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

---

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

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

Revision Summary

Date	Revision History	Revision Class	Comments
04/03/2007	0.01		MCPP Milestone Longhorn Initial Availability
07/03/2007	1.0	Major	MLonghorn+90
07/20/2007	1.0.1	Editorial	Revised and edited the technical content.
08/10/2007	1.0.2	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
09/28/2007	1.1	Minor	Updated the technical content.
10/23/2007	1.2	Minor	Updated the technical content.
11/30/2007	1.2.1	Editorial	Revised and edited the technical content.
01/25/2008	2.0	Major	Added two sections.

Table of Contents

1 Introduction

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

The protocol consists of one RPC interface named drsuapi. This protocol was originally implemented in Windows 2000 Server and is implemented in all subsequent server releases. It is not implemented in Windows 3.1, Windows NT 3.51, Windows NT 4.0, Windows XP, or Windows Vista.

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

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

The special typographical conventions used in this document are described in section 3.2.

State is included in the state model for this specification only as necessitated by the requirement that a licensee's implementation of Windows server protocols must be capable of receiving messages and responding in the same manner as a Windows server. Behavior is specified in terms of request message received, processing based on current state, resulting state transformation, and response message sent. Unless otherwise specified, all behaviors are required elements of the protocol. Any specified behavior not explicitly qualified with MAY or SHOULD is to be treated as if it were specified as a MUST behavior.

1.1 Glossary

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

1.1.1 Pervasive Concepts

This specification uses [KNUTH1] section 2.3.4.2 as a reference for the graph-related terms **oriented tree**, **root**, **vertex**, **arc**, **initial vertex**, and **final vertex**.

Replica: A variable containing a set of objects.

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

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

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

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

As the term "replica" suggests, two replicas often contain "the same objects". In this usage, objects in two replicas are considered "the same" if they have the same value of the identifying attribute and if there is a process in place (that is, replication) to converge the values of the remaining attributes.

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

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

Parent Object: See **Object**.

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

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

1.1.2 Glossary Entries

The following terms are defined in [MS-GLOS]:

Ancestor Object
AttributeStamp
Authentication
Authentication Level
Back Link Value
Binary Large Object (BLOB)
Constructed Attribute
Container
Control Access Right (CAR)
Digest
Directory
Discretionary Access Control List (DACL)

Domain Controller (DC)
Dynamic Endpoint
Expunge
Forest
Forward Link Value
FSMO Role Owner
Full NC Replica
Garbage Collection
Global Catalog (GC)
Global Catalog Server (GC Server)
Interface Definition Language (IDL)
LDAP Connection
Lightweight Directory Access Protocol (LDAP)
Link Attribute
Link Value
LinkValueStamp
Local Domain Controller (Local DC)
Microsoft Interface Definition Language (MIDL)
Network Data Representation (NDR)
Non-Replicated Attribute
NULL GUID
Opnum
Originating Update
Partial Attribute Set (PAS)
PDC Role Owner
Prefix Table
Remote Procedure Call (RPC)
Replication Latency
RPC Transport
Schema
Security Principal
Security Provider

The following terms are specific to this document:

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

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

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

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

Active Directory Domain Services (AD/DS): AD/DS is an operating system **directory** service implemented by a **domain controller (DC)**. The directory service provides a data store for objects that is distributed across multiple DCs. The DCs interoperate as peers to ensure that a local change to an object replicates correctly across DCs. AD/DS first became available as part of Windows 2000 and is available as part of Windows 2000 Server and Windows Server 2003 products; in these products it is called "Active Directory". It is also available as part of Windows Server 2008. AD/DS is not present in Windows 3.1,

Windows NT 3.51, Windows NT 4.0, or Windows XP. For more information, see [MS-SECO] section 2.2.2 and [MS-ADTS].

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

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

Attribute: (Note: This definition is a specialization of the **Attribute** entry in Pervasive Concepts.) An identifier for a single- or multivalued data element associated with an LDAP directory object. An object consists of its **attributes** and their values. For example, cn (common name), street (street address), and mail (e-mail addresses) can all be **attributes** of a user object. An **attribute's schema**, including the syntax of its values, is defined in an attributeSchema object.

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

attributeID**:** An **OID**-valued identifying **attribute** of each attributeSchema object in the **schema NC**.

Back Link Attribute: A computed **attribute** whose values include object references (for example, an **attribute** of syntax Object(DS-DN)). The values are derived from the values of a related **attribute**, a **forward link attribute**, on other objects. If *f* is the **forward link attribute**, one back **link value** exists on object *o* for each object *r* that contains a value of *o* for **attribute** *f*. The relationship between **forward link attributes** and **back link attributes** is expressed using the linkID **attribute** on the attributeSchema objects representing the two **attributes**. The forward link's linkID is an even number, the back link's linkID is the forward link's linkID plus one. For more information, see [MS-ADTS] section 3.1.1.1.6.

Binary OID: An **OID** in a BER-encoded binary format, as specified in [ITUX690] section 8.19.

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

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

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

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

Class: See **Object Class**.

Computer Object, computer **Object:** An **object of class** computer. A computer object is a **security principal object**; the **principal** is the operating system running on the computer.

The shared secret allows the operating system running on the computer to authenticate itself independently of any user running on the system. See Security Principal.

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

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

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

crossRef **Object:** An **object of class** crossRef. Each crossRef object is a **child** of the **partitions container** in the **config NC**. A crossRef describes the properties of an **NC**, such as its DNS name, operational settings, and so on.

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

Default Naming Context Replica (Default NC Replica): Part of the state of a DC. A DC's **default NC replica** is the full domain NC replica hosted by the DC.

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

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

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

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

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

domainDNS**:** A specific object class. The root of a **domain NC** or an **application NC** is an **object of class** domainDNS. The **DN** of such an object takes the form

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

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

$n_1.n_2. \ldots .n_k$

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

DSA GUID: The objectGUID of a **DSA object**.

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

DSName: A **DSName** is a tuple that contains between one and three identifiers for an object. The possible identifiers are the object's **GUID** (**attribute** objectGUID), **SID** (**attribute** objectSid), and **DN** (**attribute** distinguishedName). Given a **DSName**, an object can be identified within a set of **NC replicas** according to the matching rules defined in section 5.33.

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

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

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

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

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

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

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

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

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

FSMO Role Abandon: A request to a DC *D*. The effect is for *D* to request the current owner of a specified **FSMO role** to transfer the role to *D* (see **FSMO Role Transfer**). Abandon is typically initiated by the current role owner in anticipation of being unable to host the role; for example, because the DC is being decommissioned. The server-to-server methods required to implement any aspect of a **FSMO role**, or to transfer a **FSMO role** from one DC to another DC, are not included in this document, and are not required for interoperation with Windows client operating systems.

FSMO Role Object: The object in the **domain** or forest that represents a specific **FSMO role**. This object is a member of the **FSMO role** and contains the fSMORoleOwner **attribute**.

FSMO Role Transfer: A request to a DC *D*. If *D* is the current owner of the specified **FSMO role**, the effect is to transfer that role to the client; if *D* is not the current owner of the role, the effect is to **update** the client's role objects from *D*'s replica, so that the client can try the request again on another DC. The server-to-server methods required to implement any aspect of a **FSMO role**, or to transfer a **FSMO role** from one DC to another DC, are not included in this document, and are not required for interoperation with Windows client operating systems.

Global Group: An **Active Directory group** that allows user objects from its own **domain** and **global groups** from its own **domain** as members. **Universal groups** can contain **global groups**. A group object *G* is a **global group** if GROUP_TYPE_ACCOUNT_GROUP is present in *G*'s groupType **attribute**. A **global group** contributes to the creation of **security contexts** if GROUP_TYPE_SECURITY_ENABLED is present in *G*'s groupType **attribute**; in this case the **group** is valid for inclusion within **access control lists (ACLs)** anywhere in the forest.

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

governsID**:** An **OID**-valued identifying **attribute** of each classSchema object in the **schema NC**.

Group: See **Group Object**.

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

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

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

Knowledge Consistency Checker (KCC): An internal Windows component of **Active Directory replication** used to create spanning trees for server-to-server **replication** and to translate those trees into settings of variables that implement the **replication** topology. The implementation details of this component are not included in this document, and are not required for interoperation with Windows client operating systems.

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

Mixed Mode: A state of an **Active Directory domain** that supports DCs running Windows NT Server 4.0. **Mixed mode** does not allow organizations to take advantage of new **Active**

Directory features such as **universal groups**, nested group membership, and inter-domain group membership. See **Native Mode**.

Naming Context (NC): An **NC** is a **DSName**, containing at least a **DN** and a **GUID**, used in forming names for a tree of objects. The **DN** of the **DSName** is the distinguishedName **attribute** of the tree root. The **GUID** of the **DSName** is the objectGUID **attribute** of the tree root. The **SID** of the **DSName**, if present, is the objectSid **attribute** of the tree root; the **SID** is present if and only if the **NC** is a **domain NC**. **Active Directory** allows **NCs** to be organized into a tree structure.

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

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

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

Object Class Name: The lDAPDisplayName of the classSchema object of a class. This document consistently uses **object class names** to denote **classes**; for example, user and group are both **object class names**. The correspondence between LDAP display names and numeric **OIDs** in the **Active Directory** schema is specified in the following appendices of [MS-ADTS]: [MS-ADSC], [MS-ADA1], [MS-ADA2], and [MS-ADA3].

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

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

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

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

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

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

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

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

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

PDC Emulator: A DC that is designated to track changes made to the accounts of all computers in a **domain**. The PDC emulator is the only computer to receive these changes directly and is specialized so as to ensure consistency and to eliminate the potential for conflicting **entries** in the **Active Directory** database. A **domain** has only one **PDC emulator**.

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

Principal: See Security Principal.

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

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

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

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

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

Replicated Update: An **update** performed to an **NC replica** by the implemented **replication** system to propagate the effect of an originating update at another **NC replica**. The **stamp** assigned during the originating update to an **attribute** or a link value is preserved by **replication**. Neither this stamp nor any other specific aspects of a replicated update are required for interoperation with Windows client operating systems.

Replication: The process of propagating the effects of all originating writes, to any replica of an **NC**, to all replicas of the **NC**. If originating writes cease and **replication** continues, all replicas converge to a common application-visible state. The description and details of the methods used for this server-to-server implementation are not included in this document, and are not required for interoperation with Windows client operating systems.

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

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

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

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

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

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

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

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

Security Identifier (SID): An identifier for a **security principal object**. The **SID** is composed of an account authority portion (typically corresponding to a **domain**) and an integer representing an identity relative to the account authority, termed the **RID**. The **SID** format is specified in [MS-DTYP] section **2.4.2**; a string representation of a **SID** is specified in [MS-SECO] section 2.1.2. See **Relative Identifier (RID)**.

Security Principal Object: An object that corresponds to a security principal. A **security principal object** contains an identifier, used by the system and applications to name the **principal**, and a secret shared only by the **principal**. In **Active Directory**, a **security principal object** is identified by the objectSid **attribute**. In **Active Directory**, the domainDNS, user, computer, and group classes are examples of **security principal object classes** (though not every group object is a **security principal object**). See domainDNS, objectSid, **Computer Object**, **Group Object**, **User Object**.

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

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

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

Site: A collection of one or more well-connected (reliable and fast) TCP/IP subnets. By defining **sites** (represented by site objects) an administrator can optimize both **Active Directory** access and **Active Directory replication** with respect to the physical network. When a user logs in, an **Active Directory** client finds a DC that is in the same **site** as the client, or near

the same **site** if there is no DC in the **site**. See **Knowledge Consistency Checker (KCC)**, **Site Object**.

Site Object: An **object of class** site, representing a **site**.

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

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

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

Tombstone: A deleted object in the directory that remains in storage until a configured amount of time (the **tombstone lifetime**) has passed, after which the object is permanently deleted. The delay of the permanent deletion exists to support the server-to-server replication implementation. By keeping the **tombstone** in existence for the **tombstone lifetime**, the deleted state of the object has time to replicate to all DCs. An implementation of tombstones is not required for interoperation with Windows client operating systems.

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

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

Universal Group: An **Active Directory** group that allows user objects, **global groups**, and **universal groups** from anywhere in the forest as members. A group object *G* is a **universal group** if GROUP_TYPE_UNIVERSAL_GROUP is present in *G*'s groupType **attribute**. A **universal group** contributes to the creation of **security contexts** if GROUP_TYPE_SECURITY_ENABLED is present in *G*'s groupType **attribute**; in this case, the group is valid for inclusion within **ACLs** anywhere in the forest.

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

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

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

Update Sequence Number (USN): A monotonically increasing sequence number used in assigning a **stamp** to an originating update. For more information, see [MS-ADTS] section 3.1.1.1.9. This structure is used in the server-to-server replication implementation, and is not required for interoperation with Windows client operating systems. See **Invocation ID**.

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

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

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

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

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

1.2 References

1.2.1 Normative References

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

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

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

[MS-ADA1] Microsoft Corporation, "Active Directory Schema Attributes A-L", June 2007.

[MS-ADA2] Microsoft Corporation, "Active Directory Schema Attributes M", June 2007.

[MS-ADA3] Microsoft Corporation, "Active Directory Schema Attributes N-Z", June 2007.

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

[MS-ADSC] Microsoft Corporation, "Active Directory Schema Classes", June 2007.

[MS-ADTS] Microsoft Corporation, "Active Directory Technical Specification", June 2007.

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

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

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

[MS-LSAD] Microsoft Corporation, "Local Security Authority (Domain Policy) Remote Protocol Specification", June 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[MS-SECO] Microsoft Corporation, "Windows Security Overview", January 2007.

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

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

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

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

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

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

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

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

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

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

1.2.2 Informative References

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

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

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

1.3 Protocol Overview (Synopsis)

This document specifies the Directory Replication Service Remote Protocol, an RPC protocol for replication between domain controllers and management of Active Directory. Only the methods that are used to communicate between a Windows client and a Windows server are defined in this document. Some of the methods that are defined in this document are used to diagnose, monitor, or manage Windows directory replication services, and are not required for interoperation with Windows clients. The protocol consists of one RPC interface, named drsuapi. The name of each drsuapi method begins with "IDL_DRS".

1.3.1 Methods Categorized by Function

The Directory Replication Service (DRS) Remote Protocol contains 28 methods, eight of which are not documented here and are not required for interoperation with Windows client operating systems. The documented methods are diverse in function and fall into the following categories:

- Context handle methods: IDL_DRSBind, IDL_DRSUnbind. These methods create and destroy RPC context handles that maintain volatile state used by drsuapi methods.
- Replication methods: IDL_DRSReplicaSync, IDL_DRSReplicaVerifyObjects, IDL_DRSGetReplInfo. The IDL_DRSReplicaSync and IDL_DRSReplicaVerifyObjects methods cause the server to initiate server-to-server replication. The IDL_DRSGetReplInfo method is used to gather information about the details of the server-to-server replication implementation. These methods are exposed only to monitor, diagnose, and manage the replication implementation, and are not required for interoperation with Windows clients.
- Lookup: IDL_DRSCrackNames. This method performs directory lookups.
- DC Locator support methods: IDL_DRSDomainControllerInfo, IDL_DRSQuerySitesByCost. These methods retrieve information about the domain controllers in a domain or forest and information about the cost of connections between different sites.
- Knowledge Consistency Checker support methods: IDL_DRSUpdateRefs, IDL_DRSReplicaAdd, IDL_DRSReplicaDel, IDL_DRSReplicaModify, IDL_DRSExecuteKCC. These methods can be used to create a replication topology, and by administrator tools to manage that replication topology. These methods are exposed only to monitor, diagnose, and manage the replication topology implementation, and are not required for interoperation with Windows clients.
- Administrator-tool support methods: IDL_DRSAddSidHistory, IDL_DRSRemoveDsServer, IDL_DRSRemoveDsDomain, IDL_DRSWriteSPN. These methods are used by administrator tools to perform various specialized functions.

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

1.3.2 Sequencing Issues

The sequencing issues in this RPC protocol are as follows:

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

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

  Because the state associated with a **DRS_HANDLE** is immutable so long as the **DRS_HANDLE** remains valid, there are no special considerations involved in making concurrent method calls using the same **DRS_HANDLE**; the client is free to make concurrent method calls.
- IDL_DRSInitDemotion is called before the other demotion methods: IDL_DRSReplicaDemotion and IDL_DRSFinishDemotion.

- Otherwise, all method requests are independent, apart from their dependencies on the state of the directory. The potential dependencies are varied and understanding them requires understanding the state model specified in [MS-ADTS] section 3.1.1. Here are some examples:

  - Successfully processing an IDL_DRSReplicaAdd request creates a repsFrom value on a server. When a server is in this state (that is, when it holds the repsFrom value), it has the information needed to make a server-to-server replication request on the DC that is specified in IDL_DRSReplicaAdd.

  - Successfully processing an IDL_DRSUpdateRefs request creates a repsTo value on a server. When a server is in this state (that is, when it holds the repsTo value), it has the information needed to make an IDL_DRSReplicaSync request on the DC that is specified in IDL_DRSUpdateRefs.

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

- One method, IDL_DRSGetReplInfo, has a parameter of both input and output, dwEnumerationContext. This parameter is defined for the following:

  - dwInVersion=2, and

  - InfoType=DS_REPL_INFO_METADATA_FOR_ATTR_VALUE, or DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE, or DS_REPL_INFO_CURSORS_2_FOR_NC, or DS_REPL_INFO_CURSORS_3_FOR_NC.

  For the first call to this method for a specific InfoType, the client sets dwEnumerationContext in pmsgIn to zero. The server returns an implementation-specific value for dwEnumerationContext in pmsgOut. On subsequent calls to this method with the same InfoType, the client sets the input dwEnumerationContext in pmsgIn to the last value of that field returned from the server. The purpose of this field is to allow the client to gather all the requested information, but in more than one server call. The final call is identified when the method returns ERROR_NO_MORE_ITEMS. See the server implementation section for IDL_DRSGetReplInfo (4.1.7.3) for exact details.

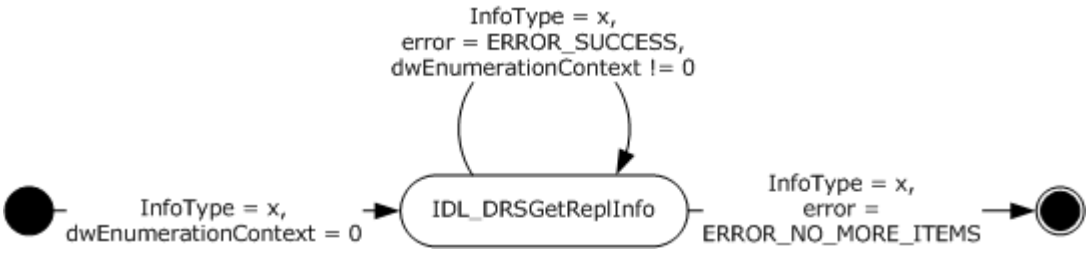


Figure 1: Using dwEnumerationContext

1.3.3 Most Frequently Used Types

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

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

1.4 Relationship to Other Protocols

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

- RootDSE constructed attributes: dnsHostName, dsServiceName, isGlobalCatalogReady, serverName (these expose some functionality of IDL_DRSDomainControllerInfo).
- RootDSE modify operations: becomeDomainMaster, becomeInfrastructureMaster, becomePdc, becomeRidMaster, becomeSchemaMaster, replicateSingleObject, removeLingeringObject. The last two operations expose some functionality of the server-to-server replication implementation. These operations are not required for interoperation with Windows clients, and are exposed only for diagnostics, monitoring, and management of the server-to-server replication implementation.
- Object constructed attributes: canonicalName (this exposes some functionality of IDL_DRSCrackNames), tokenGroups, tokenGroupsNoGCAcceptable, tokenGroupsGlobalAndUniversal (these expose some functionality of the implementation of group expansion).
- Controls: LDAP_SERVER_DIRSYNC_OID

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

- The function of IDL_DRSWriteSPN can be performed as an LDAP Modify of the servicePrincipalName attribute.

1.5 Prerequisites/Preconditions

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

Security configuration for usage of RPC is described further in section 2.2.

1.6 Applicability Statement

This protocol is appropriate for the management of objects in a directory, and for overall management of the directory service.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports**: RPC can be implemented on top of TCP and other protocol sequences as described in section 2.1.
- **Protocol Versions:** The drsuapi protocol interface described in this document has a single version number of 4.0.
- **Security and Authentication Methods**: See [MS-RPCE] section 1.7.
- **Capability Negotiation**: This protocol does explicit capability negotiation as described in IDL_DRSBind (section 4.1.2) behavior.

1.8 Vendor-Extensible Fields

This protocol does not define any vendor-extensible fields.

1.9 Standards Assignments

Parameter	Value	Reference
RPC Interface **UUID** for drsuapi methods	e3514235-4b06-11d1-ab04-00c04fc2dcd2	Section 4.1.1 - section 4.1.20

2 Message Transport

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

2.1 RPC Transport

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

This protocol uses RPC **dynamic endpoints**, as described in [C706] part 4.

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

2.2 Protocol Security

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

2.2.1 General Background

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

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

2.2.2 Service Principal Names for Domain Controllers

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

Kerberos verifies the services provided by a server by reading the servicePrincipalName attribute of the server's computer object. The servicePrincipalName attribute contains a set of Unicode strings; each string is an SPN. If the client produces an SPN that is not present on the computer object of the server it has requested a connection to, the mutual authentication fails and so does the connection attempt.

Each DC maintains the values of the servicePrincipalName attribute on its own computer object.

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

2.2.3 Client-to-DC Operations

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

2.2.3.1 Security Provider

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

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

2.2.3.2 SPN for a Target DC in AD/DS

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

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

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

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

  The SPN that a client constructs depends on the information that the client has available. For example, some clients have only a NetBIOS name for a DC, while others have only a DNS name for a DC.
- A client wants to connect to a DC in a particular domain. The client constructs either of the following two SPNs:
  - "ldap/<DNS hostname>/<NetBIOS domain name>"
  - "ldap/<DNS hostname>/<DNS domain name>"

  The SPN that a client constructs depends on the information that the client has available. For example, some clients have only a NetBIOS name for a domain, while others have only a DNS name for a domain.
- A client wants to connect to a GC server in the forest:
  - "GC/<DNS hostname >/<DNS forest name>"

In the preceding SPN descriptions:

- "ldap" and "GC" are literal strings representing service classes,
- The forward slash ('/') is the literal separator between parts of the SPN,

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

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

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

To allow mutual authentication to occur in client-to-DC protocol operations, an AD/DS DC MUST store these six forms of SPN as values of the servicePrincipalName attribute of the DC's computer object.

2.2.3.3 SPN for a Target DC in AD/LDS

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

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

In the preceding SPN descriptions:

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

If an AD/LDS DC runs on a computer joined to an external AD domain, and NTDSDSA_OPT_DISABLE_SPN_REGISTRATION is not present in the options attribute of its nTDSDSA object in AD/LDS (see [MS-ADTS] section 7.1.1.2.2.1.2.1.1), the AD/LDS DC MUST store these two forms of SPN as values of the servicePrincipalName attribute of the DC's **service account object** in

the external AD domain. This action allows mutual authentication to occur in client-to-AD/LDS DC protocol operations.

2.2.4 DC-to-DC Operations

The security and mutual authentication requirements for those DRS protocol operations that involve DC-to-DC interactions are implementation-specific.<2>

3 Background to Behavior Specifications

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

3.1 Document Organization

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

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

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

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

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

3.2 Typographical Conventions

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

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

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

3.3 State Model

3.3.1 Preliminaries

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

3.3.2 Transactions

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

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

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

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

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

When the specification includes a client preparing a method request or processing a method response, the pattern is similar. There is no use of distributed transactions.

3.3.3 Concrete and Abstract Types

This protocol specification involves both concrete and **abstract** types.

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

- Types in the IDL definition of the drsuapi RPC interface that determine the format of network requests and responses.
- Types that are hand marshaled onto the network, such as types that are sent in drsuapi requests and responses as octet strings whose actual structure is hidden from the IDL compiler. The hand

marshaling and corresponding hand unmarshaling are performed by the implementation of Active Directory (AD) and by clients of the drsuapi RPC interface.

- Types that are hand marshaled into directory attributes, such as types that are stored in the directory as octet strings. The hand marshaling and corresponding hand unmarshaling are performed by the implementation of AD and by clients of the AD LDAP interface [MS-ADTS] section 3.1.1.3.

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

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

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

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

3.4 Pseudocode Language

3.4.1 Naming Conventions

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

- Examples: REPS_FROM, DRS_MSG_UPDREFS

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

- Examples: repsFrom, nTDSDSA

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

- Examples: RepsFrom, ValidateSiteRDN

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

- Examples: dc, vSet

3.4.2 Language Constructs for Concrete Types

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

```
reqV1: DRS_MSG_REPADD_V1
reqV2: DRS_MSG_REPADD_V2

reqV2 := reqV1
```

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

```
reqV1: DRS_MSG_REPADD_V1
reqV2: DRS_MSG_REPADD_V2

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

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

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

3.4.3 Language Constructs for Abstract Types

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

The notation [first .. last] stands for the *subrange* first, first+1, ... , last. The type *byte* is the subrange [0.. 255].

A *sequence* is an indexed collection of variables, called the *elements* of the sequence. The elements all have the same type. The *index type* of a sequence is a zero-based subrange. *S*[*i*] denotes the element of the sequence *S* that corresponds to the value *i* of the index type. The number of elements in a sequence *S* is denoted *S*.length. Therefore, the index type of a sequence *S* is [0 .. *S*.length-1].

A sequence type can be *open* (index type not specified) or *closed* (index type specified):

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

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

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

Therefore:

- s := []

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The language includes *sets*. If *S* is a set, number(*S*) is the cardinality of the set *S*.

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

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

Therefore:

- *S* := {}

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

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

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

is false.

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

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

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

- stringSet := stringArrayPtr^

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

3.4.4 Common Language Constructs

Syntax of standard control structures:

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

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

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

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

while boolean-expr
  stmts
endwhile

return expr
```

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

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

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

var: type     : declare a variable with a type
```

```
var := expr    : assignment
expr^          : pointer dereferencing
expr.id        : field selection
```

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

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

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

```
a - b + c
```

means:

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

Parentheses can be used to override the precedence rules.

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

3.4.5 Access to Objects and Their Attributes

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

```
o!attr
```

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

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

- If the attr attribute is not present on o, the value of the expression is the distinguished value null.

- If the attr attribute is present and declared multi-valued, the value of the expression is a *set* that contains all the values of attr. If only one value is present, the value of the expression is a set that contains one element, the value.
- If the attr attribute is present and declared single-valued, the value of the expression is the value of attr.

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

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

... removes the attr attribute from object o.

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

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

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

The value null can be used in the following ways:

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

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

```
i: integer
s: set

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

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

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

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

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

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

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

Scopes take the form:

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

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

```
var from o!attr
```

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

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

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

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

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

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

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

Here is a query example:

```
rt := select one v from nc!repsTo where
    v.naDsa = pReq^.V1.pszDsaDest or
    v.uuidDsa = pReq^.V1.uuidDsaObjDest
```

```
if rt = null then
  /* no matching values */
endif
```

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

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

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

3.4.6 Asynchronous Processing

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

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

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

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

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

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

3.5 Conventions for Protocol Examples

3.5.1 Common Configuration

This section specifies the test setup that is used for most of the examples presented in section 4. The behavior of certain methods can be highlighted only by starting from a different state. The example section for such a method specifies the difference between the initial state used for that example and the state given here.

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

Forest: CONTOSO.COM

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

Domains:

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

Sites:

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

DCs:

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

Domain-joined computer:

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

Users added:

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

Groups added:

- Domain: CONTOSO.COM
  - CN =GroupA, CN =Users, DC =CONTOSO, DC =COM,
    - objectSid: S-1-5-21-254470460-2440132622-709970653-1114
    - member: null
    - groupType: {GROUP_TYPE_RESOURCE_GROUP, GROUP_TYPE_SECURITY_ENABLED}
  - CN = Administrators, CN =Builtin, DC =CONTOSO, DC =COM

- objectSid: S-1-5-32-544
- member: Domain Admins, Enterprise Admins, Local Administrator of DC1
- groupType: {GROUP_TYPE_BUILTIN_LOCAL_GROUP, GROUP_TYPE_RESOURCE_GROUP, GROUP_TYPE_SECURITY_ENABLED}

3.5.2 Data Display Conventions

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

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

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

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

For example, the string:

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

in the LDP transcript is changed to:

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

assuming that the search requested that only the repsTo attribute be returned.

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

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

3.6 Server and Client Initialization

The server MUST start the RPC service to listen on the incoming RPC. For server configurations, see section 2.1.

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

3.6.1 AD/LDS Specifics

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

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

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

4 RPC Methods and Their Behavior

The methods for the drsuapi RPC interface are described in section 4.1.

4.1 drsuapi RPC Interface

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

Methods in RPC Opnum Order

Method	Description
IDL_DRSBind	Creates a context handle necessary to call any other method in this interface. Opnum: 0
IDL_DRSUnbind	Destroys a context handle previously created by the IDL_DRSBind method. Opnum: 1
IDL_DRSReplicaSync	Triggers replication from another DC. Not required for interoperation with Windows clients. Opnum: 2
IDL_DRSUpdateRefs	Adds or deletes a value from the repsTo attribute of a specified NC replica. Not required for interoperation with Windows clients. Opnum: 4
IDL_DRSReplicaAdd	Adds a replication source reference for the specified NC. Not required for interoperation with Windows clients. Opnum: 5
IDL_DRSReplicaDel	Deletes a replication source reference for the specified NC. Not required for interoperation with Windows clients. Opnum: 6
IDL_DRSReplicaModify	Updates the value for repsFrom for the NC replica. Not required for interoperation with Windows clients. Opnum: 7
IDL_DRSCrackNames	Looks up each of a set of objects in the directory and returns it to the caller in the requested format. Opnum: 12
IDL_DRSWriteSPN	Updates the set of service principal names (SPNs) on an object. Opnum: 13
IDL_DRSRemoveDsServer	Removes the representation of a DC from the directory. Not required for interoperation with Windows clients. Opnum: 14
IDL_DRSRemoveDsDomain	Removes the representation of a domain from the directory. Not required for interoperation with Windows clients. Opnum: 15

Method	Description
IDL_DRSDomainControllerInfo	Retrieves information about DCs in a given domain. Opnum: 16
IDL_DRSExecuteKCC	Validates the replication interconnections of DCs and updates them if necessary. Not required for interoperation with Windows clients. Opnum: 18
IDL_DRSGetReplInfo	Retrieves the implemented replication state of the server. Not required for interoperation with Windows clients. Opnum: 19
IDL_DRSAddSidHistory	Adds one or more SIDs to the sIDHistory attribute of a given object. Opnum: 20
IDL_DRSReplicaVerifyObjects	Verifies the existence of objects in an NC replica. Not required for interoperation with Windows clients. Opnum: 22
IDL_DRSQuerySitesByCost	Determines the communication cost from a "from" site to one or more "to" sites. Opnum: 24
IDL_DRSInitDemotion	Performs the first phase of the removal of a DC from an AD/LDS forest. Not required for interoperation with Windows clients. Opnum: 25
IDL_DRSReplicaDemotion	Replicates off all changes to the specified NC and moves any FSMOs held to another server. Not required for interoperation with Windows clients. Opnum: 26
IDL_DRSFinishDemotion	Finishes or cancels the removal of a DC from an AD/LDS forest. Not required for interoperation with Windows clients. Opnum: 27

Note that gaps in the opnum numbering sequence represent opnums that MUST NOT<4> be used over the wire.

