Otherwise, the server writes both IP4\_ARRAY and DNS\_ADDR\_ARRAY values. Windows NT 4.0, Windows 2000 Server, and Windows Server 2003 do not support this forest version check, and write only IP4\_ARRAY values to the directory server.

<263> [Section 3.1.4.1](#): Windows 2000 and Windows Server 2003 accept DNSSRV\_TYPEID\_IPARRAY and IP4\_ARRAY and silently disregard DNSSRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY. They also write only the IP4\_ARRAY value to the directory server if the server is DS-integrated; when reading from DS, only the IP4\_ARRAY value is read, and any DNS\_ADDR\_ARRAY values are ignored. Windows Server 2008 and Windows Server 2008 R2 accept DNSSRV\_TYPEID\_ADDRARRAY and DNS\_ADDR\_ARRAY and do not accept DNSSRV\_TYPEID\_IPARRAY and IP4\_ARRAY, and write both formats to the directory server if the server is DS-integrated; when reading from DS, the DNS\_ADDR\_ARRAY value is read if it exists; otherwise the IP4\_ARRAY value is read.

<264> [Section 3.1.4.1](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<265> [Section 3.1.4.1](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<266> [Section 3.1.4.1](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<267> [Section 3.1.4.1](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<268> [Section 3.1.4.1](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<269> [Section 3.1.4.1](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<270> [Section 3.1.4.1](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<271> [Section 3.1.4.1](#): CreateZoneScope is available only in Windows Server 2012 R2.

<272> [Section 3.1.4.1](#): DeleteZoneScope is available only in Windows Server 2012 R2.

<273> [Section 3.1.4.1](#): In Windows NT 4.0, Windows 2000 Server, and Windows Server 2003, no range limiting or zero/nonzero restrictions are applied.

<274> [Section 3.1.4.1](#): The following table lists property names that are supported as an input to the pszOperation parameter for different versions of Windows Server.

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
AdditionalRecursionTimeout			X	X	X	X	X
AddressAnswerLimit	X	X	X	X	X	X	X
AdminConfigured		X	X	X	X	X	X
AllowCNAMEAtNS		X	X	X	X	X	X
AllowMsdcsLookupRetry				X			

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
AllowReadOnlyZoneTransfer				X	X	X	X
AllowUpdate	X	X	X	X	X	X	X
AppendMsZoneTransferTag			X	X	X	X	X
AutoCacheUpdate	X	X	X	X	X	X	X
AutoConfigFileZones			X	X	X	X	X
AutoCreateDelegations			X	X	X	X	X
BindSecondaries	X	X	X	X	X	X	X
BootMethod	X	X	X	X	X	X	X
BreakOnAscFailure			X	X	X	X	X
CacheEmptyAuthResponses			X	X	X	X	X
CacheLockingPercent					X	X	X
DebugLevel	X	X	X	X	X	X	X
DefaultAgingState		X	X	X	X	X	X
DefaultNoRefreshInterval		X	X	X	X	X	X
DefaultRefreshInterval		X	X	X	X	X	X
DeleteOutsideGlue	X	X	X	X	X	X	X
DirectoryPartitionAutoEnlistInterval			X	X	X	X	X
DisjointNets	X	X	X	X	X	X	X
DsBackgroundLoadPaused				X	X	X	X
DsLazyUpdateInterval			X	X	X	X	X
DsMinimumBackgroundLoadThreads				X	X	X	X
DsPollingInterval	X	X	X	X	X	X	X
DsRemoteReplicationDelay				X	X	X	X
DsTombstoneInterval	X	X	X	X	X	X	X

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
EDnsCacheTimeout			X	X	X	X	X
EnableDirectoryPartitions			X	X	X	X	X
EnableDnsSec			X	X	X	X	X
EnableDuplicateQuerySuppression				X	X	X	X
EnableEDnsProbes			X	X	X	X	X
EnableEDnsReception			X	X	X	X	X
EnableForwarderReordering						X	X
EnableGlobalNamesSupport				X	X	X	X
EnableIPv6			X	X	X	X	X
EnableIQueryResponseGeneration			X	X	X	X	X
EnableOnlineSigning						X	X
EnableRegistryBoot	X	X	X	X	X	X	X
EnableRsoForRodc				X	X	X	X
EnableSendErrorSuppression			X	X	X	X	X
EnableUpdateForwarding			X	X	X	X	X
EnableVersionQuery				X	X	X	X
EnableWinsR			X	X	X	X	X
EventLogLevel	X	X	X	X	X	X	X
ForceDomainBehaviorVersion			X	X	X	X	X
ForceDsaBehaviorVersion			X	X	X	X	X
ForceForestBehaviorVersion			X	X	X	X	X
ForceRODCMode				X	X	X	X
ForceSoaExpire	X	X	X	X	X	X	X

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
ForceSoaMinimumTtl	X	X	X	X	X	X	X
ForceSoaRefresh	X	X	X	X	X	X	X
ForceSoaRetry	X	X	X	X	X	X	X
ForceSoaSerial	X	X	X	X	X	X	X
ForwardDelegations	X	X	X	X	X	X	X
ForwardingTimeout	X	X	X	X	X	X	X
GlobalNamesAlwaysQuerySrv				X	X	X	X
GlobalNamesBlockUpdates				X	X	X	X
GlobalNamesEnableEDnsProbes				X	X	X	X
GlobalNamesPreferAAAA				X	X	X	X
GlobalNamesQueryOrder				X	X	X	X
GlobalNamesSendTimeout				X	X	X	X
GlobalNamesServerQueryInterval				X	X	X	X
HeapDebug			X	X	X	X	X
IsSlave	X	X	X	X	X	X	X
LameDelegationTtl			X	X	X	X	X
LocalNetPriority	X	X	X	X	X	X	X
LocalNetPriorityNetMask			X	X	X	X	X
LogFileMaxSize		X	X	X	X	X	X
LogLevel	X	X	X	X	X	X	X
LooseWildcarding	X	X	X	X	X	X	X
MaxCacheSize			X	X	X	X	X
MaxCacheTtl	X	X	X	X	X	X	X
MaximumRodcRsoAttemptsPerCycle				X	X	X	X

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
MaximumRodcRsoQueueLength				X	X	X	X
MaximumSignaturesScanPeriod						X	X
MaximumUdpPacketSize			X	X	X	X	X
MaxNegativeCacheTtl		X	X	X	X	X	X
MaxResourceRecordsInNonSecureUpdate			X	X	X	X	X
MaxTrustAnchorActiveRefreshInterval						X	X
NameCheckFlag	X	X	X	X	X	X	X
NoRecursion	X	X	X	X	X	X	X
NoUpdateDelegations		X	X	X	X	X	X
OpenACLOnProxyUpdates					X	X	X
OperationsLogLevel			X	X	X	X	X
OperationsLogLevel2			X	X	X	X	X
PublishAutonet			X	X	X	X	X
QuietRecvFaultInterval		X	X	X	X	X	X
QuietRecvLogInterval		X	X	X	X	X	X
RecurseToInternetRootMask			X	X	X	X	X
RecursionRetry	X	X	X	X	X	X	X
RecursionTimeout	X	X	X	X	X	X	X
ReloadException			X	X	X	X	X
RemoteIPv4RankBoost				X	X	X	X
RemoteIPv6RankBoost				X	X	X	X
RoundRobin	X	X	X	X	X	X	X
RpcProtocol	X	X	X	X	X	X	X
ScavengingInterval		X	X	X	X	X	X

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
SecureResponses	X	X	X	X	X	X	X
SelfTest			X	X	X	X	X
SendPort	X	X	X	X	X	X	X
SilentlyIgnoreCNameUpdateConflicts			X	X	X	X	X
SocketPoolSize		X	X	X	X	X	X
StrictFileParsing	X	X	X	X	X	X	X
SyncDsZoneSerial		X	X	X	X	X	X
TcpReceivePacketSize			X	X	X	X	X
UdpRecvThreadCount						X	X
UpdateOptions			X	X	X	X	X
UseSystemEventLog	X	X	X	X	X	X	X
Version				X	X	X	X
WriteAuthorityNs	X	X	X	X	X	X	X
XfrConnectTimeout		X	X	X	X	X	X
XfrThrottleMultiplier			X	X	X	X	X
ZoneWritebackInterval							X

<275> [Section 3.1.4.1](#): The following table lists property names that are supported as an input to the *pszOperation* parameter for different versions of Windows Server.