The following considerations apply to the order of method calls. See section 1.3.2 for details.

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

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

All methods MUST NOT throw exceptions.

4.1.1 IDL_DRSAddSidHistory (Opnum 20)

The **IDL_DRSAddSidHistory** method adds one or more SIDs to the sIDHistory attribute of a given object.

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

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

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

pmsgIn: Pointer to the request message.

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

pmsgOut: Pointer to the response message.

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

4.1.1.1 Method-Specific Concrete Types

4.1.1.1.1 DRS_MSG_ADDSIDREQ

The **DRS_MSG_ADDSIDREQ** union defines the request messages that are sent to the **IDL_DRSAddSidHistory** method. Only one version, identified by dwInVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.1.1.2 DRS_MSG_ADDSIDREQ_V1

The **DRS_MSG_ADDSIDREQ_V1** structure defines the request message sent to the **IDL_DRSAddSidHistory** method.

```
typedef struct {
```

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

Flags: A set of zero or more **DRS_ADDSID_FLAGS** bit flags.

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

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

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

SrcCredsUserLength: Count of characters in the **SrcCredsUser** array.

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

SrcCredsDomainLength: Count of characters in the **SrcCredsDomain** array.

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

SrcCredsPasswordLength: Count of characters in the **SrcCredsPassword** array.

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

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

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

4.1.1.1.3 DRS_MSG_ADDSIDREPLY

The **DRS_MSG_ADDSIDREPLY** union defines the response messages received from the **IDL_DRSAddSidHistory** method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

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

V1: Version 1 of the reply packet structure.

4.1.1.1.4 DRS_MSG_ADDSIDREPLY_V1

The **DRS_MSG_ADDSIDREPLY_V1** structure defines the response message received from the **IDL_DRSAddSidHistory** method.

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

dwWin32Error: Windows error code.

4.1.1.1.5 DRS_ADDSID_FLAGS

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

The valid bit flags are shown in the following diagram.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	CS	DEL	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X: Unused. MUST be zero and ignored.

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

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

This type is declared as follows:

```
typedef DWORD DRS_ADDSID_FLAGS;
```

4.1.1.2 Method-Specific Abstract Types and Procedures

4.1.1.2.1 ConnectionCtx

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

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

4.1.1.2.2 ConnectToDC

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

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

4.1.1.2.3 ConnectToDCWithCreds

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

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

4.1.1.2.4 GenerateFailureAudit

```
procedure GenerateFailureAudit()
```

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

4.1.1.2.5 GenerateSuccessAudit

```
procedure GenerateSuccessAudit()
```

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

4.1.1.2.6 GenerateSuccessAuditRemotely

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

If auditing is enabled on the server associated with ctx, the GenerateSuccessAuditRemotely procedure generates a success audit event on that server and returns true. The generated audit event indicates that an operation succeeded. Returns false if auditing is not enabled on that server. The content of the audit event is an implementation-specific behavior.<5>

4.1.1.2.7 GetKeyLength

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

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

4.1.1.2.8 GetPDC

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

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

4.1.1.2.9 HasAdminRights

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

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

4.1.1.2.10 IsAuditingEnabledForNc

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

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

4.1.1.2.11 IsLocalRpcCall

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

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

4.1.1.2.12 IsNT4SP4OrBetter

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

If the DC named in ctx is running Windows NT 4.0 and is not running at least Windows NT 4.0 SP4, this procedure returns false. Otherwise, it returns true.<6>

4.1.1.2.13 IsWellKnownDomainRelativeSid

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

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

4.1.1.2.14 LastRID

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

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

4.1.1.2.15 RemoteQuery

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

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

4.1.1.3 Server Behavior of the IDL_DRSAddSidHistory Method

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

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

- If the DS_ADDSID_FLAG_PRIVATE_CHK_SECURE flag is specified, the first variant is selected. In this variant, the method verifies only that the RPC call is secure. It does not perform any further processing or manipulate the sIDHistory attribute of any object, regardless of other flags that may be present.
- If DS_ADDSID_FLAG_PRIVATE_CHK_SECURE is not specified but DS_ADDSID_FLAG_PRIVATE_DEL_SRC_OBJ is specified, the second variant is selected. In this variant, the source and destination principals are in the same domain. The values of the objectSid and sIDHistory attributes of the source principal are added to the destination principal's

sIDHistory attribute, and then the source principal is deleted. Loosely speaking, the destination principal adopts the source principal as an "alias" and the source principal disappears.

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

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

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

if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif

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

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

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

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

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

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

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

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

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

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

  /* Destination domain must be in native mode. */
  partCtr := DescendantObject(ConfigNC(), "CN=Partitions,")
  if partCtr ≠ null
    crDst := select one dd from subtree partCtr where
```

```
        (crossRef in dd!objectClass and
         dd!nCName = dstNc)
    endif
    if partCtr = null or crDst = null then
      pmsgOut^.V1.dwWin32Error := ERROR_DS_INTERNAL_FAILURE
      return ERROR_DS_INTERNAL_FAILURE
    else
      if crDst!nTMixedDomain = 1 then
        pmsgOut^.V1.dwWin32Error := ERROR_DS_DST_DOMAIN_NOT_NATIVE
        return ERROR_DS_DST_DOMAIN_NOT_NATIVE
      endif
    endif

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

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

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

    RemoveObj(srcPrinc)
    GenerateSuccessAudit()
    return 0
  endif

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

  /* Basic parameter validation. */
  if (pmsgIn^.V1.SrcDomain = null) or
     (pmsgIn^.V1.SrcDomain = "") or
     (pmsgIn^.V1.DstDomain = null) or
     (pmsgIn^.V1.DstDomain = "") or
     (pmsgIn^.V1.SrcCredsUserLength > 0 and
        pmsgIn^.V1.SrcCredsUser = null) or
     (pmsgIn^.V1.SrcCredsDomainLength > 0 and
        pmsgIn^.V1.SrcCredsDomain = null) or
```

```
    (pmsgIn^.V1.SrcCredsPasswordLength > 0 and
       pmsgIn^.V1.SrcCredsPassword = null) or
    (pmsgIn^.V1.SrcDomainController = "") or
    (pmsgIn^.V1.SrcPrincipal = null) or
    (pmsgIn^.V1.SrcPrincipal = "") or
    (pmsgIn^.V1.DstPrincipal = null) or
    (pmsgIn^.V1.DstPrincipal = "") then
  pmsgOut^.V1.dwWin32Error := ERROR_INVALID_PARAMETER
  return ERROR_INVALID_PARAMETER
endif

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

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

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

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

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

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

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

```
  return ERROR_DS_INSUFF_ACCESS_RIGHTS
endif

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

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

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

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

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

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

/* Verify that if source domain is running Windows NT 4.0, it is
```

```
 * running at least Service Pack 4 of that operating system. */
if not IsNT4SP4OrBetter(srcCtx)
  pmsgOut^.V1.dwWin32Error := ERROR_DS_SRC_DC_MUST_BE_SP4_OR_GREATER
  return ERROR_DS_SRC_DC_MUST_BE_SP4_OR_GREATER
endif

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

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

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

/* Source and destination principals must both be computer, or both
 * be user, or both be group. The order is important: although
 * computer objects are user objects, we disallow the case where
 * one principal is a computer and the other principal is a user
 * but not a computer. */
if ((computer in srcPrinc!objectClass and
      not computer in dstPrinc!objectClass) or
    (computer in dstPrinc!objectClass and
      not computer in srcPrinc!objectClass)) or
   ((user in srcPrinc!objectClass and
      not user in dstPrinc!objectClass) or
    (user in dstPrinc!objectClass and
      not user in srcPrinc!objectClass)) or
   ((group in srcPrinc!objectClass and
      not group in dstPrinc!objectClass) or
    (group in dstPrinc!objectClass and
      not group in srcPrinc!objectClass)) then
  pmsgOut^.V1.dwWin32Error :=
      ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
  return ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
endif

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

```
                                        ADS_UF_SERVER_TRUST_ACCOUNT} then
    pmsgOut^.V1.dwWin32Error :=
        ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
    return ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH
endif

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

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

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

/* Make sure a principal with the SID of the source principal
 * does not already exist in destination domain. */
srcPrincInDst := select one o from subtree DefaultNC() where
  (o ≠ dstPrinc) and
  ((o!objectSid = srcPrincSid) or
   (o!objectSid in srcPrincSidHistory) or
   (srcPrincSid in o!sIDHistory)) or
   ((srcPrincSidHistory ∩ o!sIDHistory) ≠ {}))
if srcPrincInDst ≠ null then
  pmsgOut^.V1.dwWin32Error := ERROR_DS_SRC_SID_ALREADY_IN_FOREST
  return ERROR_DS_SRC_SID_ALREADY_IN_FOREST
endif

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

4.1.2 IDL_DRSBind (Opnum 0)

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

```
ULONG IDL_DRSBind(
  [in] handle_t rpc_handle,
```

```
    [in, unique] UUID* puuidClientDsa,
    [in, unique] DRS_EXTENSIONS* pextClient,
    [out] DRS_EXTENSIONS** ppextServer,
    [out, ref] DRS_HANDLE* phDrs
);
```

rpc_handle: An RPC binding handle, as specified in [C706].

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

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

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

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

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

4.1.2.1 Client Behavior When Sending the IDL_DRSBind Request

The client uses puuidClientDsa to pass a unique identifier. The client freely determines puuidClientDsa; the server places no constraints on it. This identifier is intended for diagnostic purposes only.<7>

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

The remaining information in the DRS_EXTENSIONS_INT structure must be consistent with the client's capabilities. This information affects the versions of response structures that the server returns in method calls using the **DRS_HANDLE** returned by **IDL_DRSBind**. In descriptions of method calls that use a **DRS_HANDLE**, this handle is sometimes called the client's RPC context.<8>

If a method of this protocol takes a parameter named *dwInVersion*, the client uses that parameter to specify the version of the referent of the next parameter to that method, often named *pmsgIn*. The referent of this parameter is called the method's request. The *dwInVersion* parameter is called the request version. For example, if the client passes *dwInVersion* = 2 to **IDL_DRSGetReplInfo**, the client also passes a **DRS_MSG_GETREPLINFO_REQ_V2** request.

If a method of this protocol takes an integer parameter named *pdwOutVersion*, the server uses that parameter to return the version number of the referent of the next parameter to that method, often named *pmsgOut*. The referent of this parameter is called the method's response. The referent of *pdwOutVersion* is called the response version. For example, when the server returns *pdwOutVersion*^ = 2 from **IDL_DRSDomainControllerInfo**, the server also returns a **DRS_MSG_DCINFOREPLY_V2** response.

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

- **IDL_DRSDomainControllerInfo** has only one request version; it contains an **InfoLevel** field. The **InfoLevel**, not the dwInputVersion, determines the response version. Similarly,

IDL_DRSGetReplInfo has two request versions, which both contain an **InfoType** field. The **InfoType**, not the dwInputVersion, determines the response version.

The following tables describe how the server determines the response version based on the request version, the DRS_EXTENSIONS_INT structure specified when creating the **DRS_HANDLE**, and in some cases, the contents of the request message. The tables show all the valid combinations. All cases not shown in the tables below are a client error; the server returns ERROR_INVALID_PARAMETER.

IDL_DRSReplicaSync

Request version	Response version
1	-

IDL_DRSUpdateRefs

Request version	Response version
1	-

IDL_DRSReplicaAdd

Request version	Response version
1	-
2	-

IDL_DRSReplicaDel

Request version	Response version
1	-

IDL_DRSReplicaModify

Request version	Response version
1	-

IDL_DRSCrackNames

Request version	Response version
1	1

IDL_DRSWriteSPN

Request version	Response version
1	1

IDL_DRSRemoveDsServer

Request version	Response version
1	1

IDL_DRSRemoveDsDomain

Request version	Response version
1	1

IDL_DRSDomainControllerInfo

Request version	Response version
1	request.InfoLevel [1]

IDL_DRSExecuteKCC

Request version	Response version
1	1

IDL_DRSGetReplInfo

Request version	Response version
1	request.InfoType [2]
2	request.InfoType [2]

IDL_DRSAddSidHistory

Request version	Response version
1	1

IDL_DRSReplicaVerifyObjects

Request version	Response version
1	-

IDL_DRSQuerySitesByCost

Request version	Response version
1	1

IDL_DRSInitDemotion

Request version	Response version
1	1

IDL_DRSReplicaDemotion

Request version	Response version
1	1

IDL_DRSFinishDemotion

Request version	Response version
1	1

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

[2] Possible values are detailed in section 4.1.7.

4.1.2.2 Server Behavior of the IDL_DRSBind Method

The server retains the DRS_EXTENSIONS_INT structure passed as pextClient^.

The server sets ppextServer to a DRS_EXTENSIONS_INT structure whose **dwReplEpoch** field is initialized by reading the value of msDS-ReplicationEpoch from its nTDSDSA object and assigning this value to it, and whose other fields describe the server. The server then returns a **DRS_HANDLE** as the referent of phDrs. The **dwReplEpoch** field is for server-to-server replication implementation only, the client does not interpret it. The field MAY have meaning to Windows server implementations, but its meaning is not significant to Windows clients.

The following tables specify the capability assertions made by a server that sets bits in the DRS_EXTENSIONS_INT structure returned from **IDL_DRSBind**. Each row of a table gives a request version (including both dwInVersion and the InfoLevel of **IDL_DRSDomainControllerInfo** and the InfoType of **IDL_DRSGetReplInfo**) and the DRS_EXTENSIONS_INT bit or bits that the server sets to indicate support for that request.

A server supports all requests via the RPC transport.

IDL_DRSReplicaSync

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

IDL_DRSUpdateRefs

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

IDL_DRSReplicaAdd

Request version	DRS_EXTENSIONS_INT bit(s)
1	-
2	DRS_EXT_ASYNCREPL

IDL_DRSReplicaDel

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

IDL_DRSReplicaModify

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

IDL_DRSCrackNames

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

IDL_DRSWriteSPN

Request version	DRS_EXTENSIONS_INT bit(s)
1	-

IDL_DRSRemoveDsServer

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_REMOVEAPI

IDL_DRSRemoveDsDomain

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_REMOVEAPI

IDL_DRSDomainControllerInfo

Request version	DRS_EXTENSIONS_INT bit(s)
1 InfoLevel = 0x1	DRS_EXT_DCINFO_V1
1 InfoLevel = 0x2	DRS_EXT_DCINFO_V2

Request version	DRS_EXTENSIONS_INT bit(s)
1 InfoLevel = 0x3	DRS_ EXT_LH_BETA2
1 InfoLevel = 0xffffffff	DRS_EXT_DCINFO_VFFFFFFFF

IDL_DRSExecuteKCC

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_KCC_EXECUTE

IDL_DRSGetReplInfo

Request version	DRS_EXTENSIONS_INT bit(s)
1	-
2	DRS_EXT_GETCHGREQ_V8
2 InfoType = [3..5]	DRS_EXT_POST_BETA3
2 InfoType = 6	DRS_EXT_GETCHGREQ_V8
2 InfoType = [7..10]	DRS_EXT_GETCHGREPLY_V6

IDL_DRSAddSidHistory

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_ADD_SID_HISTORY

IDL_DRSReplicaVerifyObjects

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_VERIFY_OBJECT

IDL_DRSQuerySitesByCost

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_QUERYSITESBYCOST_V1

IDL_DRSInitDemotion

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_ADAM

IDL_DRSReplicaDemotion

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_ADAM

IDL_DRSFinishDemotion

Request version	DRS_EXTENSIONS_INT bit(s)
1	DRS_EXT_ADAM

4.1.2.3 Client Behavior When Receiving the IDL_DRSBind Response

The client receives a DRS_EXTENSIONS_INT structure from the server as the referent of ppextServer.

A server supports only a subset of the possible request versions, including both dwInVersion and the InfoLevel of **IDL_DRSDomainControllerInfo** and the InfoType of **IDL_DRSGetReplInfo**. The server informs the client of its capabilities via the DRS_EXTENSIONS_INT structure returned from **IDL_DRSBind**, as described in Server Behavior of the IDL_DRSBind Method (section 4.1.2.2).

The client receives a **DRS_HANDLE** as the referent of phDrs.

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

4.1.2.4 Examples of the IDL_DRSBind Method

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

4.1.2.4.1 Initial State

Querying the nTDSDSA objects for the **root domain** NC DC=CONTOSO, DC=COM for DC1:

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

  3> objectClass: top; applicationSettings; nTDSDSA;

1> cn: NTDS Settings;

1> distinguishedName: CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com;

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

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

4.1.2.4.2 Client Request

A client invokes the **IDL_DRSBind** method against DC1, with the following parameters (**DRS_HANDLE** to DC1 omitted):

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

4.1.2.4.3 Server Response

Return code of 0 (**DRS_HANDLE** to DC1 omitted) with the following values:

- ppextServer:
  - cb: 0x1c
  - dwFlags:
    - DRS_EXT_BASE
    - DRS_EXT_ASYNCREPL
    - DRS_EXT_REMOVEAPI
    - DRS_EXT_MOVEREQ_V2
    - DRS_EXT_GETCHG_ DEFLATE
    - DRS_EXT_DCINFO_V1
    - DRS_EXT_RESTORE_USN_OPTIMIZATION
    - DRS_EXT_KCC_EXECUTE
    - DRS_EXT_ADDENTRY_V2
    - DRS_EXT_LINKED_VALUE_REPLICATION
    - DRS_EXT_DCINFO_V2

- DRS_EXT_INSTANCE_TYPE_NOT_REQ_ON_MOD
- DRS_EXT_GET_REPL_INFO
- DRS_EXT_STRONG_ENCRYPTION
- DRS_EXT_DCINFO_VFFFFFFFF
- DRS_EXT_TRANSITIVE_MEMBERSHIP
- DRS_EXT_ADD_SID_HISTORY
- DRS_EXT_POST_BETA3
- DRS_EXT_GETCHGREQ_V5
- DRS_EXT_GET_MEMBERSHIPS2
- DRS_EXT_GETCHGREQ_V6
- DRS_EXT_NONDOMAIN_NCS
- DRS_EXT_GETCHGREQ_V8
- DRS_EXT_GETCHGREPLY_V5
- DRS_EXT_GETCHGREPLY_V6
- DRS_EXT_WHISTLER_BETA3
- DRS_EXT_W2K3_DEFLATE

- SiteObjGuid: GUID {620954c7-7044-400f-9c0b-5c9154198aa6}
- Pid: 632
- dwReplEpoch: 0

4.1.2.4.4 Final State

No change in state.

4.1.3 IDL_DRSCrackNames (Opnum 12)

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

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

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

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

4.1.3.1 Method-Specific Concrete Types

4.1.3.1.1 DRS_MSG_CRACKREQ

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

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

V1: Version 1 request.

4.1.3.1.2 DRS_MSG_CRACKREQ_V1

The **DRS_MSG_CRACKREQ_V1** structure defines the request message sent to the **IDL_DRSCrackNames** method.

```
typedef struct {
  ULONG CodePage;
  ULONG LocaleId;
  DWORD dwFlags;
  DWORD formatOffered;
  DWORD formatDesired;
  [range(1,10000)] DWORD cNames;
  [string, size_is(cNames)] WCHAR** rpNames;
} DRS_MSG_CRACKREQ_V1;
```

CodePage: The character set used by the client.

LocaleId: The locale used by the client.

dwFlags: Zero or more of the following bit flags:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	GC	TR	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	FPO

X: Unused. MUST be zero and ignored.

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

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

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

formatOffered: The format of the names in **rpNames**. This may be one of the values from DS_NAME_FORMAT (section 4.1.3.1.3) or one of the following:

Value	Meaning
DS_LIST_SITES 0xFFFFFFFF	Get all sites in the forest.
DS_LIST_SERVERS_IN_SITE 0xFFFFFFFE	Get all servers in a given site.
DS_LIST_DOMAINS_IN_SITE 0xFFFFFFFD	Get all domains in a given site.
DS_LIST_SERVERS_FOR_DOMAIN_IN_SITE 0xFFFFFFFC	Get all DCs of a specified domain in a given site.
DS_LIST_INFO_FOR_SERVER 0xFFFFFFFB	Get DNS host name and server reference for a given DC.
DS_LIST_ROLES 0xFFFFFFFA	Get FSMO role owners.
DS_NT4_ACCOUNT_NAME_SANS_DOMAIN 0xFFFFFFF9	Get value of sAMAccountName attribute.
DS_MAP_SCHEMA_GUID 0xFFFFFFF8	Get LDAP display name from schema GUID.
DS_LIST_DOMAINS 0xFFFFFFF7	Get all domains in the forest.
DS_LIST_NCS 0xFFFFFFF6	Get all NCs in the forest.
DS_ALT_SECURITY_IDENTITIES_NAME 0xFFFFFFF5	Alternate security identifier.
DS_STRING_SID_NAME	String form of SID.

Value	Meaning
0xFFFFFFF4	
DS_LIST_SERVERS_WITH_DCS_IN_SITE 0xFFFFFFF3	Get all DCs in a given site.
DS_LIST_GLOBAL_CATALOG_SERVERS 0xFFFFFFF1	Get all GCs.
DS_NT4_ACCOUNT_NAME_SANS_DOMAIN_EX 0xFFFFFFF0	Get value of sAMAccountName attribute; return status DS_NAME_ERROR_NOT_FOUND if account is invalid.
DS_USER_PRINCIPAL_NAME_AND_ALTSECID 0xFFFFFFEF	The user principal name and alternate security identifier.

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

Value	Meaning
DS_STRING_SID_NAME 0xFFFFFFF4	String form of a SID.
DS_USER_PRINCIPAL_NAME_FOR_LOGON 0xFFFFFFF2	User principal name.

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

rpNames: Input names to translate.

4.1.3.1.3 DS_NAME_FORMAT

The **DS_NAME_FORMAT** enumeration describes the format of a name sent to or received from the **IDL_DRSCrackNames** method.

```
typedef  enum
{
  DS_UNKNOWN_NAME = 0,
  DS_FQDN_1779_NAME = 1,
  DS_NT4_ACCOUNT_NAME = 2,
  DS_DISPLAY_NAME = 3,
  DS_UNIQUE_ID_NAME = 6,
  DS_CANONICAL_NAME = 7,
  DS_USER_PRINCIPAL_NAME = 8,
  DS_CANONICAL_NAME_EX = 9,
  DS_SERVICE_PRINCIPAL_NAME = 10,
  DS_SID_OR_SID_HISTORY_NAME = 11,
  DS_DNS_DOMAIN_NAME = 12
} DS_NAME_FORMAT;
```

DS_UNKNOWN_NAME: The server looks up the name by using the algorithm specified in the LookupUnknownName procedure.

DS_FQDN_1779_NAME: A distinguished name.

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

DS_DISPLAY_NAME: A user-friendly display name.

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

DS_CANONICAL_NAME: A canonical name.

DS_USER_PRINCIPAL_NAME: User principal name.

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

DS_SERVICE_PRINCIPAL_NAME: Service principal name (SPN).

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

DS_DNS_DOMAIN_NAME: Not supported.

4.1.3.1.4 DS_NAME_RESULT_ITEMW

The **DS_NAME_RESULT_ITEMW** structure defines the translated name returned by the **IDL_DRSCrackNames** method.

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

status: Status of the crack name operation for the corresponding element of the **rpNames** field in the request. The status is one of the values from the enumeration DS_NAME_ERROR.

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

pName: Object name in the requested format.

4.1.3.1.5 DS_NAME_RESULTW

The **DS_NAME_RESULTW** structure defines the translated names returned by the **IDL_DRSCrackNames** method.

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

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

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

4.1.3.1.6 DRS_MSG_CRACKREPLY

The **DRS_MSG_CRACKREPLY** union defines the response messages received from the **IDL_DRSCrackNames** method.

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

V1: Version 1 reply.

4.1.3.1.7 DRS_MSG_CRACKREPLY_V1

The **DRS_MSG_CRACKREPLY_V1** structure defines the response message received from the IDL_DRSCrackNames method.

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

pResult: Translated form of the names.

4.1.3.1.8 DS_NAME_ERROR

This section enumerates the possible statuses of a translation operation.

Symbolic name	Value
DS_NAME_NO_ERROR	0
DS_NAME_ERROR_RESOLVING	1
DS_NAME_ERROR_NOT_FOUND	2
DS_NAME_ERROR_NOT_UNIQUE	3
DS_NAME_ERROR_NO_MAPPING	4
DS_NAME_ERROR_DOMAIN_ONLY	5
DS_NAME_ERROR_TRUST_REFERRAL	7
DS_NAME_ERROR_IS_SID_HISTORY_UNKNOWN	0xFFFFFFF2

Symbolic name	Value
DS_NAME_ERROR_IS_SID_HISTORY_ALIAS	0xFFFFFFF3
DS_NAME_ERROR_IS_SID_HISTORY_GROUP	0xFFFFFFF4
DS_NAME_ERROR_IS_SID_HISTORY_USER	0xFFFFFFF5
DS_NAME_ERROR_IS_SID_UNKNOWN	0xFFFFFFF6
DS_NAME_ERROR_IS_SID_ALIAS	0xFFFFFFF7
DS_NAME_ERROR_IS_SID_GROUP	0xFFFFFFF8
DS_NAME_ERROR_IS_SID_USER	0xFFFFFFF9
DS_NAME_ERROR_SCHEMA_GUID_CONTROL_RIGHT	0xFFFFFFFA
DS_NAME_ERROR_SCHEMA_GUID_CLASS	0xFFFFFFFB
DS_NAME_ERROR_SCHEMA_GUID_ATTR_SET	0xFFFFFFFC
DS_NAME_ERROR_SCHEMA_GUID_ATTR	0xFFFFFFFD
DS_NAME_ERROR_SCHEMA_GUID_NOT_FOUND	0xFFFFFFFE
DS_NAME_ERROR_IS_FPO	0xFFFFFFFF

4.1.3.2 Method-Specific Abstract Types and Procedures

4.1.3.2.1 CanonicalNameFromCanonicalNameEx

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

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

4.1.3.2.2 DomainDNSNameFromDomain

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

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

4.1.3.2.3 DomainFromDomainDNSName

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

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

4.1.3.2.4 DomainNameFromCanonicalName

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

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

4.1.3.2.5 DomainNameFromSid

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

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

4.1.3.2.6 DomainNameFromUPN

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

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

4.1.3.2.7 DomainNetBIOSNameFromDomain

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

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

4.1.3.2.8 DomainSidFromSid

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

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

4.1.3.2.9 LookupName

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

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

```
rt: sequence of DSName
obj: DSName
fSidHistory: boolean
result: DS_NAME_RESULT_ITEM
names: sequence of unicodestring

if formatOffered = DS_UNKNOWN_NAME then
  return LookupUnknownName(flags, name)
endif

if formatOffered = DS_FQDN_1779_NAME then
  rt := LookupAttr(flags, distinguishedName, name)
else if formatOffered = DS_NT4_ACCOUNT_NAME then
  rt := LookupAttr(flags, sAMAccountName,
                   UserNameFromNT4AccountName(name))
else if formatOffered = DS_USER_PRINCIPAL_NAME then
  rt := LookupAttr(flags, userPrincipalName, name)
else if formatOffered = DS_CANONICAL_NAME then
  rt := LookupCanonicalName(name)
else if formatOffered = DS_UNIQUE_ID_NAME then
  rt := LookupAttr(flags, objectGUID, name)
else if formatOffered = DS_DISPLAY_NAME then
  rt := LookupAttr(flags, displayName, name)
else if formatOffered = DS_SERVICE_PRINCIPAL_NAME then
  rt := LookupSPN(flags, name)
else if formatOffered in {DS_SID_OR_SID_HISTORY_NAME,
                          DS_STRING_SID_NAME} then
  rt := LookupSID(flags, SidFromStringSid(name))
else if formatOffered = DS_CANONICAL_NAME_EX then
  rt := LookupCanonicalName(CanonicalNameFromCanonicalNameEx(name))
else if formatOffered in {DS_NT4_ACCOUNT_NAME_SANS_DOMAIN,
    DS_NT4_ACCOUNT_NAME_SANS_DOMAIN_EX} then
  rt := LookupAttr(flags, sAMAccountName, name)
else if formatOffered = DS_ALT_SECURITY_IDENTITIES_NAME then
  rt := LookupAttr(flags, altSecurityIdentities, name)
else if formatOffered = DS_USER_PRINCIPAL_NAME_AND_ALTSECID then
  rt := LookupUPNAndAltSecID(flags, name)
else
  rt := null
endif

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

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

```
obj := rt[0]

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

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

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

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

  if obj!sAMAccountType in {SAM_USER_OBJECT, SAM_MACHINE_ACCOUNT,
      SAM_TRUST_ACCOUNT} then
    if fSidHistory then
      result.status := DS_NAME_ERROR_IS_SID_HISTORY_USER
    else
      result.status := DS_NAME_ERROR_IS_SID_USER
    endif
  else if obj!sAMAccountType in {SAM_NON_SECURITY_GROUP_OBJECT,
      SAM_GROUP_OBJECT} then
    if fSidHistory then
      result.status := DS_NAME_ERROR_IS_SID_HISTORY_GROUP
    else
      result.status := DS_NAME_ERROR_IS_SID_GROUP
    endif
  else if obj!sAMAccountType in {SAM_NON_SECURITY_ALIAS_OBJECT,
      SAM_ALIAS_OBJECT} then
    if fSidHistory then
      result.status := DS_NAME_ERROR_IS_SID_HISTORY_ALIAS
    else
      result.status := DS_NAME_ERROR_IS_SID_ALIAS
    endif
  else
    if fSidHistory then
      result.status := DS_NAME_ERROR_IS_SID_HISTORY_UNKNOWN
```

```
    else
      result.status := DS_NAME_ERROR_IS_SID_UNKNOWN
    endif
  endif
endif

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

return result
```

4.1.3.2.10 LookupAttr

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

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

```
rt: set of DSName

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

4.1.3.2.11 LookupCanonicalName

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

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

```
curObj: DSName
label: unicodestring

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

```
      return null
    endif
  endwhile
  return curObj
```

4.1.3.2.12 GetCanonicalName

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

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

```
result: unicodestring

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

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

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

4.1.3.2.13 LookupSPN

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

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

```
rt: set of DSName
obj: DSName
dcGuid: GUID
spnMappings: set of unicodestring
mappedSpn: unicodestring
/* First, try to look up the SPN directly. */
rt := LookupAttr(flags, servicePrincipalName, name)
if rt ≠ null then
  return rt
endif
/* Obtain SPN mappings value. */
obj := DescendantObject(ConfigNC(),
    "CN=Directory Service,CN=Windows,CN=Services,")
spnMappings := obj!sPNMappings
if spnMappings ≠ null
```

```
  mappedSpn := MapSPN(name, spnMappings)
  if mappedSpn ≠ null then
    /* try to lookup a mapped SPN */
    rt := LookupAttr(flags, servicePrincipalName, mappedSpn)
    if rt ≠ null then
      return rt
    endif
  endif
endif
return rt
```

4.1.3.2.14 LookupSID

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

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

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

return rt1 + rt2
```

4.1.3.2.15 LookupUnknownName

```
procedure LookupUnknownName(
  flags: DWORD,
  name: unicodestring): DS_NAME_RESULT_ITEM
```

Informative summary of behavior: The server uses LookupUnknownName to look up names of format DS_UNKNOWN_NAME. LookupUnknownName looks up the name by trying formats in the specific order listed in the foreach statement shown below until a lookup succeeds.

```
result: DS_NAME_RESULT_ITEM
format: DWORD

/* Attempt to resolve in the following formats in this specific
 * order. */
foreach format in {DS_FQDN_1779_NAME, DS_USER_PRINCIPAL_NAME,
                   DS_NT4_ACCOUNT_NAME, DS_CANONICAL_NAME,
                   DS_UNIQUE_ID_NAME, DS_DISPLAY_NAME,
                   DS_SERVICE_PRINCIPAL_NAME,
                   DS_SID_OR_SID_HISTORY_NAME,
                   DS_CANONICAL_NAME_EX}
  result := LookupName(flags, format, name)
  if result.status ≠ DS_NAME_ERROR_NOT_FOUND then
    return result
  endif
endfor
return result
```

4.1.3.2.16 LookupUPNAndAltSecID

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

Informative summary of behavior: Returns DSNames of objects, with the given value as a value of userPrincipalName, altSecurityIdentities, or sAMAccountName.

```
rt: set of DSName
/* First, try lookup by userPrincipalName. */
rt := LookupAttr(flags, userPrincipalName, name)
if rt ≠ null then
  return rt
endif
/* Next, try lookup by altSecurityIdentities. */
rt := LookupAttr(flags, altSecurityIdentities, name)
if rt ≠ null then
  return rt
endif
/* Finally, attempt to parse the name as simpleName@domain and
 * search for
 * sAMAccountName=simpleName. */
name := UserNameFromUPN(name)
if name ≠ null then
  rt := LookupAttr(flags, sAMAccountName, name)
endif
return rt
```

4.1.3.2.17 LookupDomainSyntactically

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

Informative summary of behavior: LookupDomainSyntactically is a helper function that attempts to obtain the domain name from a name that is otherwise unresolvable. If the domain name can be parsed out, and if it represents a trusted domain, then this domain name is returned in the DS_NAME_RESULT_ITEM.pDomain member variable, with an appropriate status value.

```
domainSID: SID
domainName: unicodestring
domainDN: unicodestring
result: DS_NAME_RESULT_ITEM

domainName := null
if formatOffered = DS_FQDN_1779_NAME then
  domainDN := RetrieveDCSuffixFromDn(name)
  domainName := DomainDNSNameFromDomain(domainDN)
else if formatOffered = DS_NT4_ACCOUNT_NAME then
  domainName := DomainNameFromNT4AccountName(name)
else if formatOffered in {DS_CANONICAL_NAME,
```

```
      DS_CANONICAL_NAME_EX} then
  domainName := DomainNameFromCanonicalName(name)
else if formatOffered = DS_USER_PRINCIPAL_NAME then
  domainName := DomainNameFromUPN(name)
else if formatOffered = DS_SERVICE_PRINCIPAL_NAME then
  domainName := GetServiceNameFromSPN(name)
else if formatOffered = DS_SID_OR_SID_HISTORY_NAME then
  domainSID := DomainSidFromSid(SidFromStringSid(name))
  domainName := DomainNameFromSid(domainSID)
endif

if domainName ≠ null and IsDomainNameInTrustedForest(domainName) then
  /* We found domain name syntactically, and it is a domain we
   * trust. */
  result.pDomain^ := domainName
  if DS_NAME_FLAG_TRUST_REFERRAL in flags then
    result.status := DS_NAME_ERROR_TRUST_REFERRAL
  else
    result.status := DS_NAME_ERROR_DOMAIN_ONLY
  endif
else
  /* We could not even parse the domain out. */
  result.pDomain^ := null
  result.status := DS_NAME_ERROR_NOT_FOUND
endif

return result
```

4.1.3.2.18 MapSPN

```
procedure MapSPN(spn: unicodestring,
                 spnMappings: set of unicodestring):
    unicodestring
```

The MapSPN procedure performs an SPN mapping operation on spn according to the map specified in spnMappings, and returns the mapped version of spn. The mapping operation is used to change the service class of the SPN. An SPN service class is the first part of an SPN; for example, "ldap" is the service class of the SPN "ldap/fabrikam.com".

Each value of spnMappings consists of an alias, followed by an equals sigh (=), followed by a comma-separated list of one or more SPN service classes. Thus, each value must be in the following format:

alias=serviceClass1,serviceClass2,serviceClass3,...,serviceClassN

If the service class portion of spn corresponds to one of the serviceClassX values in value v of spnMappings, then the return value of this procedure is the SPN value this is constructed from spn by substituting the alias value from v as the service class of spn. If no mapping is found (that is, if there is no such v), or if spn is not an SPN, then null is returned.

For example, suppose that spnMappings is the following set:

{"ldap=ldap,otherldap", "host=alerter,appmgmt,cisvc"}

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

4.1.3.2.19 ParseCanonicalName

```
procedure ParseCanonicalName(
  name: unicodestring,
  var firstPart: unicodestring,
  var remainder: unicodestring)
```

The ParseCanonicalName procedure parses the first label from the canonical name string name and returns the first label in firstPart and the remainder of the string in remainder. For example, name = "container1/container2/leaf" is parsed as firstPart:= "container1" and remainder:= "container2/leaf". As another example, name = "example.fabrikam.com/container/username" is parsed as firstPart:= "example.fabrikam.com" and remainder:= "container/username". If name does not contain a slash character, then it is parsed as firstPart:= name and remainder:= null.

4.1.3.2.20 RetrieveDCSuffixFromDn

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

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

4.1.3.2.21 UserNameFromUPN

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

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

4.1.3.2.22 ConstructOutput

```
procedure ConstructOutput(
  obj: DSName,
  formatDesired: DWORD): set of unicodestring
```

Informative summary of behavior: ConstructOutput is a helper function that constructs the name of the object in the required output format. Note that the returned set of values may be empty, or may contain more than one value. These situations are handled by the caller function, LookupName (section 4.1.3.2.9).

```
if formatDesired = DS_FQDN_1779_NAME then
  return {obj!distinguishedName}
else if formatDesired = DS_NT4_ACCOUNT_NAME then
  return {DomainNetBIOSNameFromDomain(GetObjectNC(obj)) + "\" +
          obj!sAMAccountName}
else if formatDesired = DS_USER_PRINCIPAL_NAME then
  return {obj!userPrincipalName}
else if formatDesired = DS_CANONICAL_NAME then
  return {GetCanonicalName(obj, false)}
```

```
else if formatDesired = DS_UNIQUE_ID_NAME then
  return {GuidToString(obj!objectGUID)}
else if formatDesired = DS_DISPLAY_NAME then
  return {obj!displayName}
else if formatDesired = DS_SERVICE_PRINCIPAL_NAME then
  return obj!servicePrincipalName
else if formatDesired = DS_CANONICAL_NAME_EX then
  return {GetCanonicalName(obj, true)}
else if formatDesired = DS_STRING_SID_NAME then
  return {StringSidFromSid(obj!objectSid)}
else if formatDesired = DS_USER_PRINCIPAL_NAME_FOR_LOGON then
  /* If UPN is set, then return it. */
  if obj!userPrincipalName ≠ null then
    return {obj!userPrincipalName}
  endif
  return {obj!sAMAccountName + "@" +
    DomainDNSNameFromDomain(GetObjectNC(obj))}
endif
/* Otherwise, unknown format. */
return null
```

4.1.3.3 Server Behavior of the IDL_DRSCrackNames Method

Informative summary of behavior: The IDL_DRSCrackNames method is a generic method that is used to look up information in the directory. The most common usage is looking up directory object names that are provided in one format (for example, SPNs) and returning them in a different format (for example, DNs). One special mode occurs when the input format is not specified, in which case the server tries to "guess" the format of the name by following some heuristics. The method can also be used to look up generic information in the directory, such as the list of sites, or the list of servers in a specific site.

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

msgIn: DRS_MSG_CRACKREQ_V1
msgOut: DS_NAME_RESULTW
i: DWORD
rt: set of DSName
serverObj, siteObj, attr, class, er: DSName

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

```
endif

if msgIn.formatOffered in {
    all constants in DS_NAME_FORMAT enumeration,
    DS_NT4_ACCOUNT_NAME_SANS_DOMAIN,
    DS_NT4_ACCOUNT_NAME_SANS_DOMAIN_EX,
    DS_ALT_SECURITY_IDENTITIES_NAME,
    DS_STRING_SID_NAME,
    DS_USER_PRINCIPAL_NAME_AND_ALTSECID} then
  /* Regular name lookup. */
  for i := 0 to msgIn.cNames - 1
    /* Perform the lookup based on the input format. */
    msgOut.rItems[i] := LookupName(
             msgIn.dwFlags, msgIn.formatOffered, msgIn.rItems[i])
    if msgOut.rItems[i] = null then
      /* Did not find anything. Try to at least guess the domain
       * name from the input name. */
      msgOut.rItems[i] := LookupDomainSyntactically(msgIn.dwFlags,
          msgIn.formatOffered, msgIn.rItems[i])
    endif
  endfor
  msgOut.cItems = msgIn.cNames
else if msgIn.formatOffered = DS_LIST_ROLES then
  /* Return the list of FSMO role owners. */
  i := 0
  foreach role in {FSMO_SCHEMA, FSMO_DOMAIN_NAMING, FSMO_PDC,
                   FSMO_RID, FSMO_INFRASTRUCTURE}
    msgOut.rItems[i].pName := GetFSMORoleOwner(role).dn
    msgOut.rItems[i].status := DS_NAME_NO_ERROR
    i := i + 1
  endfor
  msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_SITES then
  /* Return the list of known sites. */
  rt := select all o from children
        DescendantObject(ConfigNC(),"CN=Sites,")
        where o!objectCategory = GetDefaultObjectCategory(site)
  i := 0
  foreach siteObj in rt
    msgOut.rItems[i].pName := siteObj.dn
    msgOut.rItems[i].status := DS_NAME_NO_ERROR
    i := i + 1
  endfor
  msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_SERVERS_IN_SITE then
  /* Return all DCs in a site named msgIn.rItems[0]. */
  rt := select all o from subtree msgIn.rItems[0]
      where o!objectCategory = GetDefaultObjectCategory(server)
  i := 0
  foreach serverObj in rt
    msgOut.rItems[i].pName := serverObj.dn
    msgOut.rItems[i].status := DS_NAME_NO_ERROR
    i := i + 1
  endfor
  msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_DOMAINS then
  /* Return all known AD domains. */
  rt := select all o from
```

```
      subtree DescendantObject(ConfigNC(), "CN=Partitions")
        where o!objectCategory = GetDefaultObjectCategory(crossRef)
        and FLAG_CR_NTDS_DOMAIN in o!systemFlags
  i := 0
  foreach crObj in rt
    msgOut.rItems[i].pName := crObj!ncName.dn
    msgOut.rItems[i].status := DS_NAME_NO_ERROR
    i := i + 1
  endfor
  msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_NCS then
  /* Return all known NCs. */
  rt := select all o from
      subtree DescendantObject(ConfigNC(), "CN=Partitions")
        where o!objectCategory = GetDefaultObjectCategory(crossRef)
  i := 0
  foreach crObj in rt
    msgOut.rItems[i].pName := crObj!ncName.dn
    msgOut.rItems[i].status := DS_NAME_NO_ERROR
    i := i + 1
  endfor
  msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_DOMAINS_IN_SITE then
  /* Return the list of domains that are hosted by DCs in a site
   * named msgIn.rItems[0]. */
  /* First find all DCs in a site named msgIn.rItems[0]. */
  rt := select all o from subtree msgIn.rItems[0]
      where o!objectCategory = GetDefaultObjectCategory(nTDSDSA)
  /* Gather the list of all domains from DSA object. */
  hostedDomains := null
  foreach dsaObj in rt
    /* Union operation eliminates duplicates. */
    hostedDomains := hostedDomains + dsaObj!hasMasterNCs
  endfor
  i := 0
  foreach domain in hostedDomains
    if domain ≠ SchemaNC() and domain ≠ ConfigNC() then
      msgOut.rItems[i].pName := domain.dn
      msgOut.rItems[i].status := DS_NAME_NO_ERROR
      i := i + 1
    endif
  endfor
  msgOut.cItems := i
else if msgIn.formatOffered = DS_LIST_SERVERS_FOR_DOMAIN_IN_SITE then
  /* Return all DSAs hosting domain msgIn.rItems[0] in a site named
   * msgIn.rItems[0]. */
  rt := select all o from subtree msgIn.rItems[0]
      where o!objectCategory = GetDefaultObjectCategory(nTDSDSA)
        and msgIn.rItems[0] in o!msDS-hasMasterNCs
  /* Return the list of server objects (parents of DSAs). */
  i := 0
  foreach dsaObj in rt
    serverObj := select one o from subtree ConfigNC() where
        o!objectGUID = dsaObj!parent
    msgOut.rItems[i].pName := serverObj.dn
    msgOut.rItems[i].status := DS_NAME_NO_ERROR
    i := i + 1
  endfor
```

```
    msgOut.cItems := i
  else if msgIn.formatOffered = DS_LIST_SERVERS_WITH_DCS_IN_SITE then
    /* Return all servers that have DSA objects in a site named
     * msgIn.rItems[0]. */
    rt := select all o from subtree msgIn.rItems[0]
        where o!objectCategory = GetDefaultObjectCategory(nTDSDSA)
          and o!hasMasterNCs ≠ null
    /* Return the list of server objects (parents of DSAs). */
    i := 0
    foreach dsaObj in rt
      serverObj := select one o from subtree ConfigNC() where
          o!objectGUID = dsaObj!parent
      msgOut.rItems[i].pName := serverObj.dn
      msgOut.rItems[i].status := DS_NAME_NO_ERROR
      i := i + 1
    endfor
    msgOut.cItems := i
  else if msgIn.formatOffered = DS_LIST_INFO_FOR_SERVER then
    /* Returns the DSA object, the dnsHostName and the serverReference
     * for the server specified by msgIn.rItems[0]. */
    serverObj := GetDSNameFromDN(msgIn.rItems[0])
    dsaObj := select one o from subtree msgIn.rItems[0]
        where o!objectCategory = GetDefaultObjectCategory(nTDSDSA)
    if dsaObj ≠ null then
      /* Ok, looks like a valid server object. */
      msgOut.rItems[0].pName := dsaObj.dn
      msgOut.rItems[0].status := DS_NAME_NO_ERROR
      msgOut.rItems[1].pName := serverObj.dn!dnsHostName
      msgOut.rItems[1].status := DS_NAME_NO_ERROR
      msgOut.rItems[2].pName := dsaObj!serverReference
      msgOut.rItems[2].status := DS_NAME_NO_ERROR
      msgOut.cItems := 3
    endif
  else if msgIn.formatOffered = DS_LIST_GLOBAL_CATALOG_SERVERS then
    /* Returns the list of GC servers, including the info which site
     * each GC belongs to. */
    rt := select all o from subtree ConfigNC()
        where O!objectCategory = GetDefaultObjectCategory(nTDSDSA)
          and NTDSDSA_OPT_IS_GC in o!options and o!invocationId ≠ null
    i := 0
    foreach dsaObj in rt
      /* server object is the parent of the DSA object. */
      serverObj := select one o from subtree ConfigNC() where
          o!objectGUID = dsaObj!parent
      /* Site object is the parent of the server object. */
      siteObj := select one o from subtree ConfigNC() where
          o!objectGUID = serverObj!parent
      msgOut.rItems[i].pDomain := serverObj!dnsHostName
      msgOut.rItems[i].pName := siteObj.dn
      msgOut.rItems[i].status := DS_NAME_NO_ERROR
      i := i+1
    endfor
    msgOut.cItems := i
  else if msgIn.formatOffered = DS_MAP_SCHEMA_GUID then
    for i := 0 to msgIn.cNames - 1
      /* Map a guid contained in msgIn.rItems[i] to attribute or class
       * or propertySet.*/
      /* Assume no match by default. */
```

```
    msgOut.rItems[i].status := DS_NAME_ERROR_NOT_FOUND

    /* First, try to find a matching attribute. */
    attr := select one o from subtree SchemaNC()
        where attributeSchema in o!objectClass and
          o!schemaIdGuid = msgIn.rItems[i]
    if attr ≠ null
      /* Found a matching attribute object. */
      msgOut.rItems[i].pName := attr!lDAPDisplayName
      msgOut.rItems[i].status := DS_NAME_ERROR_SCHEMA_GUID_ATTR
    else
      /* Next, try to find a matching class. */
      class := select one o from subtree SchemaNC()
          where classSchema in o!objectClass
            o!schemaIdGuid = msgIn.rItems[i]
      if class ≠ null
        /* Found a matching class object. */
        msgOut.rItems[i].pName := class!lDAPDisplayName
        msgOut.rItems[i].status := DS_NAME_ERROR_SCHEMA_GUID_CLASS
      else
        /* Finally, try to find a matching extendedRight object. */
        er := select one o from
            subtree DescendantObject(ConfigNC(),
                                     "CN=Extended-Rights,")
              where extendedRight in o!objectClass and
                o!rightsGuid = msgIn.rItems[i]
        if er ≠ null
          /* Found a matching extendedRight object */
          if RIGHT_DS_READ_PROPERTY in er!validAccesses or
              RIGHT_DS_WRITE_PROPERTY in er!validAccesses then
            msgOut.rItems[i].pName := er!displayName
            msgOut.rItems[i].status :=
                DS_NAME_ERROR_SCHEMA_GUID_ATTR_SET
          else if RIGHT_DS_CONTROL_ACCESS in er!validAccesses or
              RIGHT_DS_WRITE_PROPERTY_EXTENDED in er!validAccesses
              then
            msgOut.rItems[i].pName := er!displayName
            msgOut.rItems[i].status :=
                  DS_NAME_ERROR_SCHEMA_GUID_CONTROL_RIGHT
          endif
        endif
      endif
    endif
  endfor
  msgOut.cItems := msgIn.cNames
endif

pmsgOut^.V1.pResult := ADR(msgOut)
return ERROR_SUCCESS
```

4.1.3.4 Examples of the IDL_DRSCrackNames Method

When user "Kim Akers" logs on to the computer MS1.Contoso.com using her Windows NT 4.0 account name "CONTOSO\kimakers", the domain controller needs to obtain a fully qualified DN (FQDN) that corresponds to the Windows NT 4.0 account name. The domain controller DC1 calls IDL_DRSCrackNames to translate the Windows NT 4.0 account name to an FQDN.

4.1.3.4.1 Initial State

Querying the user object with name KimAkers in the domain NC DC=CONTOSO, DC=COM on DC1:

- ldap_search_s("CN=Kim Akers,CN=Users,DC=contoso,DC=com", *baseObject*, "(objectClass=user)", [*objectClass, distinguishedName, sAMAccountName*])
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=Kim Akers,CN=Users,DC=contoso,DC=com
  - 4> objectClass: top; person; organizationalPerson; user;
  - 1> distinguishedName: CN=Kim Akers,CN=Users,DC=contoso,DC=com;
  - 1> sAMAccountName: KimAkers

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

Expanding base 'CN=CONTOSO,CN=Partitions,CN=Configuration,DC=contoso,DC=com'...

Result <0>: (null)

Matched DNs:

Getting 1 entries:

>> Dn: CN=CONTOSO,CN=Partitions,CN=Configuration,DC=contoso,DC=com

- 2> objectClass: top; crossRef;
- 1> nCName: DC=contoso,DC=com;
- 1> dnsRoot: contoso.com;
- 1> nETBIOSName: CONTOSO;

4.1.3.4.2 Client Request

A client invokes the IDL_DRSCrackNames method against DC1 with the following parameters (**DRS_HANDLE** to DC1 omitted):

- dwInVersion = 1
- pmsgIn = DRS_MSG_CRACKREQ_V1
  - CodePage = 0x4e4
  - LocaleId = US-EN
  - dwFlags = DS_NAME_NO_FLAGS
  - formatOffered = DS_NT4_ACCOUNT_NAME
  - formatDesired = DS_FQDN_1779_NAME

- cNames: 1
- rpNames: "CONTOSO\kimakers"

4.1.3.4.3 Server Response

Returns code of 0 and the following values:

- pdwMessageOut = 1

  pmsgOut = DRS_MSG_CRACKREPLY_V1

  - pResult: DS_NAME_RESULTW
  - cNames: 1
  - rItems: DS_NAME_RESULT_ITEMW
  - pDomain: "contoso.com"
  - pName: "CN=Kim Akers,CN=Users,DC=contoso,DC=com"
  - status: DS_NAME_NO_ERROR

4.1.3.4.4 Final State

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

4.1.4 IDL_DRSDomainControllerInfo (Opnum 16)

The **IDL_DRSDomainControllerInfo** method retrieves information about DCs in a given domain.

```
ULONG IDL_DRSDomainControllerInfo(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)]
    DRS_MSG_DCINFOREQ* pmsgIn,
  [out, ref] DWORD* pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_DCINFOREPLY* pmsgOut
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

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

4.1.4.1 Method-Specific Concrete Types

4.1.4.1.1 DRS_MSG_DCINFOREQ

The **DRS_MSG_DCINFOREQ** union defines the request messages sent to the **IDL_DRSDomainControllerInfo** method. Only one version, identified by dwInVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.4.1.2 DRS_MSG_DCINFOREQ_V1

The **DRS_MSG_DCINFOREQ_V1** structure defines the request message sent to the **IDL_DRSDomainControllerInfo** method.

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

Domain: DNS domain name for which the client requests DC information.

InfoLevel: Response version requested by the client: 1, 2, 3, or 0xFFFFFFFF. The responses at InfoLevel 1, 2, and 3 all contain information about DCs in the given domain. The information at InfoLevel 1 is a subset of the information at InfoLevel 2, which is a subset of the information at InfoLevel 3. InfoLevel 3 includes information about the **RODCs** in the given domain. InfoLevel 0xFFFFFFFF server returns information about the active **LDAP connections**.

4.1.4.1.3 DRS_MSG_DCINFOREPLY

The **DRS_MSG_DCINFOREPLY** union defines the response messages received from the **IDL_DRSDomainControllerInfo** method.

```
typedef
[switch_type(DWORD)]
union {
  [case(1)]
    DRS_MSG_DCINFOREPLY_V1 V1;
  [case(2)]
    DRS_MSG_DCINFOREPLY_V2 V2;
  [case(3)]
    DRS_MSG_DCINFOREPLY_V3 V3;
```

```
  [case(0xFFFFFFFF)]
    DRS_MSG_DCINFOREPLY_VFFFFFFFF VFFFFFFFF;
} DRS_MSG_DCINFOREPLY;
```

V1: Version 1 response.

V2: Version 2 response.

V3: Version 3 response.

VFFFFFFFF: Version 0xFFFFFFFF response.

4.1.4.1.4 DRS_MSG_DCINFOREPLY_V1

The **DRS_MSG_DCINFOREPLY_V1** structure defines the response message received from the **IDL_DRSDomainControllerInfo** method, when the client has requested InfoLevel = 1.

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

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

rItems: DC information.

4.1.4.1.5 DRS_MSG_DCINFOREPLY_V2

The **DRS_MSG_DCINFOREPLY_V2** structure defines the response message received from the **IDL_DRSDomainControllerInfo** method, when the client has requested InfoLevel = 2.

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

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

rItems: DC information.

4.1.4.1.6 DRS_MSG_DCINFOREPLY_V3

The **DRS_MSG_DCINFOREPLY_V3** structure defines the response message received from the **IDL_DRSDomainControllerInfo** method when the client has requested InfoLevel = 3.

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

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

rItems: DC information.

4.1.4.1.7 DRS_MSG_DCINFOREPLY_VFFFFFFFF

The **DRS_MSG_DCINFOREPLY_VFFFFFFFF** structure defines the response message received from the **IDL_DRSDomainControllerInfo** method, when the client has requested InfoLevel = 0xFFFFFFFF.

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

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

rItems: DC information.

4.1.4.1.8 DS_DOMAIN_CONTROLLER_INFO_1W

The **DS_DOMAIN_CONTROLLER_INFO_1W** structure defines DC information that is returned as a part of the response to an InfoLevel = 1 request. The struct contains information about a single DC in the domain.

```
typedef struct {
  [string, unique] WCHAR* NetbiosName;
  [string, unique] WCHAR* DnsHostName;
  [string, unique] WCHAR* SiteName;
  [string, unique] WCHAR* ComputerObjectName;
  [string, unique] WCHAR* ServerObjectName;
  BOOL fIsPdc;
  BOOL fDsEnabled;
} DS_DOMAIN_CONTROLLER_INFO_1W;
```

NetbiosName: NetBIOS name of the DC.

DnsHostName: DNS host name of the DC.

SiteName: RDN of the site object.

ComputerObjectName: DN of the computer object that corresponds to the DC.

ServerObjectName: DN of the server object that corresponds to the DC.

fIsPdc: True if and only if the DC is the PDC FSMO role owner.

fDsEnabled: MUST be true.

4.1.4.1.9 DS_DOMAIN_CONTROLLER_INFO_2W

The **DS_DOMAIN_CONTROLLER_INFO_2W** structure defines DC information that is returned as a part of the response to an InfoLevel = 2 request. The struct contains information about a single DC in the domain.

```
typedef struct {
  [string, unique] WCHAR* NetbiosName;
  [string, unique] WCHAR* DnsHostName;
  [string, unique] WCHAR* SiteName;
  [string, unique] WCHAR* SiteObjectName;
  [string, unique] WCHAR* ComputerObjectName;
  [string, unique] WCHAR* ServerObjectName;
  [string, unique] WCHAR* NtdsDsaObjectName;
  BOOL fIsPdc;
  BOOL fDsEnabled;
  BOOL fIsGc;
  GUID SiteObjectGuid;
  GUID ComputerObjectGuid;
  GUID ServerObjectGuid;
  GUID NtdsDsaObjectGuid;
} DS_DOMAIN_CONTROLLER_INFO_2W;
```

NetbiosName: NetBIOS name of the DC.

DnsHostName: DNS host name of the DC.

SiteName: RDN of the site object.

SiteObjectName: DN of the site object.

ComputerObjectName: DN of the computer object that corresponds to the DC.

ServerObjectName: DN of the server object that corresponds to the DC.

NtdsDsaObjectName: DN of the nTDSDSA object that corresponds to the DC.

fIsPdc: True if and only if the DC is the PDC FSMO role owner.

fDsEnabled: MUST be true.

fIsGc: True if and only if the DC is also a GC.

SiteObjectGuid: objectGUID of the site object.

ComputerObjectGuid: objectGUID of the computer object that corresponds to the DC.

ServerObjectGuid: objectGUID of the server object that corresponds to the DC.

NtdsDsaObjectGuid: objectGUID of the nTDSDSA object that corresponds to the DC.

4.1.4.1.10 DS_DOMAIN_CONTROLLER_INFO_3W

The **DS_DOMAIN_CONTROLLER_INFO_3W** structure defines DC information that is returned as a part of the response to an InfoLevel = 3 request. The struct contains information about a single DC in the domain.

```
typedef struct {
  [string, unique] WCHAR* NetbiosName;
  [string, unique] WCHAR* DnsHostName;
  [string, unique] WCHAR* SiteName;
  [string, unique] WCHAR* SiteObjectName;
  [string, unique] WCHAR* ComputerObjectName;
  [string, unique] WCHAR* ServerObjectName;
  [string, unique] WCHAR* NtdsDsaObjectName;
  BOOL fIsPdc;
  BOOL fDsEnabled;
  BOOL fIsGc;
  BOOL fIsRodc;
  GUID SiteObjectGuid;
  GUID ComputerObjectGuid;
  GUID ServerObjectGuid;
  GUID NtdsDsaObjectGuid;
} DS_DOMAIN_CONTROLLER_INFO_3W;
```

NetbiosName: NetBIOS name of the DC.

DnsHostName: DNS host name of the DC.

SiteName: RDN of the site object.

SiteObjectName: DN of the site object.

ComputerObjectName: DN of the computer object that corresponds to the DC.

ServerObjectName: DN of the server object that corresponds to the DC.

NtdsDsaObjectName: DN of the nTDSDSA object that corresponds to the DC.

fIsPdc: True if and only if the DC is the PDC FSMO role owner.

fDsEnabled: MUST be true.

fIsGc: True if and only if the DC is also a GC.

fIsRodc: True if and only if the DC is an RODC.

SiteObjectGuid: objectGUID of the site object.

ComputerObjectGuid: objectGUID of the computer object that corresponds to the DC.

ServerObjectGuid: objectGUID of the server object that corresponds to the DC.

NtdsDsaObjectGuid: objectGUID of the nTDSDSA object that corresponds to the DC.

4.1.4.1.11 DS_DOMAIN_CONTROLLER_INFO_FFFFFFFFW

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

```
typedef struct {
  DWORD IPAddress;
```

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

IPAddress: IPv4 address of the client that established the LDAP connection to the server. If the client connected with IPv6, this field contains zero.

NotificationCount: Number of LDAP notifications enabled on the server.

secTimeConnected: Total time in number of seconds that the connection is established.

Flags: Zero or more of the bit flags from LDAP_CONN_PROPERTIES indicating the properties of this connection.

TotalRequests: Total number of LDAP requests made on this LDAP connection.

Reserved1: MUST be 0.

UserName: Name of the security principal that established the LDAP connection.

4.1.4.2 Server Behavior of the IDL_DRSDomainControllerInfo Method

Informative summary of behavior: This method supports four information levels. For levels 1, 2, and 3, the server returns information for the DCs in the domain of the server. For level 0xffffffff, the server returns information about the currently opened LDAP connections on the server.