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
Forwarders	X	X	X	X	X	X	X
ListenAddresses	X	X	X	X	X	X	X
BreakOnReceiveFrom			X	X	X	X	X
BreakOnUpdateFrom			X	X	X	X	X
DomainDirectoryPartitio			X	X	X	X	X



Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
nBaseName							
DynamicForwarders						X	X
ForestDirectoryPartitionBaseName			X	X	X	X	X
LogFilePath			X	X	X	X	X
LogIPFilterList			X	X	X	X	X
ServerLevelPluginDll			X	X	X	X	X
GlobalQueryBlockList				X	X	X	X
RootTrustAnchorsURL						X	X
SocketPoolExcludedPortRanges					X	X	X
DsBackgroundPauseName						X	X

<276> [Section 3.1.4.1](#): zone scopes are available only in Windows Server 2012 R2.

<277> [Section 3.1.4.1](#): All versions of Windows Server listed in the supported products list in [Appendix B: Product Behavior](#) will attempt to backup the log file to the "%SYSTEMROOT%\System32\dns\backup\" directory.

<278> [Section 3.1.4.1](#): If ForceDomainBehaviorVersion (section [3.1.1.1.1](#)) indicates a domain behavior version of Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, or Windows Server 2012 R2, root hints MUST be written to the DNS domain partition. Otherwise, root hints MUST be written to the default application directory partition.

<279> [Section 3.1.4.1](#): If ForceDomainBehaviorVersion (section [3.1.1.1.1](#)) indicates a domain behavior version of Windows NT or Windows 2000 Server, stub and forwarder zones MUST NOT be created in the default application directory partition. If this partition is specified during ZoneCreate, the server MUST return a failure.

<280> [Section 3.1.4.1](#): Windows 2000 does not support this operation.

<281> [Section 3.1.4.1](#): Windows 2000 Server and Windows Server 2003 do not support RODCs and do not process the **msDS-NC-RO-Replica-Locations**.

<282> [Section 3.1.4.1](#): Windows 2000 Server and Windows Server 2003 do not support RODCs and do not process the **msDS-NC-RO-Replica-Locations**.

<283> [Section 3.1.4.1](#): Windows 2000 Server and Windows Server 2003 do not support RODCs and do not process the **msDS-NC-RO-Replica-Locations**.

<284> [Section 3.1.4.1](#): Windows 2000 does not support this operation. Windows Server 2003 takes a DWORD value for pData input parameter.

<285> [Section 3.1.4.1](#): Windows 2000 and Windows Server 2003 do not support this operation.

<286> [Section 3.1.4.1](#): Windows 2000 and Windows Server 2003 do not support this operation.

<287> [Section 3.1.4.1](#): Windows 2000 and Windows Server 2003 do not support this operation.

<288> [Section 3.1.4.1](#): If ForceDomainBehaviorVersion (section [3.1.1.1.1](#)) indicates a domain behavior version of Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, or Windows Server 2012 R2, root hints MUST be written to the DNS domain partition. Otherwise, root hints MUST be written to the default application directory partition.

<289> [Section 3.1.4.1](#): The Windows NT Server, Windows 2000 Server, Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2 operating systems store the log file relative to the "%SystemRoot%\System32" directory, if the path or filename given is not absolute.

<290> [Section 3.1.4.1](#): Windows NT 4.0 does not support invocation of the "Netlogon" protocol implementation.

<291> [Section 3.1.4.1](#): Aging is not supported on Windows NT Server 4.0.

<292> [Section 3.1.4.1](#): The following table lists the property names that are supported as input for the "ResetDwordProperty" operation when pszZone is not NULL, for different versions of Windows Server.

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
AllowUpdate	X	X	X	X	X	X	X
SecureSecondaries	X	X	X	X	X	X	X
NotifyLevel		X	X	X	X	X	X
LogUpdates	X	X	X	X	X	X	X
NoRefreshInterval		X	X	X	X	X	X
RefreshInterval		X	X	X	X	X	X
Aging		X	X	X	X	X	X
ForwarderSlave			X	X	X	X	X
ForwarderTimeout			X	X	X	X	X
Unicode	X						
DsRecordAlgorithms						X	X
DNSKEYRecordSetTTL						X	X
DsRecordSetTTL						X	X
IsKeymaster						X	X

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
IsSigned						X	X
MaintainTrustAnchor						X	X
NSEC3HashAlgorithm						X	X
NSEC3Iterations						X	X
NSEC3OptOut						X	X
NSEC3RandomSaltLength						X	X
ParentHasSecureDelegation						X	X
PropagationTime						X	X
RFC5011KeyRollovers						X	X
SecureDelegationPollingPeriod						X	X
SignatureInceptionOffset						X	X
SignWithNSEC3						X	X
PluginEnabled							X

<293> [Section 3.1.4.1](#): The following table lists property names that are supported as an input to the pszOperation parameter for different versions of Windows Server.

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
Masters	X	X	X	X	X	X	X
Secondaries	X	X	X	X	X	X	X
TypeReset	X	X	X	X	X	X	X
DatabaseFile	X	X	X	X	X	X	X
AllowAutoNS		X	X	X	X	X	X
ScavengeServe		X	X	X	X	X	X

Property name	Windows NT 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
rs							
BreakOnNameUpdate			X	X	X	X	X
ChangeDP			X	X	X	X	X
LocalMasters			X	X	X	X	X
NotifyList							
Keymaster						X	X
NSEC3UserSalt						X	X
NSEC3CurrentSalt						X	X

<294> [Section 3.1.4.1](#): If ForceDomainBehaviorVersion (section [3.1.1.1.1](#)) indicates a domain behavior version of Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, or Windows Server 2012 R2, root hints MUST be written to the DNS domain partition. Otherwise, root hints MUST be written to the default application directory partition.

<295> [Section 3.1.4.1](#): If ForceDomainBehaviorVersion (section [3.1.1.1.1](#)) indicates a domain behavior version of Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, or Windows Server 2012 R2, root hints MUST be written to the DNS domain partition. Otherwise, root hints MUST be written to the default application directory partition.

<296> [Section 3.1.4.1](#): If ForceDomainBehaviorVersion (section [3.1.1.1.1](#)) indicates a domain behavior version of Windows NT or Windows 2000 Server, the server MUST also verify that either the specified zone is not a stub or forwarder zone, or the destination application directory partition is not the default application directory partition. Otherwise, the server MUST return a failure.

<297> [Section 3.1.4.1](#): Windows 2000 Server does not implement this operation and therefore will return a failure.

<298> [Section 3.1.4.1](#): If ForceDomainBehaviorVersion (section [3.1.1.1.1](#)) indicates a domain behavior version of Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, or Windows Server 2012 R2, root hints MUST be written to the DNS domain partition. Otherwise, root hints MUST be written to the default application directory partition.

<299> [Section 3.1.4.2](#): The following table lists various DNS\_SRV\_TYPEID\_SERVER\_INFO values returned by **R\_DnsSrvQuery()**, **R\_DnsSrvQuery2()** methods, for different versions of Windows Server.

dwClientVersion	Windows NT Server 4.0	Windows 2000 Server	Windows Server 2003	Windows Server 2008	Windows Server 2008 with SP2
0x00000000		DNSSRV_TYPEID_SERVER_INFO_W2K	DNSSRV_TYPEID_SERVER_INFO_W2K	DNSSRV_TYPEID_SERVER_INFO_W2K	DNSSRV_TYPEID_SERVER_INFO_W2K
0x00060000		DNSSRV_TYPEID_SERVER_INFO_W2K	DNSSRV_TYPEID_SERVER_INFO_DOTNET	DNSSRV_TYPEID_SERVER_INFO_DOTNET	DNSSRV_TYPEID_SERVER_INFO_DOTNET
0x00070000		DNSSRV_TYPEID_SERVER_INFO_W2K	DNSSRV_TYPEID_SERVER_INFO_DOTNET	DNSSRV_TYPEID_SERVER_INFO	DNSSRV_TYPEID_SERVER_INFO

<300> [Section 3.1.4.2](#): Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2 set the dwLocalNetPriorityNetMask field of DNS\_RPC\_SERVER\_INFO to zero, regardless of the effective value of dwLocalNetPriorityNetMask.

<301> [Section 3.1.4.2](#): Windows 2000 Server uses DNSSRV\_TYPEID\_ZONE\_W2K. Windows Server 2003 uses DNSSRV\_TYPEID\_ZONE.

<302> [Section 3.1.4.2](#): Windows 2000 Server uses DNSSRV\_TYPEID\_ZONE\_INFO\_W2K. Windows Server 2003 uses DNSSRV\_TYPEID\_ZONE\_INFO\_DOTNET.

<303> [Section 3.1.4.2](#): Windows Server 2003 does not support the "Version" property with this operation.

<304> [Section 3.1.4.2](#): Windows 2000 and Windows Server 2003 use DNSSRV\_TYPEID\_IPARRAY and IP4\_ARRAY.

<305> [Section 3.1.4.2](#): Windows Server 2003 does not support the "Forwarders" and "ListenAddresses" properties here.

<306> [Section 3.1.4.2](#): All Windows Server versions incorrectly set pdwTypeId to DNSSRV\_TYPEID\_DWORD, and truncate ppData to DWORD size when R\_DnsSrvQuery is called with pszOperation set to "ListenAddresses" or "Forwarders".

<307> [Section 3.1.4.2](#): All Windows Server versions that have the "DsBackgroundPauseName" property incorrectly set pdwTypeId to DNSSRV\_TYPEID\_DWORD, and truncate ppData to DWORD size when R\_DnsSrvQuery is called with pszOperation set to "DsBackgroundPauseName".

<308> [Section 3.1.4.2](#): Windows 2000 and Windows Server 2003 use DNSSRV\_TYPEID\_IPARRAY and IP4\_ARRAY.

<309> [Section 3.1.4.3](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<310> [Section 3.1.4.3](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<311> [Section 3.1.4.3](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<312> [Section 3.1.4.3](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<313> [Section 3.1.4.3](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<314> [Section 3.1.4.3](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<315> [Section 3.1.4.3](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<316> [Section 3.1.4.3](#): This operation is supported only on Windows Server 2012 and Windows Server 2012 R2.

<317> [Section 3.1.4.3](#): ExportZoneSigningSettings is available only in Windows Server 2012 R2.

<318> [Section 3.1.4.3](#): ImportZoneSigningSettings is available only in Windows Server 2012 R2.

<319> [Section 3.1.4.3](#): EnumZoneScopes is available only in Windows Server 2012 R2.

<320> [Section 3.1.4.3](#): ZoneStatistics is available only in Windows Server 2012 R2.

<321> [Section 3.1.4.5](#): Windows 2000 does not support this operation. No version of Windows Server supports the DNS\_TYPE\_LOC for this operation.

<322> [Section 3.1.4.5](#): Windows 2000, Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2 do not support updates or deletions of the DNS\_TYPE\_ZERO, DNS\_TYPE\_LOC, and DNS\_TYPE\_ALL types.

<323> [Section 3.1.4.5](#): Windows 2000 does not allow additions with pszZoneName "..Cache" and treats pszZoneName NULL as "..RootHints". Windows Server 2003, Windows Server 2008, and Windows 7 treat pszZoneName NULL and pszZoneName "..Cache" as pszZoneName "..RootHints".

<324> [Section 3.1.4.7](#): Windows 2000 does not support this operation.

<325> [Section 3.1.4.9](#): Windows 2000 does not support this operation.

<326> [Section 3.1.4.10](#): Windows 2000 does not support this operation. No version of Windows Server supports the DNS\_TYPE\_LOC for this operation.

<327> [Section 3.1.4.11](#): **R\_DnssrvUpdateRecord3** is only available in Windows 8.1 and Windows Server 2012 R2.

<328> [Section 3.1.4.11](#): The RPC method [R\\_DnssrvUpdateRecord3](#) is available only in Windows Server 2012 R2.

<329> [Section 3.1.4.12](#): **R\_DnssrvEnumRecords3** is only available in Windows 8.1 and Windows Server 2012 R2.

<330> [Section 3.1.4.12](#): The RPC method [R\\_DnssrvEnumRecords3](#) is available only in Windows Server 2012 R2.

<331> [Section 3.1.4.13](#): **R\_DnssrvOperation3** is only available in Windows 8.1 and Windows Server 2012 R2.

<332> [Section 3.1.4.13](#): The RPC method [R\\_DnssrvOperation3](#) is available only in Windows Server 2012 R2.

<333> [Section 3.1.4.14: R\\_DnssrvQuery3](#) is only available in Windows 8.1 and Windows Server 2012 R2.

<334> [Section 3.1.4.14:](#) The RPC method R\_DnssrvQuery3 is available only in Windows Server 2012 R2.

<335> [Section 5.1.1:](#) Windows Server 2008 and Windows Server 2008 R2 clients do not use RPC over named pipes

## 8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.



## 9 Index

- ["ECDSA\\_P256" 180](#)
- ["ECDSA\\_P384" 180](#)
- ["RSA" 180](#)
- [\\_DnsCacheStats packet 179](#)
- [\\_DnsDbaseStats packet 168](#)
- [\\_DnsMemoryStats packet 161](#)
- [\\_DnsNbstatStats packet 172](#)
- [\\_DnsPacketStats packet 170](#)
- [\\_DnsPrivateStats packet 173](#)
- [\\_DnsRecordStats packet 169](#)
- [\\_DnsTimeoutStats packet 166](#)
- [\\_ErrorStats packet 177](#)
  
- A**
- [Abstract data model 207](#)
- [Applicability 22](#)
  
- B**
- [BOOT\\_METHOD\\_DIRECTORY 67](#)
- [BOOT\\_METHOD\\_FILE 67](#)
- [BOOT\\_METHOD\\_REGISTRY 67](#)
- [BOOT\\_METHOD\\_UNINITIALIZED 67](#)
  
- C**
- [Capability negotiation 22](#)
- [Change tracking 368](#)
- [Client requirements 207](#)
- [Common data types 24](#)
- [Creating and deleting a DNS record in a zone scope 300](#)
- [Creating and deleting DNS record example 297](#)
- [Creating new zone example 296](#)
  