Regular read access checks apply to the information that is returned to the caller. Therefore, if the caller does not have **read permission** on data that needs to be returned, this data is not included in the response. See [MS-ADTS] section 3.1.1.4.3 for more information about access check behavior in read operations.

```
ULONG
IDL_DRSDomainControllerInfo(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)] DRS_MSG_DCINFOREQ *pmsgIn,
  [out, ref] DWORD *pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)] DRS_MSG_DCINFOREPLY *pmsgOut)

msgIn: DRS_MSG_DCINFOREQ_V1
infoLevel, i: integer
domainName: unicodestring
dcSet: set of DSName
crObj, dc, dsaObj, svrObj, siteObj, obj, v: DSName
lc: DS_DOMAIN_CONTROLLER_INFO_FFFFFFFFW
rI1: ADDRESS OF DS_DOMAIN_CONTROLLER_INFO_1W
rI2: ADDRESS OF DS_DOMAIN_CONTROLLER_INFO_2W
rI3: ADDRESS OF DS_DOMAIN_CONTROLLER_INFO_3W
found: boolean
crackMsgIn: DRS_MSG_CRACKREQ
crackMsgOut: DRS_MSG_CRACKREPLY
```

```
outV: DWORD
userAccountControl: set of integer

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

if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif

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

pdwOutVersion^ := infoLevel

if infoLevel = 0xFFFFFFFF then
  /* Enumerate the LDAP connections. */
  if not IsMemberOfBuiltinAdminGroup() then
    return ERROR_ACCESS_DENIED
  endif

  pmsgOut^.VFFFFFFFF.cItems := number(dc.ldapConnections)

  i := 0
  foreach lc in dc.ldapConnections
    pmsgOut^.VFFFFFFFF.rItems[i].IPAddress := lc.iPAddress
    pmsgOut^.VFFFFFFFF.rItems[i].NotificationCount :=
        lc.notificationCount
    pmsgOut^.VFFFFFFFF.rItems[i].secTimeConnected :=
        lc.secTimeConnected
    pmsgOut^.VFFFFFFFF.rItems[i].Flags := lc.flags
    pmsgOut^.VFFFFFFFF.rItems[i].TotalRequests := lc.totalRequests
    pmsgOut^.VFFFFFFFF.rItems[i].UserName := lc.userName
    pmsgOut^.VFFFFFFFF.rItems[i].Reserved1 := 0

    i := i + 1
  endfor

  return 0
endif

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

crObj := select one v from children
    DescendantObject(ConfigNC(), "CN=Partitions,")
    where
      (v!dNSHostName = domainName or v!nETBiosName = domainName)
      and
      v!nCName = DefaultNC()

found := (crObj ≠ null)
```

```
if not found then
  /* Not found; use IDL_DRSCrackNames to resolve the name. */
  crackMsgIn.V1.formatOffered := DS_UNKNOWN_NAME
  crackMsgIn.V1.formatDesired := DS_FQDN_1779_NAME
  crackMsgIn.V1.cNames := 3
  crackMsgIn.V1.rpNames[0] := domainName
  crackMsgIn.V1.rpNames[1] := domainName + "\"
  crackMsgIn.V1.rpNames[2] := domainName + "/"

  /* Call IDL_DRSCrackNames as a local procedure. */
  IDL_DRSCrackNames(null, 1, crackMsgIn, ADR(outV), ADR(crackMsgOut))

  i := 0
  while i < 3 and not found
    if crackMsgOut.V1.pResult.rItems[i].status = DS_NAME_NO_ERROR
        then
      if crackMsgOut.V1.pResult.rItems[i].pName = DefaultNC().dn
          then
        found := true
      else
        return ERROR_INVALID_PARAMETER
      endif
    endif
    i := i + 1
  endwhile
endif

if not found then
  return ERROR_DS_OBJ_NOT_FOUND
endif

/* Enumerate the DCs in the domain. */
if infoLevel = 3 then
  /* client requests to return RODCs too */
  userAccountControl :=
      {ADS_UF_SERVER_TRUST_ACCOUNT, ADS_UF_PARTIAL_SECRETS_ACCOUNT}
else
  userAccountControl := {ADS_UF_SERVER_TRUST_ACCOUNT}
endif

dcSet := select all v from subtree DefaultNC() where
     v!objectCategory = GetDefaultObjectCategory(computer)
       and (userAccountControl ∩ v!userAccountControl ≠ null)

if infoLevel = 1 then
  pmsgOut^.V1.cItems := number(dcSet)

  i := 0
  foreach dc in dcSet
    rI1 := ADR(pmsgOut^.V1.rItems[i])

    rI1^.DnsHostName := dc!dNSHostName
    rI1^.ComputerObjectName := dc.dn
    /* sAMAccountName excluding the "$" at the end. */
    rI1^.NetbiosName := SubString(dc!sAMAccountName, 0,
        dc!samAccountName.length-1)
    rI1^.fDsEnabled := true
```

```
      /* select a server object from the serverReferenceBL, it is
         preferred that the server object has a child object with
         CN "NTDS Settigs" */
      srvObj :=
          select one v from all where v.dn in dc!serverReferenceBL
                and DescendentObject(v, "CN=NTDS Settings") ≠ null
      if srvObj = null then
          srvObj :=
              select one v from all where v.dn in dc!serverReferenceBL
      endif
      if svrObj ≠ null then
        rI1^.ServerObjectName := svrObj.dn
        siteObj :=
            select one o from all where o!objectGUID = svrObj!parent
        rI1^.SiteObjectName := siteObj.dn
        dsaObj := DescendantObject(v, "CN=NTDS Settings,")
        rI1^.fIsPdc := (dsaObj = GetFSMORoleOwner(FSMO_PDC))
      endif
      i := i + 1
    endfor
  else
    if infoLevel = 2 then
      pmsgOut^.V2.cItems := number(dcSet)

      i := 0
      foreach dc in dcSet
        rI2 := ADR(pmsgOut^.V2.rItems[i])

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

        /* select a server object from the serverReferenceBL, it is
           preferred that the server object has a child object with
           CN "NTDS Settigs" */
        srvObj :=
            select one v from all where v.dn in dc!serverReferenceBL
                  and DescendentObject(v, "CN=NTDS Settings") ≠ null
        if srvObj = null then
            srvObj :=
                select one v from all where v.dn in dc!serverReferenceBL
        endif
        if svrObj ≠ null then
          rI2^.ServerObjectName := svrObj.dn
          rI2^.ServerObjectGuid := svrObj.guid
          siteObj :=
            select one o from all where o!objectGUID = svrObj!parent
          rI2^.SiteObjectName := siteObj.dn
          rI2^.SiteObjectGUID := siteObj.guid
          dsaObj := DescendantObject(v, "CN=NTDS Settings,")
          rI2^.NtdsDsaObjectGUID := dsaObj.guid
          rI2^.fIsGc := (NTDSDSA_OPT_IS_GC in dsaObj!options)
          rI2^.fIsPdc := (dsaObj = GetFSMORoleOwner(FSMO_PDC))
        endif
```

```
      i := i + 1
    endfor

  else
    /* infoLevel = 3 */
    pmsgOut^.V2.cItems := number(dcSet)

    i := 0
    foreach dc in dcSet
      rI3 := ADR(pmsgOut^.V2.rItems[i])

      rI3^.DnsHostName := dc!dNSHostName
      rI3^.ComputerObjectName := dc.dn
      /* sAMAccountName excluding the "$" at the end. */
      rI3^.NetbiosName := SubString(dc!samAccountName, 0,
          dc!samAccountName.length-1)
      rI3^.ComputerObjectGUID := dc.guid
      rI3^.fDsEnabled := true

      /* select a server object from the serverReferenceBL, it is
         preferred that the server object has a child object with
         CN "NTDS Settigs" */
      srvObj :=
          select one v from all where v.dn in dc!serverReferenceBL
                and DescendentObject(v, "CN=NTDS Settings") ≠ null
      if srvObj = null then
          srvObj :=
              select one v from all where v.dn in dc!serverReferenceBL
      endif
      if svrObj ≠ null then
        rI3^.ServerObjectName := svrObj.dn
        rI3^.ServerObjectGuid := svrObj.guid
        siteObj :=
            select one o from all where o!objectGUID = svrObj!parent
        rI3^.SiteObjectName := siteObj.dn
        rI3^.SiteObjectGUID := siteObj.guid
        dsaObj := DescendantObject(v, "CN=NTDS Settings,")
        rI3^.NtdsDsaObjectGUID := dsaObj.guid
        rI3^.fIsGC := (NTDSDSA_OPT_IS_GC in dsaObj!options)
        rI3^.fIsPDC := (dsaObj = GetFSMORoleOwner(FSMO_PDC))
        Ri3^.fIsRodc := ((ADS_UF_PARTIAL_SECRETS_ACCOUNT ∩
            dc!userAccountControl) ≠ null)
      endif
      i := i + 1
    endfor
  endif
endif

return 0
```

4.1.4.3 Examples of the IDL_DRSDomainControllerInfo Method

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

4.1.4.3.1 Initial State

Querying the crossRef object on the NC root object for domain NC CONTOSO.COM on DC2:

- Expanding base 'CN=CONTOSO,CN=Partitions,CN=Configuration,DC=contoso,DC=com'...
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=CONTOSO,CN=Partitions,CN=Configuration,DC=contoso,DC=com
  - 2> objectClass: top; crossRef;
  - 1> nCName: DC=contoso,DC=com;
  - 1> dnsRoot: contoso.com;
  - 1> nETBIOSName: CONTOSO;

Querying the DC1 computer object in domain NC DC=CONTOSO, DC=COM

- Expanding base 'CN=DC1,OU=Domain Controllers,DC=contoso,DC=com'...
- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=DC1,OU=Domain Controllers,DC=contoso,DC=com
  - 5> objectClass: top; person; organizationalPerson; user; computer;
  - 1> cn: DC1;
  - 1> distinguishedName: CN=DC1, OU=Domain Controllers, DC=contoso, DC=com;
  - 1> instanceType: 0x4 = (IT_WRITE);
  - 1> whenCreated: 07/10/2006 18:04:35 Pacific Standard Daylight Time;
  - 1> whenChanged: 07/15/2006 19:39:05 Pacific Standard Daylight Time;
  - 1> uSNCreated: 12291;
  - 1> uSNChanged: 24577;
  - 1> name: DC1;
  - 1> objectGUID: ac1993e1-0377-4161-893e-ccd2a98e1bba;
  - 1> userAccountControl: (UF_SERVER_TRUST_ACCOUNT | UF_TRUSTED_FOR_DELEGATION);
  - 1> badPwdCount: 0;
  - 1> codePage: 0;

- 1> countryCode: 0;
- 1> badPasswordTime: 01/01/1601 00:00:00 UNC ;
- 1> lastLogoff: 01/01/1601 00:00:00 UNC ;
- 1> lastLogon: 07/17/2006 19:47:40 Pacific Standard Daylight Time;
- 1> localPolicyFlags: 0;
- 1> pwdLastSet: 07/10/2006 18:04:35 Pacific Standard Daylight Time;
- 1> primaryGroupID: 516;
- 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1001;
- 1> accountExpires: 09/14/30828 02:48:05 UNC ;
- 1> logonCount: 17;
- 1> sAMAccountName: DC1$;
- 1> sAMAccountType: 805306369;
- 1> operatingSystem: Windows Server 2003;
- 1> operatingSystemVersion: 5.2 (3790);
- 1> operatingSystemServicePack: Service Pack 1;
- 1> serverReferenceBL: CN=DC1,CN=Servers, CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com;
- 1> dNSHostName: DC1.contoso.com;
- 1> rIDSetReferences: CN=RID Set,CN=DC1,OU=Domain Controllers, DC=contoso, DC=com;
- 15> servicePrincipalName: ldap/DC1.contoso.com/NDNC5.contoso.com; ldap/DC1.contoso.com/NDNC2.contoso.com; ldap/DC1.contoso.com/NDNC1.contoso.com; GC/DC1.contoso.com/contoso.com; HOST/DC1.contoso.com/CONTOSO; HOST/DC1; HOST/DC1.contoso.com; HOST/DC1.contoso.com/contoso.com; E3514235-4B06-11D1-AB04-00C04FC2DCD2/c20bc312-4d35-4cc0-9903-b1073368af4a/contoso.com; ldap/c20bc312-4d35-4cc0-9903-b1073368af4a._msdcs.contoso.com; ldap/DC1.contoso.com/CONTOSO; ldap/DC1; ldap/DC1.contoso.com; ldap/DC1.contoso.com/contoso.com; NtFrs-88f5d2bd-b646-11d2-a6d3-00c04fc9b232/DC1.contoso.com;
- 1> objectCategory: CN=Computer, CN=Schema, CN=Configuration, DC=contoso, DC=com;
- 1> isCriticalSystemObject: TRUE;
- 1> frsComputerReferenceBL: CN=DC1, CN=Domain System Volume (SYSVOL share),CN=File Replication Service,CN=System,DC=contoso,DC=com;
- 1> lastLogonTimestamp: 07/11/2006 04:02:42 Pacific Std Daylight Time;

Querying the DC2 computer object in domain NC DC=CONTOSO, DC=COM

- Expanding base 'CN=DC2,OU=Domain Controllers,DC=contoso,DC=com'...

- Result <0>: (null)
- Matched DNs:
- Getting 1 entries:
- >> Dn: CN=DC2,OU=Domain Controllers,DC=contoso,DC=com
  - 5> objectClass: top; person; organizationalPerson; user; computer;
  - 1> cn: DC2;
  - 1> distinguishedName: CN=DC2, OU=Domain Controllers, DC=contoso, DC=com;
  - 1> instanceType: 0x4 = (IT_WRITE);
  - 1> whenCreated: 07/10/2006 18:12:01 Pacific Standard Daylight Time;
  - 1> whenChanged: 07/16/2006 13:46:14 Pacific Standard Daylight Time;
  - 1> displayName: DC2$;
  - 1> uSNCreated: 13711;
  - 1> uSNChanged: 28819;
  - 1> name: DC2;
  - 1> objectGUID: 09697f46-2458-4b26-a4e9-aa36059421c4;
  - 1> userAccountControl: (UF_SERVER_TRUST_ACCOUNT | UF_TRUSTED_FOR_DELEGATION);
  - 1> badPwdCount: 0;
  - 1> codePage: 0;
  - 1> countryCode: 0;
  - 1> badPasswordTime: 01/01/1601 00:00:00 UNC ;
  - 1> lastLogoff: 01/01/1601 00:00:00 UNC ;
  - 1> lastLogon: 07/17/2006 20:38:08 Pacific Standard Daylight Time;
  - 1> localPolicyFlags: 0;
  - 1> pwdLastSet: 07/10/2006 18:12:02 Pacific Standard Daylight Time;
  - 1> primaryGroupID: 516;
  - 1> objectSid: S-1-5-21-254470460-2440132622-709970653-1102;
  - 1> accountExpires: 09/14/30828 02:48:05 UNC ;
  - 1> logonCount: 8;
  - 1> sAMAccountName: DC2$;
  - 1> sAMAccountType: 805306369;

- 1> operatingSystem: Windows Server 2003;
- 1> operatingSystemVersion: 5.2 (3790);
- 1> operatingSystemServicePack: Service Pack 1;
- 1> serverReferenceBL: CN=DC2,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=contoso,DC=com;
- 1> dNSHostName: DC2.contoso.com;
- 1> rIDSetReferences: CN=RID Set,CN=DC2,OU=Domain Controllers, DC=contoso, DC=com;
- 14> servicePrincipalName: ldap/DC2.contoso.com/NDNC5.contoso.com; ldap/DC2.contoso.com/NDNC2.contoso.com; ldap/6aad8f5a-07cc-403a-9696-9102fe1c320b._msdcs.contoso.com; ldap/DC2.contoso.com/CONTOSO; ldap/DC2; ldap/DC2.contoso.com; ldap/DC2.contoso.com/contoso.com; NtFrs-88f5d2bd-b646-11d2-a6d3-00c04fc9b232/DC2.contoso.com; HOST/DC2.contoso.com/CONTOSO; HOST/DC2.contoso.com/contoso.com; C/DC2.contoso.com/contoso.com; E3514235-4B06-11D1-AB04-00C04FC2DCD2/6aad8f5a-07cc-403a-9696-9102fe1c320b/contoso.com;
- HOST/DC2; HOST/DC2.contoso.com;
- 1> objectCategory: CN=Computer, CN=Schema, CN=Configuration, DC=contoso, DC=com;
- 1> isCriticalSystemObject: TRUE;
- 1> frsComputerReferenceBL: CN=DC2,CN=Domain System Volume (SYSVOL share),CN=File Replication Service,CN=System,DC=contoso,DC=com;
- 4> dSCorePropagationData: 07/10/2006 18:14:51 Pacific Standard Daylight Time; 07/10/2006 18:14:51 Pacific Standard Time Pacific Daylight Time; 07/10/2006 18:14:51 Pacific Standard Time Pacific Daylight Time; 01/08/1601 07:15:13 Pacific Standard Time Pacific Daylight Time;
- 1> lastLogonTimestamp: 07/10/2006 19:52:48 Pacific Std Daylight Time;

4.1.4.3.2 Client Request

A client invokes the **IDL_DRSDomainControllerInfo** method against DC2 with the following parameters (**DRS_HANDLE** to DC2 omitted):

- dwInVersion = 1
- pmsgIn = **DRS_MSG_DCINFOREQ_V1**
  - Domain = "contoso.com"
  - InfoLevel = 1

4.1.4.3.3 Server Response

Return code of 0 and the following values:

- pdwOutVersion^ = 1
- pmsgOut = **DRS_MSG_DCINFOREPLY_V1**

- cItems: 2
- rItems[0]: **DS_DOMAIN_CONTROLLER_INFO_1W**
  - NetbiosName: "DC1"
  - DnsHostName: "DC1.contoso.com"
  - SiteName: "Default-First-Site-Name"
  - ComputerObjectName: "CN=DC1, OU=Domain Controllers,DC=contoso,DC=com"
  - ServerObjectName: "CN=DC1,CN=Servers, CN=Default-First-Site-Name,CN=Sites, CN=Configuration, DC=contoso,DC=com"
  - fIsPdc: 1
  - fIsDisabled: 1
- rItems[1]: DS_DOMAIN_CONTROLLER_INFO_1W
  - NetbiosName: "DC2"
  - DnsHostName: "DC2.contoso.com"
  - SiteName: "Default-First-Site-Name"
  - ComputerObjectName: "CN=DC2, OU=Domain Controllers,DC=contoso,DC=com"
  - ServerObjectName: "CN=DC2,CN=Servers, CN=Default-First-Site-Name,CN=Sites, CN=Configuration,
  - DC=contoso,DC=com"
  - fIsPdc: 0
  - fIsDisabled: 1

4.1.4.3.4 Final State

Final state is same as initial state; there is no change.

4.1.5 IDL_DRSExecuteKCC (Opnum 18)

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

```
ULONG IDL_DRSExecuteKCC(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)]
    DRS_MSG_KCC_EXECUTE* pmsgIn
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

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

4.1.5.1 Method-Specific Concrete Types

4.1.5.1.1 DRS_MSG_KCC_EXECUTE

The **DRS_MSG_KCC_EXECUTE** union defines the request messages sent to the **IDL_DRSExecuteKCC** method. Only one version, identified by dwInVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.5.1.2 DRS_MSG_KCC_EXECUTE_V1

The **DRS_MSG_KCC_EXECUTE_V1** structure defines the request message sent to the **IDL_DRSExecuteKCC** method.

```
typedef struct {
  DWORD dwTaskID;
  DWORD dwFlags;
} DRS_MSG_KCC_EXECUTE_V1;
```

dwTaskID: MUST be 0.

dwFlags: Zero or more of the following bit flags:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
AS	DP	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X: Unused. MUST be zero and ignored.

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

DP (DS_KCC_FLAG_DAMPED): Request the KCC to run unless there is already such a request pending.

4.1.5.2 Method-Specific Abstract Types and Procedures

4.1.5.2.1 ExecuteKCCTasks

```
procedure ExecuteKCCTasks(): ULONG
```

Executes the tasks necessary for maintaining the replication topology between DCs.

If an error occurs, a Windows error code is returned. If successful, the method returns 0.

4.1.5.3 Server Behavior of the IDL_DRSExecuteKCC Method

Informative summary of behavior: This method triggers the implemented process that generates and maintains the replication topology between DCs.<9>

```
ULONG
IDL_DRSExecuteKCC(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_KCC_EXECUTE *pmsgIn)

msgIn: DRS_MSG_KCC_EXECUTE_V1

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

msgIn := pmsgIn^.V1

if msgIn.dwTaskID ≠ 0 then
  return ERROR_INVALID_PARAMETER
endif

if not AccessCheckCAR(ConfigNC(), DS-Replication-Manage-Topology)
    then
  return ERROR_DS_DRA_ACCESS_DENIED
endif

return ExecuteKCCTasks()
```

4.1.6 IDL_DRSFinishDemotion (Opnum 27)

The **IDL_DRSFinishDemotion** method either performs one or more steps toward the complete removal of a DC from an AD/LDS forest, or undoes the effects of the first phase of removal (performed by **IDL_DRSInitDemotion**). This method is supported by AD/LDS only. This method is used only to diagnose, monitor, and manage the implementation of server-to-server DC demotion. The structures requested and returned through this method MAY have meaning to peer DCs and applications, but are not required for interoperation with Windows clients.

```
ULONG IDL_DRSFinishDemotion(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
```

```
  [in, ref, switch_is(dwInVersion)]
    DRS_MSG_FINISH_DEMOTIONREQ* pmsgIn,
  [out, ref] DWORD* pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_FINISH_DEMOTIONREPLY* pmsgOut
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

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

4.1.6.1 Method-Specific Concrete Types

4.1.6.1.1 DRS_MSG_FINISH_DEMOTIONREQ

The **DRS_MSG_FINISH_DEMOTIONREQ** union defines the request messages sent to the **IDL_DRSFinishDemotion** method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
  [case(1)]
    DRS_MSG_FINISH_DEMOTIONREQ_V1 V1;
} DRS_MSG_FINISH_DEMOTIONREQ;
```

V1: Version 1 request. Currently only one version is defined.

4.1.6.1.2 DRS_MSG_FINISH_DEMOTIONREQ_V1

The **DRS_MSG_FINISH_DEMOTIONREQ_V1** structure defines the request message sent to the **IDL_DRSFinishDemotion** method.

```
typedef struct {
  DWORD dwOperations;
  UUID uuidHelperDest;
  LPWSTR szScriptBase;
} DRS_MSG_FINISH_DEMOTIONREQ_V1;
```

dwOperations: Zero or more of the following bit flags:

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

X: Unused. MUST be zero and ignored.

R (DS_DEMOTE_ROLLBACK_DEMOTE): Undo the effects of **IDL_DRSInitDemotion**. If present, any other flags present are ignored.

C (DS_DEMOTE_COMMIT_DEMOTE): Mark the DC's database "demotion-complete" (this effect is outside the state model).

D (DS_DEMOTE_DELETE_CSMETA): Delete the nTDSDSA object for this DC; see RemoveADLDSServer below.

U1 (DS_DEMOTE_UNREGISTER_SCPS): Delete any serviceConnectionPoint objects for this DC from AD/DS; see RemoveADLDSSCP below

U2 (DS_DEMOTE_UNREGISTER_SPNS): Delete any AD/LDS SPNs from the service account object in AD/DS; see RemoveADLDSSPNs below.

F (DS_DEMOTE_OPT_FAIL_ON_UNKNOWN_OP): Fail the request if the dwOperations field contains an unknown flag.

uuidHelperDest: MUST be **NULL GUID**.

szScriptBase: The path name of the folder to store SPN unregistration scripts. Required when DS_DEMOTE_UNREGISTER_SPNS is specified in **dwOperations**.

4.1.6.1.3 DRS_MSG_FINISH_DEMOTIONREPLY

The **DRS_MSG_FINISH_DEMOTIONREPLY** union defines the response messages received from the **IDL_DRSFinishDemotion** method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

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

V1: Version 1 reply.

4.1.6.1.4 DRS_MSG_FINISH_DEMOTIONREPLY_V1

The **DRS_MSG_FINISH_DEMOTIONREPLY_V1** structure defines the response message received from the **IDL_DRSFinishDemotion** method.

```
typedef struct {
```

```
  DWORD dwOperationsDone;
  DWORD dwOpFailed;
  DWORD dwOpError;
} DRS_MSG_FINISH_DEMOTIONREPLY_V1;
```

dwOperationsDone: The set of operations that were successfully performed. This may include the following values: DS_DEMOTE_ROLLBACK_DEMOTE, DS_DEMOTE_COMMIT_DEMOTE, DS_DEMOTE_DELETE_CSMETA, DS_DEMOTE_UNREGISTER_SCPS, DS_DEMOTE_UNREGISTER_SPNS.

dwOpFailed: The set of operations that failed during demotion. This may include the same values as the **dwOperationsDone** field.

dwOpError: The Win32 error code (as specified in [MS-ERREF] section 3.0) of the first failed operation (if any).

4.1.6.2 Method-Specific Abstract Types and Procedures

4.1.6.2.1 RemoveADLDSServer

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

Procedure RemoveADLDSServer() connects to any available replication partner and deletes the nTDSDSA object corresponding to this DC using the **IDL_DRSRemoveDsServer** method. If no such nTDSDSA object exists or the deletion is successful, RemoveADLDSServer returns ERROR_SUCCESS, otherwise it returns a Win32 error.

4.1.6.2.2 RemoveADLDSSCP

```
procedure RemoveADLDSSCP(): DWORD
```

Procedure RemoveADLDSSCP connects to an AD/DS DC and deletes any serviceConnectionPoint object that was created in AD/DS for this AD/LDS DC. See [MS-ADTS] section 7.3.8 for more information on AD/LDS serviceConnectionPoint objects. If no such serviceConnectionPoint object exists or the deletion is successful, RemoveADLDSSCP returns ERROR_SUCCESS, otherwise it returns a Win32 error.

4.1.6.2.3 RemoveADLDSSPNs

```
procedure RemoveADLDSSPNs(szScriptBase: unicodestring): DWORD
```

Procedure RemoveADLDSSPNs connects to an AD/DS DC and deletes any SPN values registered for the AD/LDS DC on its service account object in AD/DS. Section 2.2.3.2 specifies the SPN values removed by this procedure. If no such SPN values exist or the deletion is successful, RemoveADLDSSPNs returns ERROR_SUCCESS, otherwise it returns a Win32 error and writes a batch file in the folder specified by szScriptBase parameter. This batch file contains commands that an administrator can run to clean up the SPNs.

4.1.6.3 Server Behavior of the IDL_DRSFinishDemotion Method

Informative summary of behavior: Either performs one or more steps toward the complete removal of a DC from an AD/LDS forest, or undoes the effects of the first phase of removal (performed by **IDL_DRSInitDemotion**).

```
ULONG
IDL_DRSFinishDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_FINISH_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_FINISH_DEMOTIONREPLY* pmsgOut
    )

msgIn: DRS_MSG_FINISH_DEMOTIONREQ_V1
msgOut: DRS_MSG_FINISH_DEMOTIONREPLY_V1
ret: DWORD

if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
if DS_DEMOTE_OPT_FAIL_ON_UNKNOWN_OP in msgIn.dwOperations
    and msgIn.dwOperations - { DS_DEMOTE_COMMIT_DEMOTE,
      DS_DEMOTE_DELETE_CSMETA, DS_DEMOTE_UNREGISTER_SCPS,
      DS_DEMOTE_UNREGISTER_SPNS, DS_DEMOTE_OPT_FAIL_ON_UNKNOWN_OP }
    ≠ null then
  /* unknown operation bit is set */
  return ERROR_INVALID_PARAMETER
endif
if DS_DEMOTE_UNREGISTER_SPNS in msgIn.dwOperations
    and msgIn.szScriptBase = null then
  /* szScriptBase must be specified when UNREGISTER_SPN is
   * requested */
  return ERROR_INVALID_PARAMETER
endif
if not IsMemberOfBuiltinAdminGroup() then
  /* only BA is allowed to demote an AD/LDS service */
  return ERROR_ACCESS_DENIED
endif

msgOut.dwOperationDone := 0
msgOut.dwOpFailed := 0
msgOut.dwOpError := ERROR_SUCCESS

if DS_DEMOTE_ROLLBACK_DEMOTE in msgIn.dwOperations then
  /* undo the effects of IDL_DRSInitDemotion */
  ret := Mark database as read-write
  if ret = ERROR_SUCCESS then
    ret := Enable originating and replicated updates
  endif
  if ret = ERROR_SUCCESS then
    msgOut.dwOperationDone :=
      msgOut.dwOperationDone + {DS_DEMOTE_ROLLBACK_DEMOTE}
  else
```

```
      msgOut.dwOpFailed =
        msgOut.dwOpFailed + {DS_DEMOTE_ROLLBACK_DEMOTE}
      if msgOut.dwOpError = ERROR_SUCCESS then
        msgOut.dwOpError := ret
      endif
    endif
    /* no other operations are allowed on rollback */
  else
    if DS_DEMOTE_COMMIT_DEMOTE in msgIn.dwOperations then
      ret := Mark database as demotion-complete
      if ret = ERROR_SUCCESS then
        msgOut.dwOperationDone :=
          msgOut.dwOperationDone + {DS_DEMOTE_COMMIT_DEMOTE}
      else
        msgOut.dwOpFailed =
          msgOut.dwOpFailed + {DS_DEMOTE_COMMIT_DEMOTE}
        if msgOut.dwOpError = ERROR_SUCCESS then
          msgOut.dwOpError := ret
        endif
      endif
    endif
    if DS_DEMOTE_DELETE_CSMETA in msgIn.dwOperations then
      ret := RemoveADLDSServer()
      if ret = ERROR_SUCCESS then
        msgOut.dwOperationDone :=
          msgOut.dwOperationDone + {DS_DEMOTE_DELETE_CSMETA}
      else
        msgOut.dwOpFailed =
          msgOut.dwOpFailed + {DS_DEMOTE_DELETE_CSMETA}
        if msgOut.dwOpError = ERROR_SUCCESS then
          msgOut.dwOpError := ret
        endif
      endif
    endif
    if DS_DEMOTE_UNREGISTER_SCPS in msgIn.dwOperations then
      ret := RemoveADLDSSCP()
      if ret = ERROR_SUCCESS then
        msgOut.dwOperationDone :=
          msgOut.dwOperationDone + {DS_DEMOTE_UNREGISTER_SCPS}
      else
        msgOut.dwOpFailed =
          msgOut.dwOpFailed + {DS_DEMOTE_UNREGISTER_SCPS}
        if msgOut.dwOpError = ERROR_SUCCESS then
          msgOut.dwOpError := ret
        endif
      endif
    endif
    if DS_DEMOTE_UNREGISTER_SPNS in msgIn.dwOperations then
      ret := RemoveADLDSSPNs(msgIn.szScriptBase)
      if ret = ERROR_SUCCESS then
        msgOut.dwOperationDone :=
          msgOut.dwOperationDone + {DS_DEMOTE_UNREGISTER_SPNS}
      else
        msgOut.dwOpFailed =
          msgOut.dwOpFailed + {DS_DEMOTE_UNREGISTER_SPNS}
        if msgOut.dwOpError = ERROR_SUCCESS then
          msgOut.dwOpError := ret
        endif
```

```
    endif
  endif
endif
pmsgOut^ := msgOut
pdwMsgOut^ := 1
return ERROR_SUCCESS
```

4.1.7 IDL_DRSGetReplInfo (Opnum 19)

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

```
ULONG IDL_DRSGetReplInfo(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)]
    DRS_MSG_GETREPLINFO_REQ* pmsgIn,
  [out, ref] DWORD* pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_GETREPLINFO_REPLY* pmsgOut
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

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

4.1.7.1 Method-Specific Concrete Types

4.1.7.1.1 DRS_MSG_GETREPLINFO_REQ

The **DRS_MSG_GETREPLINFO_REQ** union defines the request message versions sent to the **IDL_DRSGetReplInfo** method.

```
typedef
[switch_type(DWORD)]
union {
  [case(1)]
    DRS_MSG_GETREPLINFO_REQ_V1 V1;
  [case(2)]
    DRS_MSG_GETREPLINFO_REQ_V2 V2;
} DRS_MSG_GETREPLINFO_REQ;
```

V1: Version 1 request (Windows 2000 and later).

V2: Version 2 request (Windows Server 2003 and later). The V2 request structure is a superset of the V1 request structure.

4.1.7.1.2 DRS_MSG_GETREPLINFO_REQ_V1

The **DRS_MSG_GETREPLINFO_REQ_V1** structure defines a version 1 request message sent to the **IDL_DRSGetReplInfo** method. This request version is supported by Windows 2000 and later releases of Windows.

```
typedef struct {
  DWORD InfoType;
  [string] LPWSTR pszObjectDN;
  UUID uuidSourceDsaObjGuid;
} DRS_MSG_GETREPLINFO_REQ_V1;
```

InfoType: MUST be a DS_REPL_INFO code.

pszObjectDN: DN of the object on which the operation should be performed. The meaning of this parameter depends on the value of the **InfoType** parameter.

uuidSourceDsaObjGuid: NULL GUID or the **DSA GUID** of a DC.

4.1.7.1.3 DRS_MSG_GETREPLINFO_REQ_V2

The **DRS_MSG_GETREPLINFO_REQ_V2** structure defines a version 2 request message sent to the **IDL_DRSGetReplInfo** method. The V2 request structure is a superset of the V1 request structure, and is supported by Windows Server 2003 and later versions of Windows.

```
typedef struct {
  DWORD InfoType;
  [string] LPWSTR pszObjectDN;
  UUID uuidSourceDsaObjGuid;
  DWORD ulFlags;
  [string] LPWSTR pszAttributeName;
  [string] LPWSTR pszValueDN;
  DWORD dwEnumerationContext;
} DRS_MSG_GETREPLINFO_REQ_V2;
```

InfoType: MUST be a DS_REPL_INFO code.

pszObjectDN: DN of the object on which the operation should be performed. The meaning of this parameter depends on the value of the **InfoType** parameter.

uuidSourceDsaObjGuid: NULL GUID or the DSA GUID of a DC.

ulFlags: Zero or more of the following bit flags:

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

X: Unused. MUST be zero and ignored.