- D**
- [Data model - abstract 207](#)
- [Data types 24](#)
- [DCPROMO\\_CONVERT\\_DOMAIN 197](#)
- [DCPROMO\\_CONVERT\\_FOREST 197](#)
- [DCPROMO\\_CONVERT\\_NONE 197](#)
- [Deleting DNS record example 297](#)
- [Directory service schema elements 193](#)
- [DNS\\_ADD\\_USER packet 63](#)
- [DNS\\_ADDR packet 63](#)
- [DNS\\_ADDR structure 62](#)
- [DNS\\_ADDR\\_ARRAY structure 65](#)
- [DNS\\_ALLOW\\_ALL\\_NAMES 67](#)
- [DNS\\_ALLOW\\_MULTIBYTE\\_NAMES 67](#)
- [DNS\\_ALLOW\\_NONRFC\\_NAMES 67](#)
- [DNS\\_ALLOW\\_RFC\\_NAMES\\_ONLY 67](#)
- [DNS\\_COUNT\\_NAME packet 41](#)
- [DNS\\_DP\\_AUTOCREATED 115](#)
- [DNS\\_DP\\_DELETED 115](#)
- [DNS\\_DP\\_DOMAIN\\_DEFAULT 115](#)
- [DNS\\_DP\\_ENLISTED 115](#)
- [DNS\\_DP\\_FOREST\\_DEFAULT 115](#)
- [DNS\\_DP\\_LEGACY 115](#)
- [DNS\\_FLAT\\_RECORD 57](#)
- [DNS\\_IPVAL\\_DNS\\_DELEGATIONS 61](#)
- [DNS\\_IPVAL\\_DNS\\_FORWARDERS 61](#)
- [DNS\\_IPVAL\\_DNS\\_ROOTHINTS 61](#)
- [DNS\\_IPVAL\\_DNS\\_SERVERS 61](#)
- [DNS\\_IPVAL\\_DNS\\_ZONE\\_MASTERS 61](#)
- [DNS\\_IPVAL\\_INVALID\\_ADDR 61](#)
- [DNS\\_IPVAL\\_NO\\_RESPONSE 61](#)
- [DNS\\_IPVAL\\_NO\\_TCP 61](#)
- [DNS\\_IPVAL\\_NOT\\_AUTH\\_FOR\\_ZONE 61](#)
- [DNS\\_IPVAL\\_UNKNOWN\\_ERROR 61](#)
- [DNS\\_IPVAL\\_UNREACHABLE 61](#)
- [DNS\\_LOG\\_LEVEL\\_ALL\\_PACKETS 123](#)
- [DNS\\_LOG\\_LEVEL\\_ANSWERS 123](#)
- [DNS\\_LOG\\_LEVEL\\_DS\\_UPDATE 123](#)
- [DNS\\_LOG\\_LEVEL\\_DS\\_WRITE 123](#)
- [DNS\\_LOG\\_LEVEL\\_FULL\\_PACKETS 123](#)
- [DNS\\_LOG\\_LEVEL\\_NOTIFY 123](#)
- [DNS\\_LOG\\_LEVEL\\_QUERY 123](#)
- [DNS\\_LOG\\_LEVEL\\_QUESTIONS 123](#)
- [DNS\\_LOG\\_LEVEL\\_RECV 123](#)
- [DNS\\_LOG\\_LEVEL\\_SEND 123](#)
- [DNS\\_LOG\\_LEVEL\\_TCP 123](#)
- [DNS\\_LOG\\_LEVEL\\_UDP 123](#)
- [DNS\\_LOG\\_LEVEL\\_UNMATCHED\\_RESPONSE 123](#)
- [DNS\\_LOG\\_LEVEL\\_UPDATE 123](#)
- [DNS\\_LOG\\_LEVEL\\_WRITE\\_THROUGH 123](#)
- [DNS\\_ROLLOVER\\_ACTION\\_DEFAULT 78](#)
- [DNS\\_ROLLOVER\\_ACTION\\_NORMAL 78](#)
- [DNS\\_ROLLOVER\\_ACTION\\_RETIRE 78](#)
- [DNS\\_ROLLOVER\\_ACTION\\_REVOKE\\_STANDBY 78](#)
- [DNS\\_ROLLOVER\\_TYPE\\_DOUBLE\\_SIGNATURE 78](#)
- [DNS\\_ROLLOVER\\_TYPE\\_PREPUBLISH 78](#)
- [DNS\\_RPC\\_AUTOCONFIG\\_ALL 121](#)
- [DNS\\_RPC\\_AUTOCONFIG\\_INTERNAL\\_FORWARDERS 121](#)
- [DNS\\_RPC\\_AUTOCONFIG\\_INTERNAL\\_RETURN\\_ERROR 121](#)
- [DNS\\_RPC\\_AUTOCONFIG\\_INTERNAL\\_ROOTHINTS 121](#)
- [DNS\\_RPC\\_AUTOCONFIG\\_INTERNAL\\_SELFPOINT 121](#)
- [DNS\\_RPC\\_AUTOCONFIG\\_INTERNAL\\_SELFPOINT\\_AP\\_PEND 121](#)
- [DNS\\_RPC\\_AUTOCONFIG\\_INTERNAL\\_SELFPOINT\\_EPEND 121](#)
- [DNS\\_RPC\\_AUTOCONFIG\\_INTERNAL\\_ZONES 121](#)
- [DNS\\_RPC\\_AUTOCONFIGURE structure 123](#)
- [DNS\\_RPC\\_BUFFER structure 31](#)
- [DNS\\_RPC\\_CURRENT\\_CLIENT\\_VER packet 30](#)
- [DNS\\_RPC\\_DP\\_ENUM structure 118](#)
- [DNS\\_RPC\\_DP\\_INFO structure 116](#)
- [DNS\\_RPC\\_DP\\_LIST structure 119](#)
- [DNS\\_RPC\\_DP\\_REPLICA structure 118](#)
- [DNS\\_RPC\\_ENLIST\\_DP structure 119](#)
- [DNS\\_RPC\\_ENUM\\_ZONE\\_SCOPE\\_LIST structure 191](#)
- [DNS\\_RPC\\_ENUM\\_ZONES\\_FILTER structure 98](#)
- [DNS\\_RPC\\_FLAG\\_AGING\\_ON 40](#)

[DNS RPC FLAG AUTH\\_ZONE\\_ROOT](#) 40  
[DNS RPC FLAG CACHE\\_DATA](#) 40  
[DNS RPC FLAG NODE\\_COMPLETE](#) 40  
[DNS RPC FLAG NODE\\_STICKY](#) 40  
[DNS RPC FLAG OPEN\\_ACL](#) 40  
[DNS RPC FLAG RECORD\\_CREATE\\_PTR](#) 40  
[DNS RPC FLAG RECORD\\_DEFAULT\\_TTL](#) 40  
[DNS RPC FLAG RECORD\\_TTL\\_CHANGE](#) 40  
[DNS RPC FLAG SUPPRESS\\_NOTIFY](#) 40  
[DNS RPC FLAG\\_ZONE\\_DELEGATION](#) 40  
[DNS RPC FLAG\\_ZONE\\_ROOT](#) 40  
[DNS RPC FORWARDERS](#) 99  
[DNS RPC FORWARDERS structure](#) 98  
[DNS RPC FORWARDERS\\_DOTNET structure](#) 99  
[DNS RPC FORWARDERS\\_LONGHORN structure](#) 99  
[DNS RPC FORWARDERS\\_W2K structure](#) 99  
[DNS RPC IP\\_VALIDATE structure](#) 65  
[DNS RPC NAME packet](#) 41  
[DNS RPC NAME\\_AND\\_PARAM structure](#) 32  
[DNS RPC NODE packet](#) 42  
[DNS RPC RECORD structure](#) 57  
[DNS RPC RECORD\\_A packet](#) 43  
[DNS RPC RECORD\\_AAAA packet](#) 51  
[DNS RPC RECORD\\_ATMA packet](#) 52  
[DNS RPC RECORD\\_DATA structure](#) 42  
[DNS RPC RECORD\\_DHCID packet](#) 50  
[DNS RPC RECORD\\_DNSKEY packet](#) 50  
[DNS RPC RECORD\\_DS packet](#) 49  
[DNS RPC RECORD\\_KEY packet](#) 50  
[DNS RPC RECORD\\_MAIL\\_ERROR packet](#) 46  
[DNS RPC RECORD\\_NAME\\_PREFERENCE packet](#) 46  
[DNS RPC RECORD\\_NAPTR packet](#) 53  
[DNS RPC RECORD\\_NODE\\_NAME packet](#) 43  
[DNS RPC RECORD\\_NSEC packet](#) 49  
[DNS RPC RECORD\\_NSEC3 packet](#) 56  
[DNS RPC RECORD\\_NSEC3PARAM packet](#) 56  
[DNS RPC RECORD\\_NULL packet](#) 45  
[DNS RPC RECORD\\_NXT packet](#) 51  
[DNS RPC RECORD\\_RRSIG packet](#) 48  
[DNS RPC RECORD\\_SIG packet](#) 47  
[DNS RPC RECORD\\_SOA packet](#) 44  
[DNS RPC RECORD\\_SRV packet](#) 52  
[DNS RPC RECORD\\_STRING packet](#) 45  
[DNS RPC RECORD\\_TS packet](#) 55  
[DNS RPC RECORD\\_WINS packet](#) 54  
[DNS RPC RECORD\\_WINSR packet](#) 54  
[DNS RPC RECORD\\_WKS packet](#) 45  
[DNS RPC SERVER\\_INFO](#) 75  
[DNS RPC SERVER\\_INFO structure](#) 68  
[DNS RPC SERVER\\_INFO\\_DOTNET structure](#) 72  
[DNS RPC SERVER\\_INFO\\_LONGHORN structure](#) 75  
[DNS RPC SERVER\\_INFO\\_W2K structure](#) 68  
[DNS RPC SIGNING\\_VALIDATION\\_ERROR structure](#) 111  
[DNS RPC SKD structure](#) 101  
[DNS RPC SKD\\_LIST structure](#) 103  
[DNS RPC SKD\\_STATE structure](#) 103  
[DNS RPC SKD\\_STATE\\_EX structure](#) 114  
[DNS RPC TRUST\\_ANCHOR structure](#) 108  
[DNS RPC TRUST\\_ANCHOR\\_LIST structure](#) 110  
[DNS RPC TRUST\\_POINT structure](#) 107  
[DNS RPC TRUST\\_POINT\\_LIST structure](#) 108  
[DNS RPC UNICODE\\_STRING\\_LIST structure](#) 32  
[DNS RPC USE\\_ALL\\_PROTOCOLS](#) 29  
[DNS RPC USE\\_LPC](#) 29  
[DNS RPC USE\\_NAMED\\_PIPE](#) 29  
[DNS RPC USE\\_TCPIP](#) 29  
[DNS RPC UTF8\\_STRING\\_LIST structure](#) 31  
[DNS RPC\\_ZONE](#) 80  
[DNS RPC\\_ZONE structure](#) 79  
[DNS RPC\\_ZONE\\_CHANGE\\_DP structure](#) 120  
[DNS RPC\\_ZONE\\_CREATE\\_INFO](#) 96  
[DNS RPC\\_ZONE\\_CREATE\\_INFO structure](#) 92  
[DNS RPC\\_ZONE\\_CREATE\\_INFO\\_DOTNET structure](#) 95  
[DNS RPC\\_ZONE\\_CREATE\\_INFO\\_LONGHORN structure](#) 96  
[DNS RPC\\_ZONE\\_CREATE\\_INFO\\_W2K structure](#) 93  
[DNS RPC\\_ZONE\\_DATABASE](#) 92  
[DNS RPC\\_ZONE\\_DATABASE structure](#) 91  
[DNS RPC\\_ZONE\\_DATABASE\\_DOTNET structure](#) 92  
[DNS RPC\\_ZONE\\_DATABASE\\_W2K structure](#) 91  
[DNS RPC\\_ZONE\\_DNSSEC\\_SETTINGS structure](#) 111  
[DNS RPC\\_ZONE\\_DOTNET structure](#) 80  
[DNS RPC\\_ZONE\\_EXPORT\\_INFO structure](#) 97  
[DNS RPC\\_ZONE\\_INFO](#) 88  
[DNS RPC\\_ZONE\\_INFO structure](#) 83  
[DNS RPC\\_ZONE\\_INFO\\_DOTNET structure](#) 86  
[DNS RPC\\_ZONE\\_INFO\\_LONGHORN structure](#) 88  
[DNS RPC\\_ZONE\\_INFO\\_W2K structure](#) 83  
[DNS RPC\\_ZONE\\_LIST](#) 82  
[DNS RPC\\_ZONE\\_LIST structure](#) 82  
[DNS RPC\\_ZONE\\_LIST\\_DOTNET structure](#) 82  
[DNS RPC\\_ZONE\\_LIST\\_W2K structure](#) 82  
[DNS RPC\\_ZONE\\_SCOPE\\_CREATE\\_INFO\\_V1 structure](#) 192  
[DNS RPC\\_ZONE\\_SCOPE\\_INFO\\_V1 structure](#) 192  
[DNS RPC\\_ZONE\\_SECONDARIES](#) 91  
[DNS RPC\\_ZONE\\_SECONDARIES structure](#) 89  
[DNS RPC\\_ZONE\\_SECONDARIES\\_DOTNET structure](#) 90  
[DNS RPC\\_ZONE\\_SECONDARIES\\_LONGHORN structure](#) 91  
[DNS RPC\\_ZONE\\_SECONDARIES\\_W2K structure](#) 90  
[DNS RPC\\_ZONE\\_SKD structure](#) 113  
[DNS RPC\\_ZONE\\_STATS\\_V1 structure](#) 191  
[DNS RPC\\_ZONE\\_W2K structure](#) 79  
[DNS\\_SYSTEMTIME packet](#) 128  
[DNS\\_TYPE\\_A](#) 37  
[DNS\\_TYPE\\_AAAA](#) 37  
[DNS\\_TYPE\\_AFSDB](#) 37  
[DNS\\_TYPE\\_ALL](#) 37  
[DNS\\_TYPE\\_ATMA](#) 37  
[DNS\\_TYPE\\_CNAME](#) 37  
[DNS\\_TYPE\\_DHCID](#) 37  
[DNS\\_TYPE\\_DNAME](#) 37  
[DNS\\_TYPE\\_DNSKEY](#) 37  
[DNS\\_TYPE\\_DS](#) 37  
[DNS\\_TYPE\\_HINFO](#) 37  
[DNS\\_TYPE\\_ISDN](#) 37  
[DNS\\_TYPE\\_KEY](#) 37  
[DNS\\_TYPE\\_LOC](#) 37