MT (DS_REPL_INFO_FLAG_IMPROVE_LINKED_ATTRS): Return attribute stamps for linked values.

pszAttributeName: Null, or the lDAPDisplayName of a link attribute.

pszValueDN: Null, or the DN of the link value for which to retrieve a stamp.

dwEnumerationContext: Range bound of values to be returned by the server.

4.1.7.1.4 DS_REPL_INFO Codes

Type of replication state information being requested.

Symbolic name	Value
DS_REPL_INFO_NEIGHBORS	0x00000000
DS_REPL_INFO_CURSORS_FOR_NC	0x00000001
DS_REPL_INFO_METADATA_FOR_OBJ	0x00000002
DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES	0x00000003
DS_REPL_INFO_KCC_DSA_LINK_FAILURES	0x00000004
DS_REPL_INFO_PENDING_OPS	0x00000005
DS_REPL_INFO_METADATA_FOR_ATTR_VALUE	0x00000006
DS_REPL_INFO_CURSORS_2_FOR_NC	0x00000007
DS_REPL_INFO_CURSORS_3_FOR_NC	0x00000008
DS_REPL_INFO_METADATA_2_FOR_OBJ	0x00000009
DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE	0x0000000A
DS_REPL_INFO_SERVER_OUTGOING_CALLS	0xFFFFFFFA
DS_REPL_INFO_UPTODATE_VECTOR_V1	0xFFFFFFFB
DS_REPL_INFO_CLIENT_CONTEXTS	0xFFFFFFFC
DS_REPL_INFO_REPSTO	0xFFFFFFFE

4.1.7.1.5 DRS_MSG_GETREPLINFO_REPLY

The **DRS_MSG_GETREPLINFO_REPLY** union defines response messages received from the **IDL_DRSGetReplInfo** method.

```
typedef
[switch_type(DWORD)]
union {
  [case(0)]
    DS_REPL_NEIGHBORSW* pNeighbors;
  [case(1)]
    DS_REPL_CURSORS* pCursors;
  [case(2)]
    DS_REPL_OBJ_META_DATA* pObjMetaData;
  [case(3)]
    DS_REPL_KCC_DSA_FAILURESW* pConnectFailures;
  [case(4)]
    DS_REPL_KCC_DSA_FAILURESW* pLinkFailures;
  [case(5)]
    DS_REPL_PENDING_OPSW* pPendingOps;
  [case(6)]
    DS_REPL_ATTR_VALUE_META_DATA* pAttrValueMetaData;
  [case(7)]
    DS_REPL_CURSORS_2* pCursors2;
  [case(8)]
    DS_REPL_CURSORS_3W* pCursors3;
  [case(9)]
    DS_REPL_OBJ_META_DATA_2* pObjMetaData2;
  [case(10)]
    DS_REPL_ATTR_VALUE_META_DATA_2* pAttrValueMetaData2;
  [case(0xFFFFFFFA)]
    DS_REPL_SERVER_OUTGOING_CALLS* pServerOutgoingCalls;
  [case(0xFFFFFFFB)]
    UPTODATE_VECTOR_V1_EXT* pUpToDateVec;
  [case(0xFFFFFFFC)]
    DS_REPL_CLIENT_CONTEXTS* pClientContexts;
  [case(0xFFFFFFFE)]
    DS_REPL_NEIGHBORSW* pRepsTo;
} DRS_MSG_GETREPLINFO_REPLY;
```

pNeighbors: Neighbor information.

pCursors: Cursors for an NC replica.

pObjMetaData: Attribute stamps.

pConnectFailures: Connection failure data.

pLinkFailures: Link failure data.

pPendingOps: Pending operations in the replication queue.

pAttrValueMetaData: Link value stamps.

pCursors2: Cursors for an NC replica.

pCursors3: Cursors for an NC replica.

pObjMetaData2: Attribute stamps.

pAttrValueMetaData2: Link value stamps.

pServerOutgoingCalls: Outstanding requests from this DC to other DCs.

pUpToDateVec: Cursors for an NC replica.

pClientContexts: Active RPC client connections.

pRepsTo: Neighbor information.

4.1.7.1.6 DS_REPL_NEIGHBORSW

The **DS_REPL_NEIGHBORSW** structure defines a set of replication neighbors. This structure is a concrete representation of a sequence of RepsFrom or RepsTo values.

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

cNumNeighbors: Count of items in the **rgNeighbor** array.

dwReserved: MUST be 0.

rgNeighbor: Set of replication neighbors.

4.1.7.1.7 DS_REPL_NEIGHBORW

The **DS_REPL_NEIGHBORW** structure defines a replication neighbor. This structure is a concrete representation of a RepsFrom or RepsTo value.

```
typedef struct {
  [string] LPWSTR pszNamingContext;
  [string] LPWSTR pszSourceDsaDN;
  [string] LPWSTR pszSourceDsaAddress;
  [string] LPWSTR pszAsyncIntersiteTransportDN;
  DWORD dwReplicaFlags;
  DWORD dwReserved;
  UUID uuidNamingContextObjGuid;
  UUID uuidSourceDsaObjGuid;
  UUID uuidSourceDsaInvocationID;
  UUID uuidAsyncIntersiteTransportObjGuid;
  USN usnLastObjChangeSynced;
  USN usnAttributeFilter;
  FILETIME ftimeLastSyncSuccess;
  FILETIME ftimeLastSyncAttempt;
  DWORD dwLastSyncResult;
  DWORD cNumConsecutiveSyncFailures;
} DS_REPL_NEIGHBORW;
```

pszNamingContext: NC root of the NC replica.

pszSourceDsaDN: DN of server DC nTDSDSA object.

pszSourceDsaAddress: NetworkAddress of server DC.

pszAsyncIntersiteTransportDN: DN of the interSiteTransport object corresponding to the transport used to communicate with the server DC.

dwReplicaFlags: The **DRS_OPTIONS** flags.

dwReserved: MUST be 0.

uuidNamingContextObjGuid: objectGUID of the NC root.

uuidSourceDsaObjGuid: DSA GUID of the server DC.

uuidSourceDsaInvocationID: Invocation ID associated with the server DC.

uuidAsyncIntersiteTransportObjGuid: objectGUID of the interSiteTransport object corresponding to the transport used to communicate with the server DC.

usnLastObjChangeSynced: Implementation-specific value.

usnAttributeFilter: Implementation-specific value.

ftimeLastSyncSuccess: Time of last successful replication from the server DC.

ftimeLastSyncAttempt: Time of last attempt to replicate from the server DC.

dwLastSyncResult: 0, or Windows error code resulting from last sync attempt.

cNumConsecutiveSyncFailures: Number of consecutive failures to replicate from the server DC.

4.1.7.1.8 DS_REPL_CURSORS

The **DS_REPL_CURSORS** structure defines a set of replication cursors for a given NC replica. This structure is a concrete representation of a sequence of ReplUpToDateVector values.

```
typedef struct {
  DWORD cNumCursors;
  DWORD dwReserved;
  [size_is(cNumCursors)] DS_REPL_CURSOR rgCursor[];
} DS_REPL_CURSORS;
```

cNumCursors: Count of items in the **rgCursor** array.

dwReserved: MUST be 0.

rgCursor: Set of replication cursors.

4.1.7.1.9 DS_REPL_CURSOR

The **DS_REPL_CURSOR** structure defines a replication cursor for a given NC replica. This structure is a concrete representation of a ReplUpToDateVector value.

```
typedef struct {
  UUID uuidSourceDsaInvocationID;
  USN usnAttributeFilter;
} DS_REPL_CURSOR;
```

uuidSourceDsaInvocationID: Invocation ID of a DC.

usnAttributeFilter: USN at which an update was applied on the DC.

4.1.7.1.10 DS_REPL_CURSORS_2

The **DS_REPL_CURSORS_2** structure defines a set of replication cursors for a given NC replica. This structure is a concrete representation of a sequence of ReplUpToDateVector values; it is a superset of **DS_REPL_CURSORS**.

```
typedef struct {
  DWORD cNumCursors;
  DWORD dwEnumerationContext;
  [size_is(cNumCursors)] DS_REPL_CURSOR_2 rgCursor[];
} DS_REPL_CURSORS_2;
```

cNumCursors: Count of items in the **rgCursor** array.

dwEnumerationContext: 0xFFFFFFFF, or the range bound of the results.

rgCursor: Set of replication cursors.

4.1.7.1.11 DS_REPL_CURSOR_2

The **DS_REPL_CURSOR_2** structure defines a replication cursor for a given NC replica. This structure is a concrete representation of a ReplUpToDateVector value; it is a superset of **DS_REPL_CURSOR**.

```
typedef struct {
  UUID uuidSourceDsaInvocationID;
  USN usnAttributeFilter;
  FILETIME ftimeLastSyncSuccess;
} DS_REPL_CURSOR_2;
```

uuidSourceDsaInvocationID: Invocation ID of a DC.

usnAttributeFilter: USN at which an update was applied on the DC.

ftimeLastSyncSuccess: Time at which the last successful replication occurred from the DC identified by uuidDsa, for **replication latency** reporting only.

4.1.7.1.12 DS_REPL_CURSORS_3W

The **DS_REPL_CURSORS_3W** structure defines a replication cursor for a given NC replica. This structure is a concrete representation of a sequence of ReplUpToDateVector values; it is a superset of **DS_REPL_CURSORS_2**.

```
typedef struct {
  DWORD cNumCursors;
  DWORD dwEnumerationContext;
  [size_is(cNumCursors)] DS_REPL_CURSOR_3W rgCursor[];
} DS_REPL_CURSORS_3W;
```

cNumCursors: Count of items in the **rgCursor** array.

dwEnumerationContext: 0xFFFFFFFF, or the range bound of the results.

rgCursor: Set of replication cursors.

4.1.7.1.13 DS_REPL_CURSOR_3W

The **DS_REPL_CURSOR_3W** structure defines a replication cursor for a given NC replica. This structure is a concrete representation of a ReplUpToDateVector value; it is a superset of **DS_REPL_CURSOR_2**.

```
typedef struct {
  UUID uuidSourceDsaInvocationID;
  USN usnAttributeFilter;
  FILETIME ftimeLastSyncSuccess;
  [string] LPWSTR pszSourceDsaDN;
} DS_REPL_CURSOR_3W;
```

uuidSourceDsaInvocationID: Invocation ID of a DC.

usnAttributeFilter: USN at which an update was applied on the DC.

ftimeLastSyncSuccess: Time at which the last successful replication occurred from the DC identified by uuidDsa, for replication latency reporting only.

pszSourceDsaDN: DN of the nTDSDSA object with invocationId **uuidSourceDsaInvocationID**.

4.1.7.1.14 DS_REPL_OBJ_META_DATA

The **DS_REPL_OBJ_META_DATA** structure defines a set of attribute stamps for a given object. This structure is a concrete representation of the sequence of AttributeStamp values for all attributes of a given object.

```
typedef struct {
  DWORD cNumEntries;
  DWORD dwReserved;
  [size_is(cNumEntries)] DS_REPL_ATTR_META_DATA rgMetaData[];
} DS_REPL_OBJ_META_DATA;
```

cNumEntries: Count of items in the **rgMetaData** array.

dwReserved: MUST be 0.

rgMetaData: Set of attribute stamps.

4.1.7.1.15 DS_REPL_ATTR_META_DATA

The **DS_REPL_ATTR_META_DATA** structure defines an attribute stamp for a given object. This structure is a concrete representation of an AttributeStamp.

```
typedef struct {
  [string] LPWSTR pszAttributeName;
  DWORD dwVersion;
  FILETIME ftimeLastOriginatingChange;
  UUID uuidLastOriginatingDsaInvocationID;
  USN usnOriginatingChange;
  USN usnLocalChange;
} DS_REPL_ATTR_META_DATA;
```

pszAttributeName: lDAPDisplayName of the attribute to which the stamp corresponds.

dwVersion: Stamp version.

ftimeLastOriginatingChange: Date/time at which the last originating update was made.

uuidLastOriginatingDsaInvocationID: Invocation ID of the DC that performed the last originating update.

usnOriginatingChange: USN assigned to the last originating update by the DC that performed it.

usnLocalChange: Implementation-specific value.

4.1.7.1.16 DS_REPL_OBJ_META_DATA_2

The **DS_REPL_OBJ_META_DATA_2** structure defines a set of attribute stamps for a given object. This structure is a concrete representation of the sequence of AttributeStamp values for all attributes of a given object; it is a superset of **DS_REPL_OBJ_META_DATA**.

```
typedef struct {
  DWORD cNumEntries;
  DWORD dwReserved;
  [size_is(cNumEntries)] DS_REPL_ATTR_META_DATA_2 rgMetaData[];
} DS_REPL_OBJ_META_DATA_2;
```

cNumEntries: Count of items in the **rgMetaData** array.

dwReserved: MUST be 0.

rgMetaData: Set of attribute stamps.

4.1.7.1.17 DS_REPL_ATTR_META_DATA_2

The **DS_REPL_ATTR_META_DATA_2** structure defines an attribute stamp for a given object. This structure is a concrete representation of an AttributeStamp; it is a superset of **DS_REPL_ATTR_META_DATA**.

```
typedef struct {
  [string] LPWSTR pszAttributeName;
  DWORD dwVersion;
  FILETIME ftimeLastOriginatingChange;
  UUID uuidLastOriginatingDsaInvocationID;
  USN usnOriginatingChange;
  USN usnLocalChange;
  [string] LPWSTR pszLastOriginatingDsaDN;
} DS_REPL_ATTR_META_DATA_2;
```

pszAttributeName: lDAPDisplayName of the attribute to which the stamp corresponds.

dwVersion: Stamp version.

ftimeLastOriginatingChange: Date/time at which the last originating update was made.

uuidLastOriginatingDsaInvocationID: Invocation ID of the DC that performed the last originating update.

usnOriginatingChange: USN assigned to the last originating update by the DC that performed it.

usnLocalChange: Implementation-specific value.

pszLastOriginatingDsaDN: DN of the nTDSDSA object with invocationId uuidLastOriginatingDsaInvocationID.

4.1.7.1.18 DS_REPL_KCC_DSA_FAILURESW

The **DS_REPL_KCC_DSA_FAILURESW** structure defines a set of DCs that are in an error state with respect to replication. This structure is a concrete representation of KCCFailedConnections and KCCFailedLinks.

```
typedef struct {
  DWORD cNumEntries;
  DWORD dwReserved;
  [size_is(cNumEntries)] DS_REPL_KCC_DSA_FAILUREW rgDsaFailure[];
} DS_REPL_KCC_DSA_FAILURESW;
```

cNumEntries: Count of items in the **rgDsaFailure** array.

dwReserved: MUST be 0.

rgDsaFailure: Array of **DS_REPL_KCC_DSA_FAILUREW**.

4.1.7.1.19 DS_REPL_KCC_DSA_FAILUREW

The **DS_REPL_KCC_DSA_FAILUREW** structure defines a DC that is in a replication error state. This structure is a concrete representation of a tuple in a KCCFailedConnections or KCCFailedLinks sequence.

```
typedef struct {
  [string] LPWSTR pszDsaDN;
  UUID uuidDsaObjGuid;
  FILETIME ftimeFirstFailure;
  DWORD cNumFailures;
  DWORD dwLastResult;
} DS_REPL_KCC_DSA_FAILUREW;
```

pszDsaDN: DN of the nTDSDSA object corresponding to the DC.

uuidDsaObjGuid: DSA GUID of the DC.

ftimeFirstFailure: Date/time at which the DC entered an error state.

cNumFailures: Number of errors which have occurred.

dwLastResult: Windows error code for the last error.

4.1.7.1.20 DS_REPL_PENDING_OPSW

The **DS_REPL_PENDING_OPSW** structure defines a sequence of replication operations to be processed by a DC. This structure is a concrete representation of ReplicationQueue.

```
typedef struct {
  FILETIME ftimeCurrentOpStarted;
  DWORD cNumPendingOps;
  [size_is(cNumPendingOps)] DS_REPL_OPW rgPendingOp[];
} DS_REPL_PENDING_OPSW;
```

ftimeCurrentOpStarted: Time when the current operation started.

cNumPendingOps: Count of items in the **rgPendingOp** array.

rgPendingOp: Sequence of replication operations to be performed.

4.1.7.1.21 DS_REPL_OPW

The **DS_REPL_OPW** structure defines a replication operation to be processed by a DC. This structure is a concrete representation of a tuple in a ReplicationQueue sequence.

```
typedef struct {
  FILETIME ftimeEnqueued;
  ULONG ulSerialNumber;
  ULONG ulPriority;
  DS_REPL_OP_TYPE OpType;
  ULONG ulOptions;
```

```
  [string] LPWSTR pszNamingContext;
  [string] LPWSTR pszDsaDN;
  [string] LPWSTR pszDsaAddress;
  UUID uuidNamingContextObjGuid;
  UUID uuidDsaObjGuid;
} DS_REPL_OPW;
```

ftimeEnqueued: Date/time at which the operation was requested.

ulSerialNumber: Unique ID associated with the operation.

ulPriority: Priority of the operation.

OpType: Type of operation.

ulOptions: The **DRS_OPTIONS** flags.

pszNamingContext: NC root of the relevant NC replica.

pszDsaDN: DN of the relevant DC's nTDSDSA object.

pszDsaAddress: NetworkAddress of the relevant DC.

uuidNamingContextObjGuid: objectGUID of the NC root of the relevant NC replica.

uuidDsaObjGuid: DSA GUID of the DC.

4.1.7.1.22 DS_REPL_ATTR_VALUE_META_DATA

The **DS_REPL_ATTR_VALUE_META_DATA** structure defines a sequence of **link value stamps**. This structure is a concrete representation of a sequence of LinkValueStamp values.

```
typedef struct {
  DWORD cNumEntries;
  DWORD dwEnumerationContext;
  [size_is(cNumEntries)] DS_REPL_VALUE_META_DATA rgMetaData[];
} DS_REPL_ATTR_VALUE_META_DATA;
```

cNumEntries: Count of items in **rgMetaData** array.

dwEnumerationContext: 0xFFFFFFFF, or the range bound of the results.

rgMetaData: Sequence of link value stamps.

4.1.7.1.23 DS_REPL_VALUE_META_DATA

The **DS_REPL_VALUE_META_DATA** structure defines a link value stamp. This structure is a concrete representation of a LinkValueStamp.

```
typedef struct {
  [string] LPWSTR pszAttributeName;
  [string] LPWSTR pszObjectDn;
```

```
  DWORD cbData;
  [size_is(cbData), ptr] BYTE* pbData;
  FILETIME ftimeDeleted;
  FILETIME ftimeCreated;
  DWORD dwVersion;
  FILETIME ftimeLastOriginatingChange;
  UUID uuidLastOriginatingDsaInvocationID;
  USN usnOriginatingChange;
  USN usnLocalChange;
} DS_REPL_VALUE_META_DATA;
```

pszAttributeName: lDAPDisplayName of the attribute.

pszObjectDn: DN of the object.

cbData: Size in bytes of the **pbData** array.

pbData: Null, or data associated with the attribute value if the attribute is of syntax Object(DN-Binary) or Object(DN-String).

ftimeDeleted: Date/time at which the last replicated update was made that deleted the value, or 0 if the value is not currently deleted.

ftimeCreated: Date/time at which the first originating update was made.

dwVersion: Stamp version.

ftimeLastOriginatingChange: Date/time at which the last originating update was made.

uuidLastOriginatingDsaInvocationID: Invocation ID of the DC that performed the last originating update.

usnOriginatingChange: USN assigned to the last originating update by the DC that performed it.

usnLocalChange: Implementation-specific value.

4.1.7.1.24 DS_REPL_ATTR_VALUE_META_DATA_2

The **DS_REPL_ATTR_VALUE_META_DATA_2** structure defines a sequence of link value stamps. This structure is a concrete representation of a sequence of LinkValueStamp values; it is a superset of **DS_REPL_ATTR_VALUE_META_DATA**.

```
typedef struct {
  DWORD cNumEntries;
  DWORD dwEnumerationContext;
  [size_is(cNumEntries)] DS_REPL_VALUE_META_DATA_2 rgMetaData[];
} DS_REPL_ATTR_VALUE_META_DATA_2;
```

cNumEntries: Count of items in **rgMetaData** array.

dwEnumerationContext: 0xFFFFFFFF, or the range bound of the results.

rgMetaData: Sequence of link value stamps.

4.1.7.1.25 DS_REPL_VALUE_META_DATA_2

The **DS_REPL_VALUE_META_DATA_2** structure defines a link value stamp. This structure is a concrete representation of LinkValueStamp; it is a superset of **DS_REPL_VALUE_META_DATA**.

```
typedef struct {
  [string] LPWSTR pszAttributeName;
  [string] LPWSTR pszObjectDn;
  DWORD cbData;
  [size_is(cbData), ptr] BYTE* pbData;
  FILETIME ftimeDeleted;
  FILETIME ftimeCreated;
  DWORD dwVersion;
  FILETIME ftimeLastOriginatingChange;
  UUID uuidLastOriginatingDsaInvocationID;
  USN usnOriginatingChange;
  USN usnLocalChange;
  [string] LPWSTR pszLastOriginatingDsaDN;
} DS_REPL_VALUE_META_DATA_2;
```

pszAttributeName: lDAPDisplayName of the attribute.

pszObjectDn: DN of the object.

cbData: Size in bytes of the **pbData** array.

pbData: Null, or data associated with the attribute value if the attribute is of syntax Object(DN-Binary) or Object(DN-String).

ftimeDeleted: Date/time at which the last replicated update was made that deleted the value, or 0 if the value is not currently deleted.

ftimeCreated: Date/time at which the first originating update was made.

dwVersion: Stamp version.

ftimeLastOriginatingChange: Date/time at which the last originating update was made.

uuidLastOriginatingDsaInvocationID: Invocation ID of the DC that performed the last originating update.

usnOriginatingChange: USN assigned to the last originating update by the DC that performed it.

usnLocalChange: Implementation-specific value.

pszLastOriginatingDsaDN: DN of the nTDSDSA object with invocationId **uuidLastOriginatingDsaInvocationID**.

4.1.7.1.26 DS_REPL_CLIENT_CONTEXTS

The **DS_REPL_CLIENT_CONTEXTS** structure defines a set of active RPC client connections. This structure is a concrete representation of RPCClientContexts.

```
typedef struct {
  [range(0,10000)] DWORD cNumContexts;
  DWORD dwReserved;
  [size_is(cNumContexts)] DS_REPL_CLIENT_CONTEXT rgContext[];
} DS_REPL_CLIENT_CONTEXTS;
```

cNumContexts: Count of items in the **rgContext** array.

dwReserved: MUST be 0.

rgContext: Set of active RPC client connections.

4.1.7.1.27 DS_REPL_CLIENT_CONTEXT

The **DS_REPL_CLIENT_CONTEXT** structure defines an active RPC client connection. This structure is a concrete representation of a tuple in an RPCClientContexts sequence.

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

hCtx: Unique ID of the client context.

lReferenceCount: Number of references to the context.

fIsBound: True if and only if the context has not yet been closed by the **IDL_DRSUnbind** method.

uuidClient: Zeros, or the value pointed to by the *puuidClientDsa* parameter to **IDL_DRSBind**.

timeLastUsed: Date/time at which this context was last used in an RPC method call.

IPAddr: IPv4 address of the client. If the client connected with IPv6, this field contains zero.

pid: 0, or the process ID passed specified by the client in *pextClient* parameter to **IDL_DRSBind**.

4.1.7.1.28 DS_REPL_SERVER_OUTGOING_CALLS

The **DS_REPL_SERVER_OUTGOING_CALLS** structure defines a set of outstanding requests from this DC to other DCs. This structure is a concrete representation of RPCOutgoingContexts.

```
typedef struct {
  [range(0,256)] DWORD cNumCalls;
  DWORD dwReserved;
  [size_is(cNumCalls)] DS_REPL_SERVER_OUTGOING_CALL rgCall[];
```

```
} DS_REPL_SERVER_OUTGOING_CALLS;
```

cNumCalls: Count of items in the **rgCall** array.

dwReserved: MUST be 0.

rgCall: Set of outstanding requests from this DC to other DCs.

4.1.7.1.29 DS_REPL_SERVER_OUTGOING_CALL

The **DS_REPL_SERVER_OUTGOING_CALL** structure defines an outstanding request from this DC to another DC. This structure is a concrete representation of a tuple from an RPCOutgoingContexts sequence.

```
typedef struct {
  [string] LPWSTR pszServerName;
  BOOL fIsHandleBound;
  BOOL fIsHandleFromCache;
  BOOL fIsHandleInCache;
  DWORD dwThreadId;
  DWORD dwBindingTimeoutMins;
  DSTIME dstimeCreated;
  DWORD dwCallType;
} DS_REPL_SERVER_OUTGOING_CALL;
```

pszServerName: NetworkAddress of the server.

fIsHandleBound: True if and only if the **IDL_DRSBind** method has completed and the **IDL_DRSUnbind** method has not yet been called.

fIsHandleFromCache: True if and only the context handle used was retrieved from an implemented cache.

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

dwThreadId: Implementation specific thread ID of the thread that is using the context.

dwBindingTimeoutMins: If the context is set to be canceled then the timeout in minutes.

dstimeCreated: Date/time when the context was created.

dwCallType: Call that the client is waiting on. MUST be one of the following values:

Value	Meaning
2	**IDL_DRSBind**
3	**IDL_DRSUnbind**
4	**IDL_DRSReplicaSync**
5	Implemented server-to-server replication method

Value	Meaning
6	**IDL_DRSUpdateRefs**
7	**IDL_DRSReplicaAdd**
8	**IDL_DRSReplicaDel**
9	Implemented server-to-server object verification method
10	Implemented server-to-server group expansion method
11	Implemented server-to-server cross NC move method
12	Implemented server-to-server replication method
13	**IDL_DRSCrackNames**
14	Implemented server-to-server object creation method
15	Implemented server-to-server object lookup method
16	Implemented server-to-server replication method
17	**IDL_DRSGetReplInfo**

4.1.7.2 Method-Specific Abstract Types and Procedures

4.1.7.2.1 GetDNFromInvocationID

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

Returns the distinguished name of the nTDSDSA object which has the specified Invocation ID.

4.1.7.2.2 GetDNFromObjectGuid

```
procedure GetDNFromObjectGuid(guid: GUID): DN
```

Returns the distinguished name of the object with the specified object GUID. This is represented by the following expression:

```
obj := select one o from all where (o!objectGUID = guid)
return obj.dn
```

4.1.7.2.3 GetNCs

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

Returns a set containing the DSNames of all NCs hosted by this server.

4.1.7.3 Server Behavior of the IDL_DRSGetReplInfo Method

Informative summary of behavior: This method retrieves the replication state information of a DC. Based on the value of **InfoType** field in the request message, different information is returned. This is summarized below:

- DS_REPL_INFO_NEIGHBORS: Replication state data for each NC and source server pair, for all NC replicas hosted by this DC.
- DS_REPL_INFO_REPSTO: Replication state data for each NC and destination server (which is notified of changes) pair, for all NC replicas hosted by this DC.
- DS_REPL_INFO_CURSORS_FOR_NC
- DS_REPL_INFO_CURSORS_2_FOR_NC
- DS_REPL_INFO_CURSORS_3_FOR_NC: Replication state for the NC replica of a given NC.
- DS_REPL_INFO_METADATA_FOR_OBJ
- DS_REPL_INFO_METADATA_2_FOR_OBJ: Stamps for all the replicated attributes of the given object.
- DS_REPL_INFO_METADATA_FOR_ATTR_VALUE
- DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE: Stamps for a specific **link attribute** of the given object.
- DS_REPL_INFO_KCC_DSA_FAILURES: Replication state data with respect to connection failures with inbound replication partners.
- DS_REPL_INFO_KCC_LINK_FAILURES: Replication state data with respect to link failures with inbound replication partners.
- DS_REPL_INFO_PENDING_OPS: Replication tasks currently executing or queued to execute.
- DS_REPL_INFO_CLIENT_CONTEXTS: List of all outstanding RPC client contexts.
- DS_REPL_INFO_SERVER_OUTGOING_CALLS: List of all outstanding RPC server call contexts.

```
ULONG
IDL_DRSGetReplInfo(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)]
      DRS_MSG_GETREPLINFO_REQ *pmsgIn,
  [out, ref] DWORD *pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
      DRS_MSG_GETREPLINFO_REPLY *pmsgOut)

msgIn: DRS_MSG_GETREPLINFO_REQ_V2
infoType: DWORD
accessGranted: boolean
infoTypeValid: boolean
defaultNC: DSName
object: DSName
enumerationContext: DWORD
baseIndex: DWORD
```

```
endIndex: DWORD
ncs: set of DSName
nc: DSName
i, j: DWORD
r: RepsFrom
s: RepsTo
pNeighbor: ADDRESS OF DS_REPL_NEIGHBORW
utd: sequence of ReplUpToDateVector
pCursor: ADDRESS OF DS_REPL_CURSOR
pCursor2: ADDRESS OF DS_REPL_CURSOR_2
pCursor3: ADDRESS OF DS_REPL_CURSOR_3W
a: ATTRTYP
attr: ATTRTYP
attrs: set of ATTRTYP
attrsSeq: sequence of ATTRTYP
s: AttributeStamp
pObjMetaData: ADDRESS OF DS_REPL_OBJ_META_DATA
pObjMetaData2: ADDRESS OF DS_REPL_OBJ_META_DATA_2
values: set of attribute value
valuesSeq: sequence of attribute value
ls: LinkValueStamp
pAttrValueMetaData: ADDRESS OF DS_REPL_ATTR_VALUE_META_DATA
pAttrValueMetaData2: ADDRESS OF DS_REPL_ATTR_VALUE_META_DATA_2
dc: DC
pFailedConnection: ADDRESS OF DS_REPL_KCC_DSA_FAILUREW
pFailedLink: ADDRESS OF DS_REPL_KCC_DSA_FAILUREW
pPendingOp: ADDRESS OF DS_REPL_OPW
pClientContext: ADDRESS OF DS_REPL_CLIENT_CONTEXT
pOutgoingContext: ADDRESS OF DS_REPL_SERVER_OUTGOING_CALL

/* Validate the version of the request message */
if (dwInVersion ≠ 1 and dwInVersion ≠ 2) then
  return ERROR_REVISION_MISMATCH
endif

if dwInVersion = 1 then
  msgIn := pmsgIn^.V1
else
  msgIn := pmsgIn^.V2
endif

/* For some of the request types, paging is supported. For these
 * cases, we need a starting index into the result set based on
* what has already been returned in a previous call. Only version 2
* request messages provide a mechanism for the client to supply the
* context information from a previous call. */
if dwInVersion = 1 then
  baseIndex := 0
else
  baseIndex := msgIn.dwEnumerationContext
endif

/* Perform the necessary access checks. */
defaultNC := DefaultNC()
fAccessGranted := false
if (infoType = DS_REPL_INFO_NEIGHBORS and object ≠ null) then
  infoTypeValid := true
  fAccessGranted :=
```

```
    AccessCheckAttr(object, repsFrom, RIGHT_DS_READ_PROPERTY) or
    AccessCheckCAR(object, DS-Replication-Manage-Topology) or
    AccessCheckCAR(object, DS-Replication-Monitor-Topology)
endif
if (infoType = DS_REPL_INFO_NEIGHBORS and object = null) then
  infoTypeValid := true
  fAccessGranted :=
    AccessCheckAttr(defaultNC, repsFrom, RIGHT_DS_READ_PROPERTY) or
    AccessCheckCAR(defaultNC, DS-Replication-Manage-Topology) or
    AccessCheckCAR(defaultNC, DS-Replication-Monitor-Topology)
endif
if (infoType = DS_REPL_INFO_REPSTO and object ≠ null) then
  infoTypeValid := true
  fAccessGranted :=
    AccessCheckAttr(object, repsTo, RIGHT_DS_READ_PROPERTY) or
    AccessCheckCAR(object, DS-Replication-Manage-Topology) or
    AccessCheckCAR(object, DS-Replication-Monitor-Topology)
endif
if (infoType = DS_REPL_INFO_REPSTO and object = null) then
  infoTypeValid := true
  fAccessGranted :=
    AccessCheckAttr(defaultNC, repsTo, RIGHT_DS_READ_PROPERTY) or
    AccessCheckCAR(defaultNC, DS-Replication-Manage-Topology) or
    AccessCheckCAR(defaultNC, DS-Replication-Monitor-Topology)
endif
if infoType in {DS_REPL_INFO_CURSORS_FOR_NC,
                DS_REPL_INFO_CURSORS_2_FOR_NC,
                DS_REPL_INFO_CURSORS_3_FOR_NC,
                DS_REPL_INFO_UPTODATE_VECTOR_V1} then
  infoTypeValid := true
  fAccessGranted :=
    AccessCheckAttr(
        object, replUpToDateVector, RIGHT_DS_READ_PROPERTY) or
    AccessCheckCAR(object, DS-Replication-Manage-Topology) or
    AccessCheckCAR(object, DS-Replication-Monitor-Topology)
endif
if infoType in {DS_REPL_INFO_METADATA_FOR_OBJ,
                DS_REPL_INFO_METADATA_2_FOR_OBJ,
                DS_REPL_INFO_METADATA_FOR_ATTR_VALUE,
                DS_REPL_INFO_METATDATA_2_FOR_ATTR_VALUE} then
  infoTypeValid := true
  fAccessGranted :=
       AccessCheckAttr(object,
                       replPropertyMetaData,
                       RIGHT_DS_READ_PROPERTY) or
       AccessCheckCAR(object, DS-Replication-Manage-Topology) or
       AccessCheckCAR(object, DS-Replication-Monitor-Topology)
endif
if infoType in {DS_REPL_INFO_PENDING_OPS,
                DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES,
                DS_REPL_INFO_KCC_DSA_LINK_FAILURES,
                DS_REPL_INFO_CLIENT_CONTEXTS,
                DS_REPL_INFO_SERVER_OUTGOING_CALLS} then
  infoTypeValid := true
  fAccessGranted :=
    AccessCheckCAR(defaultNC, DS-Replication-Manage-Topology) or
    AccessCheckCAR(defaultNC, DS-Replication-Monitor-Topology)
endif
```

```
if not infoTypeValid then
  return ERROR_INVALID_PARAMETER
endif

if not accessGranted then
  return ERROR_DS_DRA_ACCESS_DENIED
endif

/* Based on the type of information requested, the corresponding
* information is retrieved and the response message constructed */

/* DS_REPL_INFO_NEIGHBORS/DS_REPL_INFO_REPSTO */
if infoType in {DS_REPL_INFO_NEIGHBORS, DS_REPL_INFO_REPSTO}
  /* If an object is specified, it must be an NC root. */
  nc := object

  if nc ≠ null and not FullReplicaExists(nc) and not
      PartialGCReplicaExists(nc) then
    return ERROR_DS_DRA_BAD_NC
  endif

  if baseIndex = 0xffffffff then
    /* No more data is available. */
    return ERROR_NO_MORE_ITEMS
  endif

  if nc ≠ null then
    ncs := {nc}
  else
    ncs := GetNCs()
  endif

  if infoType = DS_REPL_INFO_NEIGHBORS then
    i := 0
    j := 0
    foreach nc in ncs
      foreach r in nc!repsFrom
        /* The ordering of ncs hosted by the server and the values of
         * repsFrom for each nc is arbitrary but consistent from call
         * to call on a server. */

        /* If a source server GUID is specified, only information for
         * that server is returned. */
        If (msgIn.uuidSourceDsaGuid = NULL GUID or
              msgIn.uuidSourceDsaGuid = r.uuidDsa) then
          if i >= baseIndex then
            pNeighbor := ADR(pmsgOut^.pNeighbors^.rgNeighbor[j])
            pNeighbor^.pszSourceDsaAddress := r.naDsa
            pNeighbor^.dwReplicaFlags := r.options
            pNeighbor^.timeLastAttempt := r.timeLastAttempt
            pNeighbor^.uuidSourceDsaObjGuid := r.uuidDsa
            pNeighbor^.pszSourceDsaDN :=
                GetDNFromObjectGuid(r.uuidDsa)
            pNeighbor^.pszNamingContext := nc!distinguishedName
            pNeighbor^.uuidNamingContextObjGuid := nc!objectGUID
            pNeighbor^.pszAsyncIntersiteTransportDN :=
                GetDNFromObjectGuid(r.uuidTransportObj)
```

```
            pNeighbor^.uuidSourceDsaInvocationID := r.uuidInvocId
            pNeighbor^.uuidAsyncIntersiteTransportObjGuid :=
                r.uuidTransportObj
            pNeighbor^.usnLastObjChangeSynced :=
                r.usnVec.usnHighObjUpdate
            pNeighbor^.usnAttributeFilter :=
                r.usnVec.usnHighPropUpdate
            pNeighbor^.fTimeLastSyncSuccess := r.timeLastSuccess
            pNeighbor^.dwLastSyncResult := r.ulResultLastAttempt
            pNeighbor^.cNumConsecutiveSyncFailures :=
                r.cbConsecutiveFailures
            j := j + 1
          endif
          i := i + 1
        endif
      endfor
    endfor
    pmsgOut^.pNeighbors^.cNumNeighbors := j
  else
    /* DS_REPL_INFO_REPSTO case. */
    i := 0
    j := 0
    foreach nc in ncs
      foreach r in nc!repsTo
        /* The ordering of ncs hosted by the server and the values of
         * repsTo for each nc is arbitrary but consistent from call
         * to call on a server. */
        if i >= baseIndex then
          pNeighbor := ADR(pmsgOut^.pNeighbors^.rgNeighbor[j])
          pNeighbor^.pszSourceDsaAddress := r.naDsa
          pNeighbor^.dwReplicaFlags := r.options
          pNeighbor^.timeLastAttempt := r.timeLastAttempt
          pNeighbor^.uuidSourceDsaObjGuid := r.uuidDsa
          pNeighbor^.pszSourceDsaDN := GetDNFromObjectGuid(r.uuidDsa)
          pNeighbor^.pszNamingContext := nc!distinguishedName
          pNeighbor^.uuidNamingContextObjGuid := nc!objectGUID
          j := j + 1
        endif
        i := i + 1
      endfor
    endfor
    pmsgOut^.pRepsTo^.cNumNeighbors := j
  endif
endif

/* DS_REPL_INFO_METADATA_FOR_OBJ/DS_REPL_INFO_METADATA_2_FOR_OBJ */
if infoType in {DS_REPL_INFO_METADATA_FOR_OBJ,
    DS_REPL_INFO_METADATA_2_FOR_OBJ) then
  /* Basic parameter validation */
  if object = null or not ObjExists(object) then
    return ERROR_INVALID_PARAMETER
  endif

  if baseIndex = 0xffffffff then
    /* No more data is available. */
    return ERROR_NO_MORE_ITEMS
  endif
```

```
/* Enumerate all the replicated attributes of the object */
attrsSeq := select all a from Replicated Attributes of object
i := 0
j := 0
while (i < attrsSeq.length)
  attr := attrsSeq[i]
  s := AttrStamp(object, attr)

  if (attr in Link Attributes of object and
      dwInVersion = 2 and
      DS_REPL_INFO_FLAG_IMPROVE_LINKED_ATTRS in msgIn.ulFlags)
        then
    ls := LinkValueStamp of the most recent
          value change in object!attr
    if ls is more recent than s (based on order in which
    the change was applied on server) then
      if s = null then
        s := 0 /* An AttributeStamp with 0 for all fields. */
      endif

      /* Improve the stamp with the link value stamp. */
      s.dwVersion := ls.dwVersion
      s.timeChanged := ls.timeChanged
      s.uuidOriginating := NULL GUID
      s.usnOriginating := ls.usnOriginating
    endif
  endif

  if s ≠ null then
    if i >= baseIndex
      if infoType = DS_REPL_INFO_METADATA_FOR_OBJ then
        pObjMetaData := ADR(pmsgOut^.pObjMetaData1->rgMetaData[j])
        pObjMetaData^.pszAttributeName := attr
        pObjMetaData^.dwVersion := s.dwVersion
        pObjMetaData^.timeChanged := s.timeChanged
        pObjMetaData^.uuidLastOriginatingDsaInvocationID :=
            s.uuidOriginating
        pObjMetaData^.usnOriginatingChange := s.usnOriginating
        pObjMetaData^.usnLocalChange :=
            An implementation specific value that the server
            maintains for replicated attributes
      else
        pObjMetaData2 := ADR(pmsgOut^.pObjMetaData2->rgMetaData[j])
        pObjMetaData2^.pszAttributeName := attr
        pObjMetaData2^.dwVersion := s.dwVersion
        pObjMetaData2^.timeChanged := s.timeChanged
        pObjMetaData2^.uuidLastOriginatingDsaInvocationID :=
            s.uuidOriginating
        pObjMetaData2^.usnOriginatingChange := s.usnOriginating
        pObjMetaData2^.usnLocalChange :=
            An implementation specific value that the server
            maintains for replicated attributes
        pObjMetaData2^.pszLastOriginatingDsaDN :=
            GetDNFromInvocationID(s.uuidOriginating)
      endif
      j := j + 1
    endif
    i := i + 1
```

```
    endif
  endwhile
  pmsgOut^.pObjMetaData2^.cNumEntries = j
endif

/* DS_REPL_INFO_CURSORS_FOR_NC */
if infoType = DS_REPL_INFO_CURSORS_FOR_NC then
  /* Parameter validation */
  /* The NC root object must be specified */
  nc := object

  if nc = null then
    return ERROR_INVALID_PARAMETER
  endif

  if not FullReplicaExists(nc) and
      not PartialGCReplicaExists(nc) then
    return ERROR_DS_DRA_BAD_NC
  endif

  if baseIndex = 0xffffffff then
    /* No more data is available. */
    return ERROR_NO_MORE_ITEMS
  endif

  utd := nc!replUpToDateVector
  i := baseIndex
  j := 0
  while i < utd.length
    pCursor := pmsgOut^.pCursors^.rgCursors[j]
    pCursor^.uuidSourceDsaInvocationID := utd[i].uuidDsa
    pCursor^.usnAttributeFilter := utd[i].usnHighPropUpdate
    i := i + 1
    j := j + 1
  endwhile
  pmsgOut^.pCursors^.cNumCursors := j
endif

/* DS_REPL_INFO_CURSORS_2_FOR_NC/ DS_REPL_INFO_CURSORS_3_FOR_NC */
if infoType in {DS_REPL_INFO_CURSORS_2_FOR_NC,
    DS_REPL_INFO_CURSORS_3_FOR_NC} then

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

  if (nc = null) then
    return ERROR_INVALID_PARAMETER
  endif

  if not FullReplicaExists(nc) and
      not PartialGCReplicaExists(nc) then
    return ERROR_DS_DRA_BAD_NC
  endif

  if baseIndex = 0xffffffff then
    /* No more data is available. */
    return ERROR_NO_MORE_ITEMS
```

```
    endif

    i := baseIndex
    j := 0
    utd := nc!replUpToDateVector

    /* A maximum of 1000 items will be sent in each call. */
    if utd.length - baseIndex - 1 > 1000 then
      endIndex = baseIndex + 1000
    else
      endIndex = utd.length
    endif

    while i < endIndex
      if infoType = DS_REPL_INFO_CURSORS_2_FOR_NC then
        pCursor2 := ADR(pmsgOut^.pCursors2^.rgCursors[j])
        pCursor2^.uuidSourceDsaInvocationID := utd[i].uuidDsa
        pCursor2^.usnAttributeFilter := utd[i].usnHighPropUpdate
        pCursor2^.ftimeLastSyncSucess := utd[i].timeLastSyncSuccess
      else
        pCursor3 := ADR(pmsgOut^.pCursor3^.rgCursors[j])
        pCursor3^.uuidSourceDsaInvocationID := utd[i].uuidDsa
        pCursor3^.usnAttributeFilter := utd[i].usnHighPropUpdate
        pCursor3^.ftimeLastSyncSucess := utd[i].timeLastSyncSuccess
        pCursor3^.pszSourceDsaDN :=
            GetDNFromInvocationID(utd[i].uuidDsa)
      endif
      j := j + 1
      i := i + 1
    endwhile
    pmsgOut^.pCursors3^.cNumCursors := j
    if i < utd.length - 1 then
      /* Not all items could be sent back in this call, so save the
       * index of the first item to be sent in the next call. */
      pmsgOut^.pCursors3^.dwEnumerationContext := i
    else
      /* No more data is available. */
      pmsgOut^.pCursors3^.dwEnumerationContext := 0xffffffff
    endif
  endif

  /* DS_REPL_INFO_UPTODATE_VECTOR_V1 */
  if infoType = DS_REPL_INFO_UPTODATE_VECTOR_V1 then
    /* Parameter validation. */
    /* The NC root object must be specified. */
    nc := object

    if (nc = null) then
      return ERROR_INVALID_PARAMETER
    endif

    if not FullReplicaExists(nc) and
        not PartialGCReplicaExists(nc) then
      return ERROR_DS_DRA_BAD_NC
    endif

    utd := nc!replUpToDateVector
    for i := 0 to utd.length - 1
```

```
      pCursor := ADR(pmsgOut^.pUpToDateVec^.rgCursors[i])
      pCursor^.uuidDsa := utd[i].uuidDsa
      pCursor^.usnHighPropUpdate := utd[i].usnHighPropUpdate
    endfor
    pmsgOut^.pUpToDateVec^.cNumCursors := utd.length
  endif

  /* DS_REPL_INFO_METADATA_FOR_ATTR_VALUE/
   * DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE */
  if infoType in {DS_REPL_INFO_METADATA_FOR_ATTR_VALUE,
      DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE} then

    /* Parameter validation. */
    if (object = null or not ObjExists(object)) then
      return ERROR_INVALID_PARAMETER
    endif

    if baseIndex = 0xffffffff then
      /* No more data is available. */
      return ERROR_NO_MORE_ITEMS
    endif

    /* If the attribute name is specified it must be a link
     * attribute. */
    attrs := select all a in Link Attributes of object
    if (pmsgIn^.V2.pszAttributeNameValue ≠ null and
        pmsgIn^.V2.pszAttributeNameValue not in attrs) then
      return ERROR_DS_WRONG_LINKED_ATT_SYNTAX
    endif

    /* If the attribute name is not specified, replication state for a
     * link attribute of the object which has a value is returned. */
    if (pmsgIn^.V2.pszAttributeNameValue ≠ null) then
      attr := pmsgIn^.V2.pszAttributeNameValue
    else
      attrsSeq := select all a in attrs where
          GetAttrVals(object, a, true) ≠ null
      attr := attrsSeq[0]
    endif

    if attr ≠ null then
      valuesSeq := GetAttrVals(object, attr, true)

      /* If a start value has been specified, then start at the first
       * occurrence of that value in the sequence of values, otherwise
       * start at the index determined from the enumeration context
       * which specifies the index of the next value to be returned. */
      if (pmsgIn^.V2.pszValueDN ≠ null and
          Syntax(attr) = Object(DS-DN)) then
        i := index of pmsgIn^.V2.pszValueDN in valuesSeq
      else
        i := baseIndex
      endif

      j := 0
      while (i < valuesSeq.length and j < 1000)
        ls := LinkStamp(object, attr, valuesSeq[i])
        if infoType = DS_REPL_INFO_METADATA_FOR_ATTR_VALUE then
```

```
      pAttrValueMetaData :=
          ADR(pmsgOut^.pAttrValueMetaData^.rgMetadata[j])
      pAttrValueMetaData^.pszAttributeName := attr
      pAttrValueMetaData^.pszObjectDN := object!distinguishedName
      if (Syntax(attr) = Object(DN-Binary) or
          Syntax(attr) = Object(DN-String)) then
        pAttrValueMetaData^.cbData :=
            length of data associated with valuesSeq[i]
        pAttrValueMetaData^.pbData := data associated with
            valuesSeq[i]
      endif
      pAttrValueMetaData^.ftimeCreated := ls.timeCreated
      pAttrValueMetaData^.ftimeDeleted := ls.timeDeleted
      pAttrValueMetaData^.dwVersion := ls.dwVersion
      pAttrValueMetaData^.ftimeLastOriginatingChange :=
          ls.timeChanged
      pAttrValueMetaData^.uuidLastOriginatingDsaInvocationID :=
          ls.uuidOriginating
      pAttrValueMetaData^.usnOriginatingChange := ls.usnOriginating
      pAttrValueMetaData^.usnLocalChange :=
          implementation-specific value maintained for each link
          attribute value
    else
      pAttrValueMetaData2 :=
          pmsgOut^.pAttrValueMetaData2^.rgMetadata[j]
      pAttrValueMetaData2^.pszAttributeName := attr
      pAttrValueMetaData2^.pszObjectDN := object!distinguishedName
      if (Syntax(attr) = Object(DN-Binary) or
          Syntax(attr) = Object(DN-String)) then
        pAttrValueMetaData2^.cbData :=
            length of data associated with valuesSeq[i]
        pAttrValueMetaData2^.pbData :=
            data associated with valuesSeq[i]
      endif
      pAttrValueMetaData2^.ftimeCreated := ls.timeCreated
      pAttrValueMetaData2^.ftimeDeleted := ls.timeDeleted
      pAttrValueMetaData2^.dwVersion := ls.dwVersion
      pAttrValueMetaData2^.ftimeLastOriginatingChange :=
          ls.timeChanged
      pAttrValueMetaData2^.uuidLastOriginatingDsaInvocationID :=
          ls.uuidOriginating
      pAttrValueMetaData2^.usnOriginatingChange :=
          ls.usnOriginating
      pAttrValueMetaData2^.usnLocalChange :=
          implementation-specific value maintained for each
          link attribute value
      pAttrValueMetaData2^.pszLastOriginatingDsaDN :=
          GetDNFromInvocationID(ls.uuidOriginating)
    endif

    i := i + 1
    j := j + 1
  endwhile

  if i < valuesSeq.length - 1 then
    /* Since there are more entries to be returned, save the index
     * of the first value to be returned in the next call. */
    pmsgOut^.pAttrValueMetaData2^.dwEnumerationContext := i
```

```
    else
      /* No more data is available. */
      pmsgOut^.pAttrValueMetaData2^.dwEnumerationContext :=
          0xffffffff
    endif

    pmsgOut^.pAttrValueMetaData2^.cNumEntries = j
  endif
endif

/* DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES */
if infoType = DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES then
  i := 0
  foreach t in dc.kccFailedConnections
    pConnectionFailure :=
        ADR(pmsgOut^.pConnectionFailures^.rgDsaFailure[i])
    pConnectionFailure^.pszDsaDN := t.DsaDN
    pConnectionFailure^.uuidDsaObjGuid := t.UUIDDsa
    pConnectionFailure^.fTimeFirstFailure := t.TimeFirstFailure
    pConnectionFailure^.cNumFailures := t.FailureCount
    pConnectionFailure^.dwLastResult := t.LastResult
    i := i + 1
  endfor
  pmsgOut^.pConnectionFailures^.cNumEntries := i
endif

/* DS_REPL_INFO_KCC_LINK_FAILURES */
if infoType = DS_REPL_INFO_KCC_LINK_FAILURES then
  i := 0
  foreach t in dc.kccFailedLinks
    pConnectionLink := ADR(pmsgOut^.pLinkFailures^.rgDsaFailure[i])
    pConnectionLink^.pszDsaDN := t.DsaDN
    pConnectionLink^.uuidDsaObjGuid := t.UUIDDsa
    pConnectionLink^.fTimeFirstFailure := t.TimeFirstFailure
    pConnectionLink^.cNumFailures := t.FailureCount
    pConnectionLink^.dwLastResult := t.LastResult
    i := i + 1
  endfor
  pmsgOut^.pConnectionLinks^.cNumEntries := i
endif

/* DS_REPL_INFO_PENDING_OPS */
if infoType = DS_REPL_INFO_PENDING_OPS then
  i := 0
  foreach t in dc.replicationQueue
    pPendingOp := ADR(pmsgOut^.pPendingOps^.rgPendingOp[i])
    pPendingOp^.fTimeEnqueued := t.TimeEnqueued
    pPendingOp^.ulSerailNumber := t.SerialNumber
    pPendingOp^.ulPriority := t.Priority
    pPendingOp^.OpType := t.OperationType
    pPendingOp^.ulOptions := t.Options
    pPendingOp^.pszNamingContext := t.NamingContext
    pPendingOp^.pszDsaDN := t.DsaDN
    pPendingOp^.pszDsaAddress := t.DsaAddress
    pPendingOp^.uuidNamingContextObjGuid := t.UUIDNC
    pPendingOp^.uuidDsaObjGuid := t.UUIDDsa
    i := i + 1
  endfor
```

```
    pmsgOut^.pPendingOps^.cNumPendingOps := i
    pmsgOut^.pPendingOps^.fTimeCurrentOpStarted := time when current
        operation was started
  endif

  /* DS_REPL_INFO_CLIENT_CONTEXTS */
  if infoType = DS_REPL_INFO_CLIENT_CONTEXTS then
    i := 0
    foreach t in dc.rpcClientContexts
      pClientContext := ADR(pmsgOut^.pClientContexts^.rgContext[i])
      pClientContext^.hCtx := t.BindingContext
      pClientContext^.lReferenceCount := t.RefCount
      pClientContext^.fIsBound := t.IsBound
      pClientContext^.uuidClient := t.UUIDClient
      pClientContext^.timeLastUsed := t.TimeLastUsed
      pClientContext^.IPAddr := t.IPAddress
      pClientContext^.pid := t.PID
      i := i + 1
    endfor
    pmsgOut^.pClientContexts^.cNumContexts := i
  endif

  /* DS_REPL_INFO_SERVER_OUTGOING_CALLS */
  if infoType = DS_REPL_INFO_SERVER_OUTGOING_CALLS then
    i := 0
    foreach t in dc.rpcOutgoingContexts
      pOutgoingContext =
          ADR(pmsgOut^.pServerOutgoingCalls^.rgContext[i])
      pOutgoingContext^.pszServerName := t.ServerName
      pOutgoingContext^.fIsHandleBound := t.IsBound
      pOutgoingContext^.fIsHandleFromCache := t.HandleFromCache
      pOutgoingContext^.fIsHandleInCache := t.HandleInCache
      pOutgoingContext^.dwThreadId := t.ThreadId
      pOutgoingContext^.dwBindingTimeoutMins := t.BindingTimeOut
      pOutgoingContext^.dstimeCreated := t.CreateTime
      pOutgoingContext^.dwCallType := t.CallType
      i := i + 1
    endfor
    pmsgOut^.pServerOutgoingCalls^.cNumCalls := i
  endif

  return 0
```

4.1.8 IDL_DRSInitDemotion (Opnum 25)

The **IDL_DRSInitDemotion** method performs the first phase of the removal of a DC from an AD/LDS forest. This method is supported only by AD/LDS. This method is used only to diagnose, monitor, and manage the implementation of server-to-server DC demotion. The structures requested and returned through this method MAY have meaning to peer DCs and applications, but are not required for interoperation with Windows clients.

```
  ULONG IDL_DRSInitDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
      DRS_MSG_INIT_DEMOTIONREQ* pmsgIn,
```

```
  [out, ref] DWORD* pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_INIT_DEMOTIONREPLY* pmsgOut
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

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

4.1.8.1 Method-Specific Concrete Types

4.1.8.1.1 DRS_MSG_INIT_DEMOTIONREQ

The **DRS_MSG_INIT_DEMOTIONREQ** union defines request messages sent to the **IDL_DRSInitDemotion** method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
  [case(1)]
    DRS_MSG_INIT_DEMOTIONREQ_V1 V1;
} DRS_MSG_INIT_DEMOTIONREQ;
```

V1: Version 1 request. Currently only one version is defined.

4.1.8.1.2 DRS_MSG_INIT_DEMOTIONREQ_V1

The **DRS_MSG_INIT_DEMOTIONREQ_V1** structure defines a request message sent to the **IDL_DRSInitDemotion** method.

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

dwReserved: MUST be 0.

4.1.8.1.3 DRS_MSG_INIT_DEMOTIONREPLY

The **DRS_MSG_INIT_DEMOTIONREPLY** union defines the response messages received from the **IDL_DRSInitDemotion** method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
  [case(1)]
    DRS_MSG_INIT_DEMOTIONREPLY_V1 V1;
} DRS_MSG_INIT_DEMOTIONREPLY;
```

V1: Version 1 reply.

4.1.8.1.4 DRS_MSG_INIT_DEMOTIONREPLY_V1

The **DRS_MSG_INIT_DEMOTIONREPLY_V1** structure defines a response message received from the **IDL_DRSInitDemotion** method.

```
typedef struct {
  DWORD dwOpError;
} DRS_MSG_INIT_DEMOTIONREPLY_V1;
```

dwOpError: A Win32 error code, as specified in [MS-ERREF] section 3.0.

4.1.8.2 Server Behavior of the IDL_DRSInitDemotion Method

Informative summary of behavior: Performs the first phase of the removal of a DC from an AD/LDS forest. This phase consists of (1) disabling both originating and replicated updates to the AD/LDS DC, and (2) marking the database as read-only. Both of these effects are outside the state model.

```
ULONG
IDL_DRSInitDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_INIT_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_INIT_DEMOTIONREPLY* pmsgOut
    )
msgIn: DRS_MSG_INIT_DEMOTIONREQ_V1
ret: DWORD
if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
if not IsMemberOfBuiltinAdminGroup() then
  /* only BA is allowed to demote an AD/LDS service */
  return ERROR_ACCESS_DENIED
endif
Disable
ret := Disable originating and replicated updates
if ret = ERROR_SUCCESS then
  ret := Mark database as read-only
endif
pmsgOut^.dwOpError := ret
```

```
pdwMsgOut^ := 1
return ERROR_SUCCESS
```

4.1.9 IDL_DRSQuerySitesByCost (Opnum 24)

The **IDL_DRSQuerySitesByCost** method determines the communication cost from a "from" site to one or more "to" sites.

```
ULONG IDL_DRSQuerySitesByCost(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)]
    DRS_MSG_QUERYSITESREQ* pmsgIn,
  [out, ref] DWORD* pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_QUERYSITESREPLY* pmsgOut
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

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

4.1.9.1 Method-Specific Concrete Types

4.1.9.1.1 DRS_MSG_QUERYSITESREQ

The **DRS_MSG_QUERYSITESREQ** union defines the request message versions sent to the **IDL_DRSQuerySitesByCost** method. Only one version, identified by dwVersion = 1, is currently defined.

```
typedef
[switch_type(DWORD)]
union {
  [case(1)]
    DRS_MSG_QUERYSITESREQ_V1 V1;
} DRS_MSG_QUERYSITESREQ;
```

V1: Version 1 request.

4.1.9.1.2 DRS_MSG_QUERYSITESREQ_V1

The **DRS_MSG_QUERYSITESREQ_V1** structure defines a request message sent to the **IDL_DRSQuerySitesByCost** method.

```
typedef struct {
  [string] const WCHAR* pwszFromSite;
  [range(1,10000)] DWORD cToSites;
  [string, size_is(cToSites)] WCHAR** rgszToSites;
  DWORD dwFlags;
} DRS_MSG_QUERYSITESREQ_V1;
```

pwszFromSite: RDN of site object of the "from" site.

cToSites: Count of items in the **rgszToSites** array (count of "to" sites).

rgszToSites: RDNs of site objects of the "to" sites.

dwFlags: MUST be 0.

4.1.9.1.3 DRS_MSG_QUERYSITESREPLY

The **DRS_MSG_QUERYSITESREPLY** union defines the response messages received from the **IDL_DRSQuerySitesByCost** method.

```
typedef
[switch_type(DWORD)]
union {
  [case(1)]
    DRS_MSG_QUERYSITESREPLY_V1 V1;
} DRS_MSG_QUERYSITESREPLY;
```

V1: Version 1 response.

4.1.9.1.4 DRS_MSG_QUERYSITESREPLY_V1

The **DRS_MSG_QUERYSITESREPLY_V1** structure defines a response message received from the **IDL_DRSQuerySitesByCost** method.

```
typedef struct {
  [range(0,10000)] DWORD cToSites;
  [size_is(cToSites)] DRS_MSG_QUERYSITESREPLYELEMENT_V1* rgCostInfo;
  DWORD dwFlags;
} DRS_MSG_QUERYSITESREPLY_V1;
```

cToSites: Count of items in the **rgCostInfo** array.

rgCostInfo: Sequence of computed site costs, in the same order as the **rgszToSites** field in the request message.

dwFlags: MUST be 0.

4.1.9.1.5 DRS_MSG_QUERYSITESREPLYELEMENT_V1

The **DRS_MSG_QUERYSITESREPLYELEMENT_V1** structure defines the computed cost of communication between two sites.

```
typedef struct {
  DWORD dwErrorCode;
  DWORD dwCost;
} DRS_MSG_QUERYSITESREPLYELEMENT_V1;
```

dwErrorCode: 0 if this "from-to" computation was successful, or ERROR_DS_OBJ_NOT_FOUND if the "to" site does not exist.

dwCost: Communication cost between the "from" site and this "to" site, or 0xFFFFFFFF if the sites are not connected.

4.1.9.2 Method-Specific Abstract Types and Procedures

4.1.9.2.1 ValidateSiteRDN

```
procedure ValidateSiteRDN(s: unicodestring): boolean
```

Returns true if s is a valid RDN for a site object. A valid RDN is not null, does not have 0 length, does not have length greater than 256, and contains no occurrences of the characters '=' and ',' .

4.1.9.2.2 WeightedArc and WeightedArcSet

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

The cost field of a WeightedArc is positive.

4.1.9.2.3 MinWeightPath

```
procedure MinWeightPath(
  vSet: set of DSName,
  aSet: WeightedArcSet): WeightedArcSet
```

Returns a WeightedArcSet where for each WeightedArc a, a.initial and a.final are vertices in vSet, a.final is reachable from a.initial in the graph G = (vSet, aSet), and a.cost is the cost of the minimum-cost path in G from a.initial to a.final.

4.1.9.3 Server Behavior of the IDL_DRSQuerySitesByCost Method

Informative summary of behavior: Given a site *fromSite* and an array of sites *toSites*, returns an array containing the cost from *fromSite* to each element of *toSite*, where cost is defined as follows.

The server computes a weighted graph G = (V, A). Each vertex in V corresponds to a site object. Each arc in A corresponds to a siteLink object that connects two vertices in V; the weight of an arc is the value of attribute cost on the arc's siteLink object. The cost of a path in the graph is the sum of

the arc weights on the path. The cost from one site to another is the minimum-cost path between the two sites.

The model just described corresponds to fully-transitive communications between sites: If site *a* communicates with site *b* and site *b* communicates with site *c*, then site *a* communicates with site *c* by routing through *b*. The server-to-server replication implementation can be configured to restrict transitive communication to sites specified in the same siteLinkBridge object. Suppose there is a siteLink object for site *a* and site *b*, and a siteLink object for site *b* and site *c*, but no siteLink object for site *a* and site *c*. If both of the siteLink objects are specified on the same siteLinkBridge object, site *a* can communicate with site *c* by routing through *b*. If no such siteLinkBridge object exists, site *a* cannot communicate with site *c*.

To calculate the cost when siteLinkBridge objects are used, let *nBridges* be the number of siteLinkBridge objects. For each k in the subrange [0 .. *nBridges*-1], construct a weighted graph G[k] = (V, A[k]) using siteLinkBridge object b[k]. Graph G[k] has the same vertex set as G, but its arc set A[k] is a subset of A, including only the arcs listed in attribute siteLinkList on siteLinkBridge object b[k]. Then cost from site *a* to site *c* is the minimum of the following costs:

1. The cost of the arc, if any, from *a* to *c* in G.
2. For each k in the subrange [0 .. *nBridges*-1], the cost of the minimum cost path, if any, from *a* to *c* in G[k].

Any authenticated user can perform this operation; no access checking is performed.

```
ULONG
IDL_DRSQuerySitesByCost(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_QUERYSITESREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_QUERYSITESREPLY *pmsgOut)
msgIn: DRS_MSG_QUERYSITESREQ_V1
vSet, slSet, sbSet : set of DSName
aSet, aSetB, aSetC, aSetD: WeightedArcSet
siteContainer, ipObject, fromSite, toSite: DSName
u, v, sl, sb: DSName
i, c: integer
min: WeightedArc
/* Perform input validation,
 * initialize siteContainer, ipObject, fromSite. */
if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
if not ValidateSiteRDN(msgIn.pwszFromSite) then
  return ERROR_INVALID_PARAMETER
endif
if msgIn.cToSites > 0 and msgIn.rgszToSites = null then
  return ERROR_INVALID_PARAMETER
endif
for i := 0 to msgIn.cToSites - 1
   if not ValidateSiteRDN(msgIn.rgszToSites[i]) then
     return ERROR_INVALID_PARAMETER
   endif
```

```
endfor
siteContainer := DescendantObject(ConfigNC(), "CN=Sites,")
ipObject := DescendantObject(ConfigNC(),
    "CN=IP,CN=Inter-Site Transports,CN=Sites,")
fromSite := select one v from children siteContainer where
    site in v!objectClass and v!name = msgIn.pswzFromSite
if fromSite = null then
  return ERROR_DS_OBJ_NOT_FOUND
endif
/* Construct the vertex set vSet. */
vSet := select all v from children siteContainer where
    site in v!objectClass
if vSet = {} then
  return ERROR_DS_OBJ_NOT_FOUND
endif
/* Construct the arc set aSet. */
slSet := select all v from children ipObject where
    siteLink in v!objectClass
foreach sl in slSet
  foreach u in sl!siteList
    foreach v in sl!siteList - {u}
      aSet := aSet + {[initial: u, final: v, cost: sl!cost]}
    endfor
  endfor
endfor
/* Construct minimum-cost arc set aSetC.
 * See [MS-ADTS] section 7.1.1.2.2.3.1 for definition
 * of the option NTDSTRANSPORT_OPT_BRIDGES_REQUIRED. */
if NTDSTRANSPORT_OPT_BRIDGES_REQUIRED in ipObject!options then
  /* Perform construction using siteLinkBridge objects.
   * Initial minimum cost is the cost of a direct arc if any. */
  aSetC := aSet
  sbSet := select all v from children ipObject where
      siteLinkBridge in v!objectclass
  foreach sb in sbSet
    /* Compute the minimum cost using this siteLinkBridge. */
    aSetB := {}
    foreach sl in sb!siteLinkList
      foreach u in sl!siteList
        foreach v in sl!siteList - {u}
          aSetB := aSetB + {[initial: u, final: v, cost: sl!cost)}
        endfor
      endfor
    endfor
    aSetD := MinWeightPath(vSet, aSetB)
    /* Here aSetD contains the minimum cost arc set using this
     * siteLinkBridge. Improve the current minimum cost using
     * aSetD. */
    foreach [initial: u, final: v, cost: c] in aSetD
      min := select one t from aSetC where
          t.initial = u and t.final = v
      if min = null then
        aSetC := aSetC + {[initial: u, final: v, cost: c)}
      else if min.cost > c then
        aSetC := aSetC - {[initial: u, final: v, cost: min.cost]}
                       + {[initial: u, final: v, cost: c)}
      endif
    endfor
```

```
  endfor
else
  /* Fully transitive network, ignore siteLinkBridge objects. */
  aSetC := MinWeightPath(vSet, aSet)
endif
/* Construct result message. */
pdwOutVersion^ := 1
pmsgOut^.V1.cToSites := msgIn.cToSites
pmsgOut^.V1.dwFlags := 0
for i:= 0 to msgIn.cToSites - 1
  toSite := select one v from children siteContainer where
      site in v!objectClass and v!name = msgIn.rgszToSites[i]
  if not (toSite in vSet) then
    pmsgOut^.V1.rgCostInfo[i].dwErrorCode := ERROR_DS_OBJ_NOT_FOUND
    pmsgOut^.V1.rgCostInfo[i].dwCost := 0xffffffff
  else
    min := select one t from aSetC where
        t.initial = fromSite and t.final = toSite
    if min ≠ null then
      pmsgOut^.V1.rgCostInfo[i].dwErrorCode := 0
      pmsgOut^.V1.rgCostInfo[i].dwCost := min.cost
    else
      pmsgOut^.V1.rgCostInfo[i].dwErrorCode = 0
      pmsgOut^.V1.rgCostInfo[i].dwCost := 0xffffffff
    endif
  endif
endfor
return 0
```

4.1.10 IDL_DRSRemoveDsDomain (Opnum 15)

The **IDL_DRSRemoveDsDomain** method removes the representation (also known as metadata) of a domain from the directory.

```
ULONG IDL_DRSRemoveDsDomain(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)]
    DRS_MSG_RMDMNREQ* pmsgIn,
  [out, ref] DWORD* pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_RMDMNREPLY* pmsgOut
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message. This MUST be set to 1, as this is the only version supported.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message. The value will always be 1, since that is the only version supported.

pmsgOut: Pointer to the response message.

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

4.1.10.1 Method-Specific Concrete Types

4.1.10.1.1 DRS_MSG_RMDMNREQ

The **DRS_MSG_RMDMNREQ** union defines the request messages sent to the **IDL_DRSRemoveDsDomain** method. Only one version, identified by dwInVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.10.1.2 DRS_MSG_RMDMNREQ_V1

The **DRS_MSG_RMDMNREQ_V1** structure defines a request message sent to the **IDL_DRSRemoveDsDomain** method.

```
typedef struct {
  [string] LPWSTR DomainDN;
} DRS_MSG_RMDMNREQ_V1;
```

DomainDN: DN of the NC root of the domain NC to remove.

4.1.10.1.3 DRS_MSG_RMDMNREPLY

The **DRS_MSG_RMDMNREPLY** union defines the response messages received from the **IDL_DRSRemoveDsDomain** method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

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

V1: Version 1 response.

4.1.10.1.4 DRS_MSG_RMDMNREPLY_V1

The **DRS_MSG_RMDMNREPLY_V1** structure defines a response message received from the **IDL_DRSRemoveDsDomain** method.

```
typedef struct {
  DWORD Reserved;
} DRS_MSG_RMDMNREPLY_V1;
```

Reserved: MUST be 0.

4.1.10.2 Method-Specific Abstract Types and Procedures

4.1.10.2.1 HasNCReplicated

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

Returns true if the DC's NC replica of the NC specified by nc has replicated at least once with another DC that hosts that NC since the DC was booted; otherwise, returns false.

4.1.10.3 Server Behavior of the IDL_DRSRemoveDsDomain Method

Informative summary of behavior: Removes the crossRef object defining a domain NC. Fails if any DC is currently hosting this domain as its default NC, as indicated by the state of that DC's nTDSDSA object. Fails if the server is not the Domain Naming FSMO role owner for the forest.

The removal of the crossRef object signals any DC currently hosting a partial replica of the removed domain NC to remove that replica from its state.

The method IDL_ DRSRemoveDsServer removes the state within a forest, including state on a DC's nTDSDSA object, associated with hosting a domain as a default NC on some DC. Therefore, IDL_DRSRemoveDsServer can be used to establish a precondition for the success of IDL_DRSRemoveDsDomain.

```
ULONG
IDL_DRSRemoveDsDomain(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_RMDMNREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_RMDMNREPLY *pmsgOut);

domainDN: unicodestring
otherNtdsdsa: DSName
cr: DSName

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

if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif

domainDN := pmsgIn^.V1.DomainDN

if domainDN = null or domainDN = "" then
```

```
  return ERROR_INVALID_PARAMETER
endif

/* Originating updates are not allowed on RODC */
if AmIRODC() then
  return ERROR_DS_OBJ_NOT_FOUND
endif

/* This function cannot be called on a DC for the domain
 * to be removed. */
if DefaultNC().dn = domainDN then
  return ERROR_DS_ILLEGAL_MOD_OPERATION
endif

/* Make sure no DCs still have NC replicas of this domain NC. */
otherNtdsdsa := select one o from ConfigNC() where
             (nTDSDSA in o!objectClass) and
             (domainDN in o!hasMasterNCs or
              domainDN in o!msDS-hasMasterNCs)
if otherNtdsdsa ≠ null then
  return ERROR_DS_NC_STILL_HAS_DSAS
endif

/* Find the crossRef object for the domain named by domainDN. */
cr := select one o from ConfigNC() where
        (o!nCName = domainDN) and (crossRef in o!objectClass)

if cr = null then
  return ERROR_DS_NO_CROSSREF_FOR_NC
endif

/* Make sure we are the Domain Naming FSMO role owner */
if GetFSMORoleOwner(FSMO_DOMAIN_NAMING) != DSAObj()) then
  /* We are not the Domain Naming FSMO role owner */
  return ERROR_DS_OBJ_NOT_FOUND
else
  /* We are the Domain Naming FSMO role owner. If the Config NC
   * has not replicated at least once since startup, our ownership
   * of the NC is not considered to be verified, and we exit with
   * an error. */
  if not HasNCReplicated(ConfigNC()) then
    return ERROR_DS_ROLE_NOT_VERIFIED;
  endif
endif

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

RemoveObj(cr)
return 0
```

4.1.11 IDL_DRSRemoveDsServer (Opnum 14)

The **IDL_DRSRemoveDsServer** method removes the representation (also known as metadata) of a DC from the directory.

```
ULONG IDL_DRSRemoveDsServer(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)]
    DRS_MSG_RMSVRREQ* pmsgIn,
  [out, ref] DWORD* pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_RMSVRREPLY* pmsgOut
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message. MUST be set to 1, since that is the only version supported.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message. The value will always be 1, since that is the only version supported.

pmsgOut: Pointer to the response message.

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

4.1.11.1 Method-Specific Concrete Types

4.1.11.1.1 DRS_MSG_RMSVRREQ

The **DRS_MSG_RMSVRREQ** union defines the request messages sent to the **IDL_DRSRemoveDsServer** method. Only one version, identified by dwInVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.11.1.2 DRS_MSG_RMSVRREQ_V1

The **DRS_MSG_RMSVRREQ_V1** structure defines a request message sent to the **IDL_DRSRemoveDsServer** method.

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

ServerDN: DN of the server object of the DC to remove.

DomainDN: DN of the NC root of the domain that the DC to be removed belongs to. May be set to NULL if the client does not desire the server to compute the value of pmsgOut^.V1.DomainDN.

fCommit: True if the DC's metadata should actually be removed from the directory. False if the metadata should not be removed. (This is used by a client that wants to determine the value of pmsgOut^.V1.fLastDcInDomain without altering the directory.)

4.1.11.1.3 DRS_MSG_RMSVRREPLY

The **DRS_MSG_RMSVRREPLY** union defines the response messages received from the **IDL_DRSRemoveDsServer** method.

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

V1: Version 1 response.

4.1.11.1.4 DRS_MSG_RMSVRREPLY_V1

The **DRS_MSG_RMSVRREPLY_V1** structure defines a response message received from the **IDL_DRSRemoveDsServer** method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

```
typedef struct {
  BOOL fLastDcInDomain;
} DRS_MSG_RMSVRREPLY_V1;
```

fLastDcInDomain: True if the DC is the last DC in its domain and pmsgIn^.V1.DomainDN was set to the DN of the NC root of the domain to which the DC belongs. False otherwise.

4.1.11.2 Server Behavior of the IDL_DRSRemoveDsServer Method

Informative summary of behavior: Removes the metadata defining a DC, which consists of the tree of objects rooted at the DC's nTDSDSA object as well as the DRS SPNs associated with the DC's computer object. This method is typically used if a DC is removed from the domain without being properly demoted (for example, if the DC suffers a fatal hardware failure), a client may make this call in order to remove the metadata of the no-longer-existent DC. When pmsgIn^.V1.DomainDN is specified, this function also computes whether the DC is the last replica of its default domain NC.

The behavior of this function has two variants. If pmsgIn^.V1.fCommit is false, the function is read-only with regards to abstract state, that is, it does not make any changes to the contents of the directory. In this mode the main purpose of the function is to compute pmsgOut^.V1.fLastDcInDomain (and so there is little point to calling the function in this mode without setting pmsgIn^.V1.DomainDN). For example, prior to removing the DC's metadata a client

application might want to determine whether any DCs would be left in the domain, so that it can warn the user if the user is removing the last DC in the domain.

When pmsgIn^.V1.fCommit is true, the second variant of the behavior is performed. In this mode, the function actually removes the DC metadata. The pmsgOut^.V1.fLastDcInDomain value is also computed in this mode (provided that pmsgIn^.V1.DomainDN was passed in). The removal of the DC's metadata signals other DCs in the forest that this particular DC no longer exists.

```
ULONG
IDL_DRSRemoveDsServer(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_RMSVRREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_RMSVRREPLY *pmsgOut);

serverDn: unicodestring
domainDn: unicodestring
ntdsdsa: DSName
otherNtdsdsa: DSName
spnsToRemove: set of unicodestring
computerDn: unicodestring
computer: DSName
objectsToDelete: set of DSName

serverDn := pmsgIn^.V1.ServerDN
domainDn := pmsgIn^.V1.DomainDN

pdwOutVersion^ := 1

/* Basic parameter validation */
if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif

if serverDn = null or serverDn = "" then
  return ERROR_INVALID_PARAMETER
endif

/* Note that DomainDN may be null, but it can not be empty. */
if domainDn = "" then
  return ERROR_INVALID_PARAMETER
endif

/* Originating updates are not allowed on RODC */
if AmIRODC() then
  return ERROR_DS_OBJ_NOT_FOUND
endif

ntdsdsa := DescendantObject([dn: serverDn], "CN=NTDS Settings,")
if ntdsdsa = null then
  return ERROR_DS_NO_SUCH_OBJECT
endif

/* Compute fLastDcInDomain if domainDn is non-null. */
if domainDn ≠ null then
```

```
  otherNtdsdsa := select one o from subtree ConfigNC() where
    (o!objectCategory = nTDSDSA)
    and
    (domainDn in o!hasMasterNCs or domainDn in o!msDS-hasMasterNCs)
    and
    (o ≠ ntdsdsa)
  if otherNtdsdsa = null then
    pmsgOut^.V1.fLastDcInDomain = true
  else
    pmsgOut^.V1.fLastDcInDomain = false
  endif
endif

/* If nothing to commit, processing is complete. */
if not pmsgIn^.V1.fCommit then
  return 0
endif

/* Perform the actual DC metadata removal. */

/* Locate the computer object for the DC's account. */
computerDn := ntdsdsa!serverReference
computer := null
if computerDn ≠ null then
  computer := GetDSNameFromDN(computerDn)
endif

/* Remove the subtree of objects rooted at the DC's ntdsDsa object.*/
objectsToDelete := select all o from subtree ntdsdsa where (true)

if not AccessCheckObject(ntdsdsa, RIGHT_DS_DELETE_TREE) then
  return ERROR_ACCESS_DENIED
endif

foreach o in objectsToDelete
  RemoveObj(o)
endfor

/* If the DC's computer account exists,
 * remove the DRS SPNs from the computer object. */


  foreach spn in computer!servicePrincipalName
    if StartsWith(spn, "ldap/") or
       StartsWith(spn, "GC/") or
       StartsWith(spn, "E3514235-4B06-11D1-AB04-00C04FC2DCD2/") then
      spnsToRemove := spnsToRemove + {spn}
    endif
  endfor

  if not AccessCheckAttr(computer, servicePrincipalName,
      RIGHT_DS_WRITE_PROPERTY) then
    return ERROR_ACCESS_DENIED
  endif

  computer!servicePrincipalName :=
    computer!servicePrincipalName - spnsToRemove
endif
```

```
return 0
```

4.1.12 IDL_DRSReplicaAdd (Opnum 5)

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

```
ULONG IDL_DRSReplicaAdd(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwVersion,
  [in, ref, switch_is(dwVersion)]
    DRS_MSG_REPADD* pmsgAdd
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwVersion: Version of the request message.

pmsgAdd: Pointer to the request message.

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

4.1.12.1 Method-Specific Concrete Types

4.1.12.1.1 DRS_MSG_REPADD

The **DRS_MSG_REPADD** union defines request messages that are sent to the **IDL_DRSReplicaAdd** method.

```
typedef
[switch_type(DWORD)]
union {
  [case(1)]
    DRS_MSG_REPADD_V1 V1;
  [case(2)]
    DRS_MSG_REPADD_V2 V2;
} DRS_MSG_REPADD;
```

V1: Version 1 request (Windows 2000 and later).

V2: Version 2 request (Windows 2000 and later; superset of V1).

4.1.12.1.2 DRS_MSG_REPADD_V1

The **DRS_MSG_REPADD_V1** structure defines a request message sent to the **IDL_DRSReplicaAdd** method. This request version is supported by Windows 2000 and later releases of Windows.

```
typedef struct {
  [ref] DSNAME* pNC;
  [ref, string] char* pszDsaSrc;
  REPLTIMES rtSchedule;
  ULONG ulOptions;
} DRS_MSG_REPADD_V1;
```

pNC: NC root of the NC to replicate.

pszDsaSrc: Transport-specific NetworkAddress of the DC from which to replicate updates.

rtSchedule: Schedule at which to perform periodic replication.

ulOptions: Zero or more **DRS_OPTIONS** flags.

4.1.12.1.3 DRS_MSG_REPADD_V2

The **DRS_MSG_REPADD_V2** structure defines a request message sent to the **IDL_DRSReplicaAdd** method. This request version is a superset of V1 and is supported by Windows 2000 and later releases of Windows.

```
typedef struct {
  [ref] DSNAME* pNC;
  [unique] DSNAME* pSourceDsaDN;
  [unique] DSNAME* pTransportDN;
  [ref, string] char* pszSourceDsaAddress;
  REPLTIMES rtSchedule;
  ULONG ulOptions;
} DRS_MSG_REPADD_V2;
```

pNC: NC root of the NC to replicate.

pSourceDsaDN: nTDSDSA object for the DC from which to replicate changes.

pTransportDN: interSiteTransport object identifying the network transport to be used in the server-to-server replication implementation with the specified DC.

pszSourceDsaAddress: Transport-specific NetworkAddress of the DC from which to replicate updates.

rtSchedule: Schedule at which to perform periodic replication.

ulOptions: Zero or more **DRS_OPTIONS** flags.

4.1.12.2 Server Behavior of the IDL_DRSReplicaAdd Method

Informative summary of behavior: The server adds a value to the repsFrom of the specified NC replica. If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously. The client can be an administrative client or another DC. The client includes DRS_WRIT_REP in ulOptions if the specified NC replica is writable at the server. The client includes DRS_FULL_REP and DRS_SELECT_SECRET_REP in ulOptions if the specified NC replica is a read-only full replica at the server. The server adds a value to repsFrom, the value has replicaFlags derived from ulOptions (see

below), serverAddress equal to pszSourceDsaAddress (pszDsaSrc if V1), and schedule equal to rtSchedule. If ulOptions contains DRS_ASYNC_REP but not DRS_MAIL_REP or DRS_NEVER_NOTIFY, the server sends a request to the DC specified by pszSourceDsaAddress to add a value to the repsTo of the specified NC replica by calling IDL_DRSUpdateRefs. Finally, the server begins replication by sending a server-to-server replication request.

```
ULONG
IDL_DRSReplicaAdd(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwVersion,
  [in, ref, switch_is(dwVersion)] DRS_MSG_REPADD *pmsgAdd);

options: DRS_OPTIONS
nc: DSName
partitionsObj: DSName
cr: DSName
rf: RepsFrom
msgIn: DRS_MSG_REPADD_V2
updRefs: DRS_MSG_UPDREFS /* See IDL_DRSUpdateRefs structures. */

/* Validate the version */
if dwVersion ≠ 1 and dwVersion ≠ 2 then
  return ERROR_INVALID_PARAMETER
endif
if dwVersion = 1 then
  msgIn := pmsgAdd^.V1
  msgIn.pszSourceDsaAddress = pmsgAdd^.V1.pszDsaSrc
else
  msgIn := pmsgAdd^.V2
endif

if msgIn.pNC = null
   or msgIn.pszSourceDsaAddress = null
   or msgIn.pszSourceDsaAddress = "" then
  return ERROR_INVALID_PARAMETER
endif

options := msgIn.ulOptions
nc := msgIn.pNC^

partitionsObj :=
    select one o from children ConfigNC() where o!name = "Partitions"
cr := select o from children partitionsObj where o!nCName = nc
if cr = null then
  return ERROR_DS_DRA_BAD_DN
endif
if AmIRODC() and not DRS_WRIT_REP in options then
  return ERROR_INVALID_PARAMETER
endif

if ObjExists(nc) then
  if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
    return ERROR_DS_DRA_ACCESS_DENIED
  endif
else
  if not AccessCheckCAR(DefaultNC(), DS-Replication-Manage-Topology)
        then
    return ERROR_DS_DRA_ACCESS_DENIED
```

```
  endif
endif

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

if ObjExists(nc) then
  if (IT_WRITE in nc!instanceType) ≠ (DRS_WRIT_REP in options) then
    return ERROR_DS_DRA_BAD_NC
  endif
  /* Disallow addition if server already replicates from this
   * source */
  if (select one v from nc!repsFrom
        where v.serverAddress = msgIn.pszSourceDsaAddress) ≠ null
        then
    return ERROR_DS_DRA_DN_EXISTS
  endif
endif

if msgIn.pSourceDsaDN ≠ null
    and (not ObjExists(msgIn.pSourceDsaDN^)
         or not nTDSDSA in msgIn.pSourceDsaDN^!objectClass
         or GetObjectNC(msgIn.pSourceDsaDN^) ≠ ConfigNC()) then
  return ERROR_INVALID_PARAMETER
endif

if msgIn.pTransportDN ≠ null
    and (not ObjExists(msgIn.pTransportDN^)
         or not interSiteTransport in msgIn.pTransportDN^!objectClass
         or GetObjectNC(msgIn.pTransportDN^) ≠ ConfigNC()) then
  return ERROR_INVALID_PARAMETER
endif

/* Construct RepsFrom value. */
if msgIn.pSourceDsaDN ≠ null then
  rf.uuidDsa := msgIn.pSourceDsaDN^!objectGUID
endif
if msgIn.pTransportDN ≠ null then
  rf.uuidTransportObj := msgIn.pTransportDN^!objectGUID
endif
rf.replicaFlags := msgIn.ulOptions ∩ {DRS_DISABLE_AUTO_SYNC,
    DRS_DISABLE_PERIODIC_SYNC, DRS_INIT_SYNC, DRS_MAIL_REP,
    DRS_NEVER_NOTIFY, DRS_PER_SYNC, DRS_TWOWAY_SYNC,
    DRS_USE_COMPRESSION, DRS_WRIT_REP, DRS_FULL_REP,
    DRS_SELECT_SECRET_REP }
rf.schedule := msgIn.rtSchedule^
rf.serverAddress := msgIn.pszSourceDsaAddress^
rf.timeLastAttempt := current time

if msgIn.ulOptions ∩ {DRS_ASYNC_REP, DRS_NEVER_NOTIFY, DRS_MAIL_REP,
                      DRS_FULL_REP} = {DRS_ASYNC_REP} then
  /* Enable replication notifications by requesting the server DC
   * to add a repsTo for this DC. */
  updRefs.pNC^ := ADR(nc)
  updRefs.pszDsaDest := NetworkAddress of this DC
```

```
    updRefs.uuidDsaDest := dc.serverGuid
    updRefs.ulOptions := {DRS_ASYNC_OP, DRS_ADD_REF, DRS_DEL_REF}
    if DRS_WRIT_REP in msgIn.ulOptions then
      updRefs.ulOptions := updRefs.ulOptions + {DRS_WRIT_REP}
    endif
    Send updRefs request by calling IDL_DRSUpdateRefs() on server
        msgIn.pszSourceDsaAddress^
  endif

  err := PerformReplication(nc, msgIn.pSourceDsaDN)

  return err
```

4.1.13 IDL_DRSReplicaDel (Opnum 6)

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

```
ULONG IDL_DRSReplicaDel(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwVersion,
  [in, ref, switch_is(dwVersion)]
    DRS_MSG_REPDEL* pmsgDel
);
```

hDrs: RPC context handle returned by **IDL_DRSBind**.

dwVersion: Version of the request message.

pmsgDel: Pointer to the request message.

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

4.1.13.1 Method-Specific Concrete Types

4.1.13.1.1 DRS_MSG_REPDEL

The **DRS_MSG_REPDEL** union defines the request messages sent to the **IDL_DRSReplicaDel** method. Only one version, identified by dwVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.13.1.2 DRS_MSG_REPDEL_V1

The **DRS_MSG_REPDEL_V1** structure defines a request message sent to the **IDL_DRSReplicaDel** method.

```
typedef struct {
  [ref] DSNAME* pNC;
  [string] char* pszDsaSrc;
  ULONG ulOptions;
} DRS_MSG_REPDEL_V1;
```

pNC: Pointer to DSName of the root an NC replica on the server.

pszDsaSrc: Transport-specific NetworkAddress of a DC.

ulOptions: **DRS_OPTIONS** flags.

4.1.13.2 Server Behavior of the IDL_DRSReplicaDel Method

Informative summary of behavior: When DRS_NO_SOURCE is not specified, the server removes a value from the repsFrom of the specified NC replica. If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously. The client must include DRS_WRIT_REP in ulOptions if the specified NC replica is a writable replica. The server removes the value from repsFrom whose serverAddress matches pszDsaSrc. If ulOptions does not contain DRS_LOCAL_ONLY, the server sends a request to the DC specified by pszDsaSrc to remove this DC from the values in repsTo of the specified NC replica by calling IDL_DRSUpdateRefs.

When DRS_NO_SOURCE is specified, the server **expunges** the NC replica and all its children. If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously. The client must include DRS_WRIT_REP in ulOptions if the specified NC replica is writable. If ulOptions contains DRS_ASYNC_REP, the server expunges the objects asynchronously.

```
ULONG
IDL_DRSReplicaDel(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwVersion,
  [in, ref, switch_is(dwVersion)] DRS_MSG_REPDEL *pmsgDel);
options: DRS_OPTIONS
nc: DSName
rf: RepsFrom
msgIn: DRS_MSG_REPDEL_V1
updRefs: DRS_MSG_UPDREFS /* See IDL_DRSUpdateRefs structures. */
/* Validate dwVersion */
if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgDel^.V1
/* Validate the NC */
if msgIn.pNC = null then
  return ERROR_INVALID_PARAMETER
endif
nc := msgIn.pNC^
options := msgIn.ulOptions
if not ObjExists(nc) then
```

```
  return ERROR_DS_DRA_BAD_DN
endif
if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
  return ERROR_DS_DRA_ACCESS_DENIED
endif
if DRS_ASYNC_OP in options then
  Asynchronous Processing: Initiate a logical thread of control
   to process the remainder of this request asynchronously
  return 0
endif
options := msgIn.ulOptions
if (IT_WRITE in nc!instanceType) ≠ (DRS_WRIT_REP in options) then
  return ERROR_DS_DRA_BAD_NC
endif
if DRS_NO_SOURCE in options then
  /* Expunging local copy of an NC. */
  /* NC must not replicate from any other DC. */
  if (select one v from nc!repsFrom where (true)) ≠ null then
    return ERROR_INVALID_PARAMETER
  endif
  /* NC should not replicate to any other DC. */
  if (select one v from nc!repsTo where (true)) ≠ null
     and (not DRS_REF_OK in options) then
    return ERROR_DS_DRA_OBJ_IS_REP_SOURCE
  endif
  /* Do not permit removal of important NCs. */
  if IT_WRITE in nc!instanceType
     and (nc = DefaultNC()
          or nc = ConfigNC()
          or nc = SchemaNC()) then
    return ERROR_INVALID_PARAMETER
  endif
  if DRS_ASYNC_REP in options then
    Asynchronous Processing: Initiate a logical thread of control
     to process the remainder of this request asynchronously
    return 0
  endif
  /* Expunge the subtree rooted at dn, inclusive. */
  foreach o in (select all v from subtree nc where (true))
    Expunge(o)
  endfor
  return 0
else /* not DRS_NO_SOURCE in options */
  /* Removing a single source from repsFrom, but leaving NC replica
   * on DC. */
  if msgIn.pszDsaSrc = null or msgIn.pszDsaSrc^ = "" then
    return ERROR_INVALID_PARAMETER
  endif
  rf := select one v from nc!repsFrom
        where (v.serverAddress = msgIn.pszDsaSrc)
  if rf = null then
    return ERROR_DS_DRA_NO_REPLICA
  endif
  nc!repsFrom := nc!repsFrom - {rf}
  if (not DRS_LOCAL_ONLY in options)
     and (not DRS_MAIL_REP in rf.options) then
    /* Disable replication notifications by requesting the DC
     * specified by msgIn.pszDsaSrc to remove this DC's
```

```
     * entry from its msgIn.pNC^!repsTo. */
    updRefs.pNC^ := ADR(nc)
    updRefs.pszDsaDest := NetworkAddress of this DC
    updRefs.uuidDsaDest := dc.serverGuid
    updRefs.ulOptions := {DRS_ASYNC_OP, DRS_DEL_REF}
    if DRS_WRIT_REP in msgIn.ulOptions then
      updRefs.ulOptions := updRefs.ulOptions + {DRS_WRIT_REP}
    endif
    Send updRefs request by calling IDL_DRSUpdateRefs() on server
        msgIn.pszDsaSrc^
  endif
  return 0
endif
```

4.1.14 IDL_DRSReplicaDemotion (Opnum 26)

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

```
ULONG IDL_DRSReplicaDemotion(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)]
    DRS_MSG_REPLICA_DEMOTIONREQ* pmsgIn,
  [out, ref] DWORD* pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_REPLICA_DEMOTIONREPLY* pmsgOut
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message.

pmsgOut: Pointer to the response message.

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

4.1.14.1 Method-Specific Concrete Types

4.1.14.1.1 DRS_MSG_REPLICA_DEMOTIONREQ

The **DRS_MSG_REPLICA_DEMOTIONREQ** union defines the request messages sent to the **IDL_DRSReplicaDemotion** method. Only one version, identified by dwInVersion = 1, is currently defined.

```
typedef
```

```
[switch_type(DWORD)]
union {
  [case(1)]
    DRS_MSG_REPLICA_DEMOTIONREQ_V1 V1;
} DRS_MSG_REPLICA_DEMOTIONREQ;
```

V1: Version 1 request. Currently only one version is defined.