[DNS\\_TYPE\\_MB](#) 37  
[DNS\\_TYPE\\_MD](#) 37  
[DNS\\_TYPE\\_MF](#) 37  
[DNS\\_TYPE\\_MG](#) 37  
[DNS\\_TYPE\\_MINFO](#) 37  
[DNS\\_TYPE\\_MR](#) 37  
[DNS\\_TYPE\\_MX](#) 37  
[DNS\\_TYPE\\_NAPTR](#) 37  
[DNS\\_TYPE\\_NS](#) 37  
[DNS\\_TYPE\\_NSEC](#) 37  
[DNS\\_TYPE\\_NSEC3](#) 37  
[DNS\\_TYPE\\_NSEC3PARAM](#) 37  
[DNS\\_TYPE\\_NULL](#) 37  
[DNS\\_TYPE\\_NXT](#) 37  
[DNS\\_TYPE\\_PTR](#) 37  
[DNS\\_TYPE\\_RP](#) 37  
[DNS\\_TYPE\\_RRSIG](#) 37  
[DNS\\_TYPE\\_RT](#) 37  
[DNS\\_TYPE\\_SIG](#) 37  
[DNS\\_TYPE\\_SOA](#) 37  
[DNS\\_TYPE\\_SRV](#) 37  
[DNS\\_TYPE\\_TXT](#) 37  
[DNS\\_TYPE\\_WINS](#) 37  
[DNS\\_TYPE\\_WINSR](#) 37  
[DNS\\_TYPE\\_WKS](#) 37  
[DNS\\_TYPE\\_X25](#) 37  
[DNS\\_TYPE\\_ZERO](#) 37  
[DNS\\_ZONE\\_TYPE\\_CACHE](#) 76  
[DNS\\_ZONE\\_TYPE\\_FORWARDER](#) 76  
[DNS\\_ZONE\\_TYPE\\_PRIMARY](#) 76  
[DNS\\_ZONE\\_TYPE\\_SECONDARY](#) 76  
[DNS\\_ZONE\\_TYPE\\_SECONDARY\\_CACHE](#) 76  
[DNS\\_ZONE\\_TYPE\\_STUB](#) 76  
[dnsProperty packet](#) 195  
[dnsRecord packet](#) 197  
[DNSSRV\\_DNSSEC\\_STATS packet](#) 141  
[DNSSRV\\_DS\\_STATS packet](#) 155  
[DNSSRV\\_MASTER\\_STATS packet](#) 141  
[DNSSRV\\_MEMTAG\\_STATS packet](#) 160  
[DNSSRV\\_QIERY2\\_STATS packet](#) 131  
[DNSSRV\\_QUERY\\_STATS packet](#) 130  
[DNSSRV\\_RECURSE\\_STATS packet](#) 133  
[DNSSRV\\_SECONDARY\\_STATS packet](#) 144  
[DNSSRV\\_SKWANSEC\\_STATS packet](#) 153  
[DNSSRV\\_STAT structure](#) 127  
[DNSSRV\\_STAT\\_HEADER structure](#) 127  
[DNSSRV\\_STATID\\_CACHE](#) 125  
[DNSSRV\\_STATID\\_DBASE](#) 125  
[DNSSRV\\_STATID\\_DNSSEC](#) 125  
[DNSSRV\\_STATID\\_DS](#) 125  
[DNSSRV\\_STATID\\_ERRORS](#) 125  
[DNSSRV\\_STATID\\_MASTER](#) 125  
[DNSSRV\\_STATID\\_MEMORY](#) 125  
[DNSSRV\\_STATID\\_NBSTAT](#) 125  
[DNSSRV\\_STATID\\_NONWIRE\\_UPDATE](#) 125  
[DNSSRV\\_STATID\\_PACKET](#) 125  
[DNSSRV\\_STATID\\_PRIVATE](#) 125  
[DNSSRV\\_STATID\\_QUERY](#) 125  
[DNSSRV\\_STATID\\_QUERY2](#) 125  
[DNSSRV\\_STATID\\_RECORD](#) 125  
[DNSSRV\\_STATID\\_RECURSE](#) 125  
[DNSSRV\\_STATID\\_SECONDARY](#) 125  
[DNSSRV\\_STATID\\_SKWANSEC](#) 125  
[DNSSRV\\_STATID\\_TIME](#) 125  
[DNSSRV\\_STATID\\_TIMEOUT](#) 125  
[DNSSRV\\_STATID\\_WINS](#) 125  
[DNSSRV\\_STATID\\_WIRE\\_UPDATE](#) 125  
[DNSSRV\\_TIME\\_STATS packet](#) 128  
[DNSSRV\\_TYPEID\\_ADDRARRAY](#) 24  
[DNSSRV\\_TYPEID\\_ANY](#) 24  
[DNSSRV\\_TYPEID\\_AUTOCONFIGURE](#) 24  
[DNSSRV\\_TYPEID\\_BUFFER](#) 24  
[DNSSRV\\_TYPEID\\_DP\\_ENUM](#) 24  
[DNSSRV\\_TYPEID\\_DP\\_INFO](#) 24  
[DNSSRV\\_TYPEID\\_DP\\_LIST](#) 24  
[DNSSRV\\_TYPEID\\_DWORD](#) 24  
[DNSSRV\\_TYPEID\\_ENLIST\\_DP](#) 24  
[DNSSRV\\_TYPEID\\_ENUM\\_ZONES\\_FILTER](#) 24  
[DNSSRV\\_TYPEID\\_FORWARDERS](#) 24  
[DNSSRV\\_TYPEID\\_FORWARDERS\\_DOTNET](#) 24  
[DNSSRV\\_TYPEID\\_FORWARDERS\\_W2K](#) 24  
[DNSSRV\\_TYPEID\\_IP\\_VALIDATE](#) 24  
[DNSSRV\\_TYPEID\\_IPARRAY](#) 24  
[DNSSRV\\_TYPEID\\_LPSTR](#) 24  
[DNSSRV\\_TYPEID\\_LPWSTR](#) 24  
[DNSSRV\\_TYPEID\\_NAME\\_AND\\_PARAM](#) 24  
[DNSSRV\\_TYPEID\\_NULL](#) 24  
[DNSSRV\\_TYPEID\\_SERVER\\_INFO](#) 24  
[DNSSRV\\_TYPEID\\_SERVER\\_INFO\\_DOTNET](#) 24  
[DNSSRV\\_TYPEID\\_SERVER\\_INFO\\_W2K](#) 24  
[DNSSRV\\_TYPEID\\_SIGNING\\_VALIDATION\\_ERROR](#) 24  
[DNSSRV\\_TYPEID\\_SKD](#) 24  
[DNSSRV\\_TYPEID\\_SKD\\_LIST](#) 24  
[DNSSRV\\_TYPEID\\_SKD\\_STATE](#) 24  
[DNSSRV\\_TYPEID\\_STATS](#) 24  
[DNSSRV\\_TYPEID\\_TRUST\\_ANCHOR\\_LIST](#) 24  
[DNSSRV\\_TYPEID\\_TRUST\\_POINT\\_LIST](#) 24  
[DNSSRV\\_TYPEID\\_UNICODE\\_STRING\\_LIST](#) 24  
[DNSSRV\\_TYPEID\\_UTF8\\_STRING\\_LIST](#) 24  
[DNSSRV\\_TYPEID\\_ZONE](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_CHANGE\\_DP](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_CREATE](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_CREATE\\_DOTNET](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_CREATE\\_W2K](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_DATABASE](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_DATABASE\\_W2K](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_EXPORT](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_INFO](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_INFO\\_DOTNET](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_INFO\\_W2K](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_LIST](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_LIST\\_W2K](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_RENAME](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_SCOPE\\_CREATE](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_SCOPE\\_ENUM](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_SCOPE\\_INFO](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_SECONDARIES](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_SECONDARIES\\_DOTNET](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_SECONDARIES\\_W2K](#) 24  
[DNSSRV\\_TYPEID\\_ZONE\\_SIGNING\\_SETTINGS](#) 24

[DNSSRV TYPEID\\_ZONE\\_STATS](#) 24  
[DNSSRV TYPEID\\_ZONE\\_TYPE\\_RESET](#) 24  
[DNSSRV TYPEID\\_ZONE\\_TYPE\\_RESET\\_DOTNET](#) 24  
[DNSSRV TYPEID\\_ZONE\\_TYPE\\_RESET\\_W2K](#) 24  
[DNSSRV TYPEID\\_ZONE\\_W2K](#) 24  
[DNSSRV UPDATE\\_STATS packet](#) 149  
[DNSSRV VERSION packet](#) 68  
[DNSSRV WINS\\_STATS packet](#) 149  
[DNSSRV\\_ZONE\\_QUERY\\_STATS structure](#) 189  
[DNSSRV\\_ZONE\\_TIME\\_STATS structure](#) 189  
[DNSSRV\\_ZONE\\_TRANSFER\\_STATS structure](#) 190  
[DNSSRV\\_ZONE\\_UPDATE\\_STATS structure](#) 190  
[DSROPERTY\\_ZONE\\_AGING\\_ENABLED\\_TIME](#) 195  
[DSROPERTY\\_ZONE\\_AGING\\_STATE](#) 195  
[DSROPERTY\\_ZONE\\_ALLOW\\_UPDATE](#) 195  
[DSROPERTY\\_ZONE\\_AUTO\\_NS\\_SERVERS](#) 195  
[DSROPERTY\\_ZONE\\_AUTO\\_NS\\_SERVERS\\_DA](#) 195  
[DSROPERTY\\_ZONE\\_DCPROMO\\_CONVERT](#) 195  
[DSROPERTY\\_ZONE\\_DELETED\\_FROM\\_HOSTNAME](#) 195  
[DSROPERTY\\_ZONE\\_MASTER\\_SERVERS](#) 195  
[DSROPERTY\\_ZONE\\_MASTER\\_SERVERS\\_DA](#) 195  
[DSROPERTY\\_ZONE\\_NODE\\_DBLFLAGS](#) 195  
[DSROPERTY\\_ZONE\\_NOREFRESH\\_INTERVAL](#) 195  
[DSROPERTY\\_ZONE\\_REFRESH\\_INTERVAL](#) 195  
[DSROPERTY\\_ZONE\\_SCAVENGING\\_SERVERS](#) 195  
[DSROPERTY\\_ZONE\\_SCAVENGING\\_SERVERS\\_DA](#) 195  
[DSROPERTY\\_ZONE\\_SECURE\\_TIME](#) 195  
[DSROPERTY\\_ZONE\\_TYPE](#) 195

## E

[ECDSA\\_P256\\_Key\\_Pair\\_packet](#) 181  
[ECDSA\\_P384\\_Key\\_Pair\\_packet](#) 183  
[Elements - directory\\_service\\_schema](#) 193  
[Enumerating zone scopes](#) 299  
[Enumerating zones example](#) 296  
[ERROR\\_SUCCESS](#) 61  
[EVENT\\_LOG\\_ERROR\\_TYPE](#) 125  
[EVENT\\_LOG\\_INFORMATION\\_TYPE](#) 125  
[EVENT\\_LOG\\_SUCCESS](#) 125  
[EVENT\\_LOG\\_WARNING\\_TYPE](#) 125

### Events

#### local

[directory server security descriptors reading and caching](#) 290  
[dnsRecord in directory server](#) 290  
[modifying directory server security descriptors](#) 291  
[three-phase authorization test](#) 289

### Examples

[creating and deleting a DNS record in a zone scope](#) 300  
[creating and deleting DNS record](#) 297  
[creating new zone](#) 296  
[enumerating zone scopes](#) 299  
[enumerating zones](#) 296  
[modifying DNS server DWORD property](#) 295  
[querying DNS server DWORD property](#) 295  
[ExportedKeyPair packet](#) 186

## F

[Fields - vendor-extensible](#) 22  
[Full IDL](#) 302

## G

[Glossary](#) 10

## I

[IDL](#) 302  
[Implementer - security considerations](#) 301  
[IMPORT\\_STATUS\\_CHANGED](#) 101  
[IMPORT\\_STATUS\\_NOOP](#) 101  
[IMPORT\\_STATUS\\_SIGNING\\_READY](#) 101  
[IMPORT\\_STATUS\\_UNSIGNING\\_READY](#) 101  
[Index of security parameters](#) 301  
[Informative references](#) 18  
[Initialization](#) 234  
[Introduction](#) 10  
[IP4\\_ARRAY structure](#) 62

## L

### Local events

[directory server security descriptors reading and caching](#) 290  
[dnsRecord in directory server](#) 290  
[modifying directory server security descriptors](#) 291  
[three-phase authorization test](#) 289

## M

[Message processing](#) 238  
Messages  
[data types](#) 24  
[transport](#) 23  
[Modifying DNS server DWORD property example](#) 295  
[msDNS-SigningKeyDescriptors packet](#) 200  
[msDNS-SigningKeys packet](#) 199

## N

[Normative references](#) 15

## O

[Overview \(synopsis\)](#) 18

## P

[Parameter index - security](#) 301  
[PDNS\\_ADDR](#) 62  
[PDNS\\_ADDR\\_ARRAY](#) 65  
[PDNS\\_FLAT\\_RECORD](#) 57  
[PDNS\\_RPC\\_AUTOCONFIGURE](#) 123  
[PDNS\\_RPC\\_BUFFER](#) 31  
[PDNS\\_RPC\\_DP\\_ENUM](#) 118  
[PDNS\\_RPC\\_DP\\_INFO](#) 116

[PDNS RPC DP LIST](#) 119  
[PDNS RPC DP REPLICA](#) 118  
[PDNS RPC ENLIST DP](#) 119  
[PDNS RPC ENUM\\_ZONE\\_SCOPE\\_LIST](#) 191  
[PDNS RPC ENUM\\_ZONES\\_FILTER](#) 98  
[PDNS RPC FORWARDERS](#) 99  
[PDNS RPC FORWARDERS\\_DOTNET](#) 99  
[PDNS RPC FORWARDERS\\_LONGHORN](#) 99  
[PDNS RPC FORWARDERS\\_W2K](#) 99  
[PDNS RPC IP\\_VALIDATE](#) 65  
[PDNS RPC NAME\\_AND\\_PARAM](#) 32  
[PDNS RPC RECORD](#) 57  
[PDNS RPC SERVER\\_INFO](#) 75  
[PDNS RPC SERVER\\_INFO\\_DOTNET](#) 72  
[PDNS RPC SERVER\\_INFO\\_LONGHORN](#) 75  
[PDNS RPC SERVER\\_INFO\\_W2K](#) 68  
[PDNS RPC SIGNING\\_VALIDATION\\_ERROR](#) 111  
[PDNS RPC SKD](#) 101  
[PDNS RPC SKD\\_LIST](#) 103  
[PDNS RPC SKD\\_STATE](#) 103  
[PDNS RPC SKD\\_STATE\\_EX](#) 114  
[PDNS RPC TRUST\\_ANCHOR](#) 108  
[PDNS RPC TRUST\\_ANCHOR\\_LIST](#) 110  
[PDNS RPC TRUST\\_POINT](#) 107  
[PDNS RPC TRUST\\_POINT\\_LIST](#) 108  
[PDNS RPC UNICODE\\_STRING\\_LIST](#) 32  
[PDNS RPC UTF8\\_STRING\\_LIST](#) 31  
[PDNS RPC\\_ZONE](#) 80  
[PDNS RPC\\_ZONE\\_CHANGE\\_DP](#) 120  
[PDNS RPC\\_ZONE\\_CREATE\\_INFO](#) 96  
[PDNS RPC\\_ZONE\\_CREATE\\_INFO\\_DOTNET](#) 95  
[PDNS RPC\\_ZONE\\_CREATE\\_INFO\\_LONGHORN](#) 96  
[PDNS RPC\\_ZONE\\_CREATE\\_INFO\\_W2K](#) 93  
[PDNS RPC\\_ZONE\\_DATABASE](#) 92  
[PDNS RPC\\_ZONE\\_DATABASE\\_DOTNET](#) 92  
[PDNS RPC\\_ZONE\\_DATABASE\\_W2K](#) 91  
[PDNS RPC\\_ZONE\\_DNSSEC\\_SETTINGS](#) 111  
[PDNS RPC\\_ZONE\\_DOTNET](#) 80  
[PDNS RPC\\_ZONE\\_EXPORT\\_INFO](#) 97  
[PDNS RPC\\_ZONE\\_INFO](#) 88  
[PDNS RPC\\_ZONE\\_INFO\\_DOTNET](#) 86  
[PDNS RPC\\_ZONE\\_INFO\\_LONGHORN](#) 88  
[PDNS RPC\\_ZONE\\_INFO\\_W2K](#) 83  
[PDNS RPC\\_ZONE\\_LIST](#) 82  
[PDNS RPC\\_ZONE\\_LIST\\_DOTNET](#) 82  
[PDNS RPC\\_ZONE\\_LIST\\_W2K](#) 82  
[PDNS RPC\\_ZONE\\_SCOPE\\_CREATE\\_INFO\\_V1](#) 192  
[PDNS RPC\\_ZONE\\_SCOPE\\_INFO\\_V1](#) 192  
[PDNS RPC\\_ZONE\\_SECONDARIES](#) 91  
[PDNS RPC\\_ZONE\\_SECONDARIES\\_DOTNET](#) 90  
[PDNS RPC\\_ZONE\\_SECONDARIES\\_LONGHORN](#) 91  
[PDNS RPC\\_ZONE\\_SECONDARIES\\_W2K](#) 90  
[PDNS RPC\\_ZONE\\_SKD](#) 113  
[PDNS RPC\\_ZONE\\_STATS\\_V1](#) 191  
[PDNS RPC\\_ZONE\\_W2K](#) 79  
[PDNSSRV\\_STAT](#) 127  
[PDNSSRV\\_STAT\\_HEADER](#) 127  
[PDNSSRV\\_STATS](#) 127  
[PDNSSRV\\_ZONE\\_QUERY\\_STATS](#) 189  
[PDNSSRV\\_ZONE\\_TIME\\_STATS](#) 189  
[PDNSSRV\\_ZONE\\_TRANSFER\\_STATS](#) 190

[PDNSSRV\\_ZONE\\_UPDATE\\_STATS](#) 190  
[PIP4\\_ARRAY](#) 62  
[Preconditions](#) 21  
[Prerequisites](#) 21  
Processing rules  
    [constructing exported key](#) 291  
    [importing exported key](#) 293  
[Product behavior](#) 337  
[Protection\\_Key\\_Identifier\\_packet](#) 184

## Q

[Querying DNS server DWORD property example](#)  
295

## R

[R\\_DnssrvComplexOperation\\_method](#) 271  
[R\\_DnssrvComplexOperation2\\_method](#) 284  
[R\\_DnssrvEnumRecords\\_method](#) 278  
[R\\_DnssrvEnumRecords2\\_method](#) 285  
[R\\_DnssrvEnumRecords3\\_method](#) 287  
[R\\_DnssrvOperation\\_method](#) 240  
[R\\_DnssrvOperation2\\_method](#) 283  
[R\\_DnssrvOperation3\\_method](#) 287  
[R\\_DnssrvQuery\\_method](#) 268  
[R\\_DnssrvQuery2\\_method](#) 284  
[R\\_DnssrvQuery3\\_method](#) 288  
[R\\_DnssrvUpdateRecord\\_method](#) 281  
[R\\_DnssrvUpdateRecord2\\_method](#) 286  
[R\\_DnssrvUpdateRecord3\\_method](#) 286

### References

[informative](#) 18  
[normative](#) 15  
[Relationship to other protocols](#) 19  
[RSA Key Pair packet](#) 180

## S

[Schema elements - directory service](#) 193  
Security  
    [implementer considerations](#) 301  
    [parameter index](#) 301  
[Sequencing rules](#) 238  
[Server - overview](#) 207  
[SIGN\\_SCOPE\\_ADD\\_ONLY](#) 100  
[SIGN\\_SCOPE\\_ALL\\_RECORDS](#) 100  
[SIGN\\_SCOPE\\_DEFAULT](#) 100  
[SIGN\\_SCOPE\\_DNSKEY\\_ONLY](#) 100  
[SIGN\\_SCOPE\\_DO\\_NOT\\_PUBLISH](#) 100  
[SIGN\\_SCOPE\\_REVOKED](#) 100  
[Standards assignments](#) 22

## T

[Timer events](#) 289  
[Timers](#) 234  
[Tracking changes](#) 368  
[Transport](#) 23  
[TRUSTANCHOR\\_STATE\\_ADDPEND](#) 30  
[TRUSTANCHOR\\_STATE\\_DSINVALID](#) 30  
[TRUSTANCHOR\\_STATE\\_DSPENDING](#) 30

[TRUSTANCHOR STATE MISSING](#) 30  
[TRUSTANCHOR STATE REVOKED](#) 30  
[TRUSTANCHOR STATE VALID](#) 30  
[TRUSTPOINT STATE ACTIVE](#) 29  
[TRUSTPOINT STATE DELETE\\_PENDING](#) 29  
[TRUSTPOINT STATE DSPENDING](#) 29  
[TRUSTPOINT STATE INITIALIZED](#) 29

## **V**

[Vendor-extensible fields](#) 22  
[Versioning](#) 22

## **Z**

[ZONE NOTIFY ALL SECONDARIES](#) 77  
[ZONE NOTIFY LIST ONLY](#) 77  
[ZONE NOTIFY OFF](#) 77  
[ZONE REQUEST AUTO](#) 77  
[ZONE REQUEST CACHE](#) 77  
[ZONE REQUEST CUSTOM DP](#) 77  
[ZONE REQUEST DOMAIN DP](#) 77  
[ZONE REQUEST DS](#) 77  
[ZONE REQUEST FOREST DP](#) 77  
[ZONE REQUEST FORWARD](#) 77  
[ZONE REQUEST FORWARDER](#) 77  
[ZONE REQUEST LEGACY DP](#) 77  
[ZONE REQUEST NON\\_DS](#) 77  
[ZONE REQUEST PRIMARY](#) 77  
[ZONE REQUEST REVERSE](#) 77  
[ZONE REQUEST SECONDARY](#) 77  
[ZONE REQUEST STUB](#) 77  
[ZONE SECSECURE LIST ONLY](#) 76  
[ZONE SECSECURE NO SECURITY](#) 76  
[ZONE SECSECURE NO\\_XFER](#) 76  
[ZONE SECSECURE NS\\_ONLY](#) 76  
[ZONE UPDATE OFF](#) 100  
[ZONE UPDATE SECURE](#) 100  
[ZONE UPDATE UNSECURE](#) 100