4.1.14.1.2 DRS_MSG_REPLICA_DEMOTIONREQ_V1

The **DRS_MSG_REPLICA_DEMOTIONREQ_V1** structure defines a request message sent to the **IDL_DRSReplicaDemotion** method.

```
typedef struct {
  DWORD dwFlags;
  UUID uuidHelperDest;
  DSNAME* pNC;
} DRS_MSG_REPLICA_DEMOTIONREQ_V1;
```

dwFlags: Zero or more of the following bit flags:

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

X: Unused. MUST be zero and ignored.

T (DS_REPLICA_DEMOTE_TRY_ALL_SRCS): MUST be set.

uuidHelperDest: MUST be NULL GUID.

pNC: The **DSNAME** of the NC to replicate off.

4.1.14.1.3 DRS_MSG_REPLICA_DEMOTIONREPLY

The **DRS_MSG_REPLICA_DEMOTIONREPLY** union defines the response messages received from the **IDL_DRSReplicaDemotion** method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

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

V1: Version 1 reply.

4.1.14.1.4 DRS_MSG_REPLICA_DEMOTIONREPLY_V1

The **DRS_MSG_REPLICA_DEMOTIONREPLY_V1** structure defines a response message received from the **IDL_DRSReplicaDemotion** method.

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

dwOpError: The Win32 error code, as specified in [MS-ERREF] section 3.0.

4.1.14.2 Method-Specific Abstract Types and Procedures

4.1.14.2.1 ReplicationPartners()

```
procedure ReplicationPartners(nc: DSNAME): sequence of DSNAME
```

The DC D executing this procedure hosts a portion of some forest F. This procedure computes the set of all DCs in F that host the specified NC, excluding D. It returns this set as a sequence in an arbitrary order.

4.1.14.2.2 BindToDSA()

```
procedure BindToDSA(dsa: DSNAME): DRS_HANDLE
```

Procedure BindToDSA() establishes an RPC connection to the target DC represented by its DSA object. It also performs the **IDL_DRSBind**() call. Returns the RPC handle on success or null on failure.

4.1.14.2.3 UnbindFromDSA()

```
procedure UnbindFromDSA(hDRS: DRS_HANDLE)
```

Procedure UnbindFromDSA() closes the RPC connection that was established by the BindToDSA procedure.

4.1.14.2.4 AbandonAllFSMORoles()

```
procedure AbandonAllFSMORoles(nc: DSNAME): DWORD
```

Procedure AbandonAllFSMORoles() abandons any FSMO roles represented in the supplied NC that are held by this DC. The new holder of the FSMO roles is arbitrary. AbandonAllFSMORoles returns a Win32 error value.

```
targetDSAs: sequence of DSNAME
fsmoContainer: DSNAME
ret: DWORD
bGivenAway: boolean
```

```
i: integer
msgReq: DRS_MSG_GETCHGREQ_V8
msgUpd: DRS_MSG_GETCHGREPLY_V6
hDRS: DRS_HANDLE

if nc = ConfigNC() then
  /* check domain naming FSMO role */
  fsmoContainer := DescendantObject(ConfigNC(), "CN=Partitions,")
else if nc = SchemaNC() then
  /* check schema master FSMO role */
  fsmoContainer := SchemaNC()
else
  /* application NCs don't hold FSMOs */
  return ERROR_SUCCESS
endif

/* check if we hold the fsmo */
if fsmoContainer!fSMORoleOwner ≠ DSAObj() then
  /* we don't own the role! All's well */
  return ERROR_SUCCESS
endif

/* yes, we own the role! Let's give it away */
bGivenAway := false
targetDSAs := ReplicationPartners(nc)
i := 0
while not bGivenAway
  if i ≥ targetDSAs.length then
    /* no more replication partners that would take our FSMO! */
    return ERROR_DS_UNABLE_TO_SURRENDER_ROLES
  endif
  hDRS := BindToDSA(possibleTargetDSAs[i])
  if hDRS ≠ null then
    /* the targetDSA appears to be up. Let's try to transfer the
     * role.  This operation is not described in this document. */
    ret = Perform a server-to-server call
          to abandon the held FSMO role
    if ret = ERROR_SUCCESS then
      /* yes! We got rid of it */
      bGivenAway := true
    endif
    UnbindFromDSA(hDRS)
  endif
  i := i + 1
endwhile
/* if we got here, then we managed to give away the role */
return ERROR_SUCCESS
```

4.1.14.2.5 ReplicateOffChanges()

```
procedure ReplicateOffChanges(nc: DSNAME): DWORD
```

Procedure ReplicateOffChanges() replicates all local changes in the NC to a randomly selected replication partner.

```
targetDSAs: sequence of DSNAME
ret: DWORD
bReplicated: boolean
i: integer
msgSyncReq: DRS_MSG_REPSYNC_V1
msgAddReq: DRS_MSG_REPADD_V2
hDRS: DRS_HANDLE
bReplicated := false
targetDSAs := ReplicationPartners(nc)
i := 0
while not bReplicated
  if i ≥ targetDSAs.length then
    /* no more replication partners that host the NC! */
    return ERROR_DS_CANT_FIND_DSA_OBJ
  endif
  hDRS := BindToDSA(possibleTargetDSAs[i])
  if hDRS ≠ null then
    /* the targetDSA appears to be up. Let's try to replicate to
     * it */
    /* Invoke IDL_DRSReplicaSync to get changes from us */
    msgSyncReq.pszDsaSrc := NetworkAddress of targetDSA
    msgSyncReq.uuidDsaSrc := dc.serverGuid
    msgSyncReq.pNC := ADDR(nc)
    msgSyncReq.ulOptions := DRS_WRIT_REP
    ret := IDL_DRSReplicaSync(hDRS, &msgSyncReq, 1)
    if ret = ERROR_DS_DRA_NO_REPLICA then
      /* the targetDSA does not currently have replication agreement
         (repsFrom) with this DC. Tell it to add one */
      msgAddReq.pNC := ADDR(nc)
      msgAddReq.pszSourceDsaAddress := NetworkAddress of this DC
      msgAddReq.ulOptions := DRS_WRIT_REP
      ret := IDL_DRSReplicaAdd(hDRS, &msgAddReq, 2)
    endif
    UnbindFromDSA(hDRS)
    if ret = ERROR_SUCCESS then
      /* we did it! */
      bReplicated := true
    endif
  endif
  i := i + 1
endwhile
/* if we got here, then we successfully replicated off our changes */
return ERROR_SUCCESS
```

4.1.14.3 Server Behavior of the IDL_DRSReplicaDemotion Method

Informative summary of behavior: For a given NC, this method initiates server-to-server replication to replicate out any changes that had not previously replicated out. It also abandons any NC-specific FSMO roles that are owned by this DC. This function accomplishes nothing when the DC being demoted is the last DC in the forest.

```
ULONG
IDL_DRSReplicaDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
```

```
        DRS_MSG_REPLICA_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_REPLICA_DEMOTIONREPLY* pmsgOut
    )
msgIn: DRS_MSG_REPLICA_DEMOTIONREQ_V1
ret: DWORD
nc: DSNAME
if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgIn^.V1
if  msgIn.pNC = null or
    msgIn.dwFlags ≠ DS_REPLICA_DEMOTE_TRY_ALL_SRCS then
  return ERROR_INVALID_PARAMETER
endif
if not IsMemberOfBuiltinAdminGroup() then
  /* only BA is allowed to demote an AD/LDS service */
  return ERROR_ACCESS_DENIED
endif
nc := msgIn.pNC^
ret := AbandonAllFSMORoles(nc)
if ret = ERROR_SUCCESS then
  ret := ReplicateOffChanges(nc)
endif
if ret = ERROR_SUCCESS then
  /* mark instanceType as going and not coming */
  nc!instanceType := nc!instanceType + {IT_NC_GOING} - {IT_NC_COMING}
  /* remove any repsFrom */
  nc!repsFrom := null
endif
pmsgOut^.dwOpError := ret
pdwMsgOut^ := 1
return ERROR_SUCCESS
```

4.1.15 IDL_DRSReplicaModify (Opnum 7)

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

```
ULONG IDL_DRSReplicaModify(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwVersion,
  [in, ref, switch_is(dwVersion)]
    DRS_MSG_REPMOD* pmsgMod
);
```

hDrs: RPC context handle returned by **IDL_DRSBind**.

dwVersion: Version of the request message.

pmsgMod: Pointer to the request message.

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

4.1.15.1 Method-Specific Concrete Types

4.1.15.1.1 DRS_MSG_REPMOD

The **DRS_MSG_REPMOD** union defines the request messages for the **IDL_DRSReplicaModify** method. Only one version, identified by dwVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.15.1.2 DRS_MSG_REPMOD_V1

The **DRS_MSG_REPMOD_V1** structure defines a request message for the **IDL_DRSReplicaModify** method.

```
typedef struct {
  [ref] DSNAME* pNC;
  UUID uuidSourceDRA;
  [unique, string] char* pszSourceDRA;
  REPLTIMES rtSchedule;
  ULONG ulReplicaFlags;
  ULONG ulModifyFields;
  ULONG ulOptions;
} DRS_MSG_REPMOD_V1;
```

pNC: Pointer to DSName of the root of an NC replica on the server.

uuidSourceDRA: DSA GUID.

pszSourceDRA: Transport specific NetworkAddress of a DC.

rtSchedule: Periodic replication schedule.

ulReplicaFlags: **DRS_OPTIONS** flags for the repsFrom value.

ulModifyFields: Fields to update.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
UF	UA	US	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X: Unused. MUST be zero and ignored.

UF (DRS_UPDATE_FLAGS): Updates the flags associated with the server.

UA (DRS_UPDATE_ADDRESS): Updates the transport specific address associated with the server.

US (DRS_UPDATE_SCHEDULE): Updates the replication schedule associated with the server.

ulOptions: **DRS_OPTIONS** flags for execution of this method.

4.1.15.2 Server Behavior of the IDL_DRSReplicaModify Method

Informative summary of behavior: The server replaces fields in the repsFrom of the specified NC replica. If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously. The client must include DRS_WRIT_REP in ulOptions if the specified NC replica is a full replica. The server alters timeLastSuccess and consecutiveErrors and optionally replaces (as specified by ulModifyFields) serverAddress, schedule, and replicaFlags in repsFrom with the corresponding value from pszSourceDRA, rtSchedule, and ulReplicaFlags.

```
ULONG
IDL_DRSReplicaModify(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)]
        DRS_MSG_REPMOD *pmsgMod);
options: DRS_OPTIONS
nc: DSName
rf: RepsFrom
msgIn: DRS_MSG_REPMOD_V1
/* Validate the version */
if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgMod^.V1
/* Validate input parameters */
if msgIn.pNC = null
   or msgIn.pNC^ = ""
   or (msgIn.pszSourceDRA = null
       and msgIn.uuidSourceDRA = null)
   or (DRS_UPDATE_ADDRESS in {msgIn.ulModifyFields}
       and (msgIn.pszSourceDRA = null
            or msgIn.pszSourceDRA = ""))
   or (DRS_UPDATE_SCHEDULE in {msgIn.ulModifyFields}
       and msgIn.rtSchedule = null)
   or {msgIn.ulmodifyFields} -
      {DRS_UPDATE_ADDRESS, DRS_UPDATE_SCHEDULE, DRS_UPDATE_FLAGS}
      ≠ {} then
```

```
  return ERROR_INVALID_PARAMETER
endif
/* Validate the specified NC */
options := msgIn.ulOptions
nc := msgIn.pNC^
if not ObjExists(nc) then
  return ERROR_DS_DRA_BAD_DN
endif
if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
  return ERROR_DS_DRA_ACCESS_DENIED
endif
if DRS_ASYNC_OP in options then
  Asynchronous Processing: Initiate a logical thread of control
   to process the remainder of this request asynchronously
  return 0
endif
/* Validate the specified NC's writability. */
if (IT_WRITE in nc!instanceType) ≠ (DRS_WRIT_REP in options) then
  return ERROR_DS_DRA_BAD_NC
endif

/* Find the specified repsFrom. */
rf := select one v from nc!repsFrom
      where (v.serverAddress = msgIn.pszDsaSrc)
if rf = null then
  return ERROR_DS_DRA_NO_REPLICA
endif
/* Update the specified repsFrom. */
nc!repsFrom := nc!repsFrom - {rf}
rf.timeLastSuccess := current time
rf.consecutiveFailures := 0
if  DRS_UPDATE_ADDRESS in {msgIn.ulModifyFields} then
  rf.serverAddress := msgIn.pszSourceDRA
endif
if DRS_UPDATE_SCHEDULE in {msgIn.ulModifyFields} then
  rf.schedule := msgIn.rtSchedule
endif
if DRS_UPDATE_FLAGS in {msgIn.ulModifyFields} then
  rf.replicaFlags := msgIn.ulReplicaFlags
endif
nc!repsFrom := nc!repsFrom + {rf}
return 0
```

4.1.16 IDL_DRSReplicaSync (Opnum 2)

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

```
ULONG IDL_DRSReplicaSync(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwVersion,
  [in, ref, switch_is(dwVersion)]
    DRS_MSG_REPSYNC* pmsgSync
```

```
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwVersion: Version of the request message.

pmsgSync: Pointer to the request message.

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

4.1.16.1 Method-Specific Concrete Types

4.1.16.1.1 DRS_MSG_REPSYNC

The **DRS_MSG_REPSYNC** union defines the request messages sent to the **IDL_DRSReplicaSync** method. Only one version, identified by dwVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.16.1.2 DRS_MSG_REPSYNC_V1

The **DRS_MSG_REPSYNC_V1** structure defines a request message sent to the **IDL_DRSReplicaSync** method.

```
typedef struct {
  [ref] DSNAME* pNC;
  UUID uuidDsaSrc;
  [unique, string] char* pszDsaSrc;
  ULONG ulOptions;
} DRS_MSG_REPSYNC_V1;
```

pNC: Pointer to DSName of the root of an NC replica on the server.

uuidDsaSrc: A DSA GUID.

pszDsaSrc: Transport specific NetworkAddress of a DC.

ulOptions: **DRS_OPTIONS** flags.

4.1.16.2 Server Behavior of the IDL_DRSReplicaSync Method

Informative summary of behavior: The server begins replication by sending a server-to-server replication request to the specified DC. If ulOptions contains DRS_ASYNC_OP, the server performs this operation asynchronously.

```
ULONG
IDL_DRSReplicaSync(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)]
        DRS_MSG_REPSYNC *pmsgSync);

options: DRS_OPTIONS
nc: DSName
rf: sequence of RepsFrom
msgIn: DRS_MSG_REPSYNC_V1

/* Validate the version */
if dwVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgSync^.V1

/* Validate input params */
options := msgIn.ulOptions
if msgIn.pNC = null
   or (not DRS_SYNC_ALL in options
       and msgIn.uuidDsaSrc = null
       and msgIn.pszDsaSrc = null) then
  return ERROR_INVALID_PARAMETER
endif

/* Validate the specified NC. */
nc := msgIn.pNC^
if not ObjExists(nc) then
  return ERROR_DS_DRA_BAD_DN
endif

if AccessCheckCAR(nc, DS-Replication-Synchronize) then
  return ERROR_DS_DRA_ACCESS_DENIED
endif

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

rf := select all v in nc!repsFrom
      where DRS_SYNC_ALL in options
            or (DRS_SYNC_BYNAME in options
                and v.uuidDsa = msgIn.pszDsaSrc)
            or (not DRS_SYNC_BYNAME in options
                and v.uuidDsa = msgIn.uuidDsaSrc)
if rf = null then
  return ERROR_DS_DRA_NO_REPLICA
endif
```

```
foreach r in rf
  PerformReplication(nc, r)
endfor

return 0
```

4.1.17 IDL_DRSReplicaVerifyObjects (Opnum 22)

The **IDL_DRSReplicaVerifyObjects** method verifies the existence of objects in an NC replica by comparing against a replica of the same NC on a reference DC, optionally deleting any objects that do not exist on the reference DC. This method is used only to diagnose, monitor, and manage the replication implementation. The structures requested and returned through this method MAY have meaning to peer DCs and applications, but are not required for interoperation with Windows clients.

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

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwVersion: Version of the request message.

pmsgVerify: Pointer to the request message.

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

4.1.17.1 Method-Specific Concrete Types

4.1.17.1.1 DRS_MSG_REPVERIFYOBJ

The **DRS_MSG_REPVERIFYOBJ** union defines the request messages sent to the **IDL_DRSReplicaVerifyObjects** method. Only one version, identified by dwVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.17.1.2 DRS_MSG_REPVERIFYOBJ_V1

The **DRS_MSG_REPVERIFYOBJ_V1** structure defines a request message sent to the **IDL_DRSReplicaVerifyObjects** method.

```
typedef struct {
  [ref] DSNAME* pNC;
  UUID uuidDsaSrc;
  ULONG ulOptions;
} DRS_MSG_REPVERIFYOBJ_V1;
```

pNC: NC to verify.

uuidDsaSrc: objectGUID of the nTDSDSA object for the reference DC.

ulOptions: 0 to expunge each object that is not verified, or 1 to log an event identifying each such object.

4.1.17.2 Method-Specific Abstract Types and Procedures

4.1.17.2.1 GetRemoteUTD

```
procedure GetRemoteUTD(
 dsa: DSName, nc: DSName): UPTODATE_VECTOR_V1_EXT
```

Remotely retrieves the **UPTODATE_VECTOR_V1_EXT** for the NC with the DSName nc from the DC whose nTDSDSA object has the DSName dsa. The server-to-server replication implementation that performs this operation is not included in this document.

4.1.17.2.2 ObjectExistsAtDC

```
procedure ObjectExistsAtDC(o object, dsa: DSName): boolean
```

Executes the tasks necessary to verify that the object o exists on the DC whose nTDSDSA object has the DSName dsa. If the object exists, the procedure returns true; otherwise, the procedure returns false.

4.1.17.3 Server Behavior of the IDL_DRSReplicaVerifyObjects Method

Informative summary of behavior: Let N be the NC pNC^, and let the reference DC be the DC corresponding to the nTDSDSA object uuidDsaSrc.

For the purposes of this method, an object *exists* within a NC replica if it is either an object or a tombstone.

Let S be the set of objects that exists in N at the server running IDL_DRSReplicaVerifyObjects at the time IDL_DRSReplicaVerifyObjects begins processing. Let the set S' be S minus the members of S that have never existed in N at the reference DC when IDL_DRSReplicaVerifyObjects begins processing. The members of (S - S') must be objects recently added to N on the server, since otherwise they would have replicated to the reference DC. The set S' is computable using the replUpToDateVector for N at the server and at the reference DC.

For each object *o* in S' that does not exist in N at the reference DC while IDL_DRSReplicaVerifyObjects is processing, either expunge *o* at the server (if ulOptions = 0) or log an administrator-visible event at the server (if ulOptions = 1).

If an object goes out of existence in N at the reference DC during processing of IDL_DRSReplicaVerifyObjects, IDL_DRSReplicaVerifyObjects might or might not Expunge/log the object at the server.

```
ULONG IDL_DRSReplicaVerifyObjects(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwVersion,
  [in, ref, switch_is(dwVersion)]
      DRS_MSG_REPVERIFYOBJ *pmsgVerify)

msgIn: DRS_MSG_REPVERIFYOBJ_V1
nc, refDsa, o: DSName
uTDServer, uTDRef, uTDMerge: UPTODATE_VECTOR_V1_EXT
sPrime: set of DSName

/* Perform input validation and access check */
if dwInVersion ≠ 0x1 then
  return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgVerify^.V1
if msgIn.pNC = null then
  return ERROR_DS_DRA_BAD_NC
endif
nc := msgIn.pNC^
if not FullReplicaExists(nc) and
    not PartialGCReplicaExists(nc) then
  return ERROR_DS_DRA_BAD_NC
endif
refDsa := select one object o from subtree ConfigNC() where
    o!objectGUID = msgIn.uuidDsaSrc and nTDSDSA in o!objectClass
if refDsa = null then
  return  ERROR_DS_DRA_INVALID_PARAMETER
endif
if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
  return ERROR_DS_DRA_ACCESS_DENIED
endif

/* Compute the set S' */
uTDServer := nc!replUpToDateVector
uTDRef := GetRemoteUTD(refDsa, nc)
if uTDRef = null then
  return ERROR_DS_DRA_INVALID_PARAMETER
endif
uTDMerge := MergeUTD(uTDServer, uTDRef)

sPrime := select all objects o from subtree-ts-included nc where
    StampLessThanOrEqualUTD(AttrStamp(o, whenCreated), uTDMerge)

/* Process the set S' */
for each o in sPrime
  if ObjectExistsAtDC(o, refDSA) then
    if msgIn.ulOptions = 0 then
      Expunge(o)
    else if msgIn.ulOptions = 1 then
      Log a message: o exists on server but does not exist on refDsa
    endif
endfor
```

```
return 0
```

4.1.18 IDL_DRSUnbind (Opnum 1)

The **IDL_DRSUnbind** method destroys a context handle previously created by the **IDL_DRSBind** method.

```
ULONG IDL_DRSUnbind(
  [in, out, ref] DRS_HANDLE* phDrs
);
```

phDrs: Pointer to the RPC context handle returned by the **IDL_DRSBind** method. The value is set to NULL on return.

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

4.1.18.1 Server Behavior of the IDL_DRSUnbind Method

Informative summary of behavior: The server releases any resources associated with the context handle, making the context handle unusable by the client. The server sets phDrs to null.

```
ULONG
IDL_DRSUnbind(
    [in, out, ref] DRS_HANDLE *phDrs)
phDrs^ := null
return 0
```

4.1.19 IDL_DRSUpdateRefs (Opnum 4)

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

```
ULONG IDL_DRSUpdateRefs(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwVersion,
  [in, ref, switch_is(dwVersion)]
    DRS_MSG_UPDREFS* pmsgUpdRefs
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwVersion: Version of the request message.

pmsgUpdRefs: Pointer to the request message.

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

4.1.19.1 Method-Specific Concrete Types

4.1.19.1.1 DRS_MSG_UPDREFS

The **DRS_MSG_UPDREFS** union defines the request message versions sent to the **IDL_DRSUpdateRefs** method. Only one version, identified by dwVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.19.1.2 DRS_MSG_UPDREFS_V1

The **DRS_MSG_UPDREFS_V1** structure defines a request message sent to the **IDL_DRSUpdateRefs** method.

```
typedef struct {
  [ref] DSNAME* pNC;
  [ref, string] char* pszDsaDest;
  UUID uuidDsaObjDest;
  ULONG ulOptions;
} DRS_MSG_UPDREFS_V1;
```

pNC: Pointer to DSName of the root an NC replica on the server.

pszDsaDest: Transport specific NetworkAddress of a DC.

uuidDsaObjDest: A DSA GUID.

ulOptions: The **DRS_OPTIONS** that control the update.

4.1.19.2 Server Behavior of the IDL_DRSUpdateRefs Method

Informative summary of behavior: If ulOptions contains DRS_ADD_REF, the server adds a value to the repsTo of the specified NC replica; if ulOptions contains DRS_DEL_REF, the server deletes a value. These options may be combined to replace an existing repsTo value with a new value; if a corresponding value does not already exist, this is the same as if ulOptions contained DRS_ADD_REF but not DRS_DEL_REF. The client includes DRS_WRIT_REP in ulOptions if the specified NC replica is writable. The client specifies at least one of pszDsaDest and uuidDsaObjDest to identify the value to be added or removed. If ulOptions contains DRS_ASYNC_OP, the server processes the request asynchronously. If the server adds a value to repsTo, the value has ulReplicaFlags equal to ulOptions ∩ {DRS_WRIT_REP}.

```
ULONG IDL_DRSUpdateRefs(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwVersion,
```

```
  [in, ref, switch_is(dwVersion)] DRS_MSG_UPDREFS *pmsgUpdRefs);
msgIn: DRS_MSG_UPDREFS_V1
options: DRS_OPTIONS
err: DWORD
nc: DSName
rt: RepsTo
if dwInVersion ≠ 1 then
  return ERROR_INVALID_PARAMETER
endif
msgIn := pmsgUpdRefs^.V1
options := msgIn.ulOptions
if msgIn.pNC = null or
    (msgIn.pszDsaDest = null and
     msgIn.uuidDsaObjDest = null) or
    (options ∩ {DRS_ADD_REF, DRS_DEL_REF} = null)
  return ERROR_INVALID_PARAMETER
endif
nc := msgIn.pNC^
if not ObjExists(nc) then
  return ERROR_DS_DRA_BAD_DN
endif
if (IT_WRITE in nc!instanceType) ≠
   (DRS_WRIT_REP in options) then
  return ERROR_DS_DRA_BAD_NC
endif
if not AccessCheckCAR(nc, DS-Replication-Manage-Topology) then
  return ERROR_DS_DRA_ACCESS_DENIED
endif
if DRS_ASYNC_OP in options then
  Asynchronous Processing: Initiate a logical thread of control
   to process the remainder of this request asynchronously
  return 0
endif
if DRS_DEL_REF in options then
  rt := select one v from nc!repsTo where
      (v.naDsa = msgIn.pszDsaDest or
        v.uuidDsa = msgIn.uuidDsaObjDest)
  if rt = null then
    err := ERROR_DS_DRA_REF_NOT_FOUND
  else
    nc!repsTo := nc!repsTo - {rt}
    err := 0
  endif
endif
/* If DRS_DEL_REF and DRS_ADD_REF are both specified, the return
 * value is that associated with the DRS_ADD_REF. */
if DRS_ADD_REF in options then
  rt := select one v from nc!repsTo where
      (v.naDsa = msgIn.pszDsaDest or
        v.uuidDsa = msgIn.uuidDsaObjDest)
  if rt = null then
    rt.naDsa := msgIn.pszDsaDest
    rt.uuidDsa := msgIn.uuidDsaObjDest
    rt.options := options ∩ {DRS_WRIT_REP}
    rt.timeLastAttempt := 0
    rt.timeLastSuccess := 0
    rt.consecutiveFailures := 0
    rt.resultLastAttempt := 0
```

```
      nc!repsTo := nc!repsTo + {rt}
      err := 0
    else
      err := ERROR_DS_DRA_REF_ALREADY_EXISTS
    endif
  endif
  return err
```

4.1.19.3 Examples of the IDL_DRSUpdateRefs Method

4.1.19.3.1 Initial State

The repsTo attribute on the NC root object for domain NC CONTOSO.COM on DC1 does not contain a value for DC2:

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

Result <0>: (null)

Matched DNs:

Getting 1 entries:

>> Dn: DC=CONTOSO,DC=COM

4.1.19.3.2 Client Request

A client invokes the method IDL_DRSUpdateRefs against DC1, with the following parameters (**DRS_HANDLE** to DC1 omitted):

- dwVersion = 1
- pmsgUpdRefs = 0x0006fe08 ; Pointer to the following structure:
- pNC: 0x0008cdb8 _DSNAME DC=CONTOSO, DC=COM
- pszDsaDest : "DC2.CONTOSO.COM"
- uuidDsaObjDest: _GUID {0e9240b6-b3dd-4c86-92e3-2757e56ff0b3}
- ulOptions: DRS_WRIT_REP | DRS_ADD_REF

4.1.19.3.3 Server Response

Return code of 0.

4.1.19.3.4 Final State

The repsTo attribute on the NC root object for domain NC CONTOSO.COM on DC1 contains one value:

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

Result <0>: (null)

Matched DNs:

Getting 1 entries:

>> Dn: DC=CONTOSO,DC=COM

1> repsTo: dwVersion = 1,

- V1.cb: 229, V1.cConsecutiveFailures: 0, V1.timeLastSuccess: 0,
- V1.timeLastAttempt: 0, V1.ulResultLastAttempt: 0x0,
- V1.cbOtherDraOffset: 216,
- V1.cbOtherDra: 13, V1.ulReplicaFlags: 0x10, V1.rtSchedule: <ldp:skipped>,
- V1.usnvec.usnHighObjUpdate: 0, V1.usnvec.usnHighPropUpdate: 0,
- V1.uuidDsaObj: 0e9240b6-b3dd-4c86-92e3-2757e56ff0b3
- V1.uuidInvocId: 00000000-0000-0000-0000-000000000000
- V1.uuidTransportObj: 00000000-0000-0000-0000-000000000000
- V1.mtx_address: DC2.CONTOSO.COM
- V1.cbPASDataOffset: 0 V1.PasData: version = -1, size = -1, flag = -1;

4.1.20 IDL_DRSWriteSPN (Opnum 13)

The **IDL_DRSWriteSPN** method updates the set of service principal names (SPNs) on an object.

```
ULONG IDL_DRSWriteSPN(
  [in, ref] DRS_HANDLE hDrs,
  [in] DWORD dwInVersion,
  [in, ref, switch_is(dwInVersion)]
    DRS_MSG_SPNREQ* pmsgIn,
  [out, ref] DWORD* pdwOutVersion,
  [out, ref, switch_is(*pdwOutVersion)]
    DRS_MSG_SPNREPLY* pmsgOut
);
```

hDrs: RPC context handle returned by the **IDL_DRSBind** method.

dwInVersion: Version of the request message. MUST be set to 1, as that is the only version currently supported.

pmsgIn: Pointer to the request message.

pdwOutVersion: Pointer to the version of the response message. The value will always be 1, as that is the only version currently supported.

pmsgOut: Pointer to the response message.

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

4.1.20.1 Method-Specific Concrete Types

4.1.20.1.1 DRS_MSG_SPNREQ

The **DRS_MSG_SPNREQ** union defines the request messages sent to the **IDL_DRSWriteSPN** method. Only one version, identified by dwInVersion = 1, is currently defined.

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

V1: Version 1 request.

4.1.20.1.2 DRS_MSG_SPNREQ_V1

The **DRS_MSG_SPNREQ_V1** structure defines a request message sent to the **IDL_DRSWriteSPN** method.

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

operation: SPN operation to perform. MUST be one of the DS_SPN_OPERATION values.

flags: MUST be 0.

pwszAccount: DN of the object to modify.

cSPN: Count of items in the **rpwszSPN** array.

rpwszSPN: SPN values.

4.1.20.1.3 DRS_MSG_SPNREPLY

The **DRS_MSG_SPNREPLY** union defines the response messages received from the **IDL_DRSWriteSPN** method. Only one version, identified by pdwOutVersion^ = 1, is currently defined.

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

V1: Version 1 response.

4.1.20.1.4 DRS_MSG_SPNREPLY_V1

The **DRS_MSG_SPNREPLY_V1** structure defines a response message received from the **IDL_DRSWriteSPN** method.

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

retVal: 0 or a Windows error code.

4.1.20.1.5 DS_SPN_OPERATION

The **DS_SPN_OPERATION** type indicates the operation to perform.

This type is declared as follows:

```
typedef DWORD DS_SPN_OPERATION;
```

Must be one of the following values:

Value	Meaning
DS_SPN_ADD_SPN_OP (0x00000000)	Adds the specified values to the existing set of SPNs.
DS_SPN_REPLACE_SPN_OP (0x00000001)	Removes all the existing SPNs, then adds the specified values. If the set of specified values is empty (cSPN is zero) no values are added.
DS_SPN_DELETE_SPN_OP (0x00000002)	Removes all the existing SPNs.

4.1.20.2 Server Behavior of the IDL_DRSWriteSPN Method

Informative summary of behavior: This method updates the servicePrincipalName attribute of an object. The values of this multi-valued attribute are called service principal names (SPNs). The method does one of three things:

- Adds a non-empty set of SPNs to the object's servicePrincipalName. If a member of the set is already present on the object's servicePrincipalName, it is ignored.
- Removes all current values from the object's servicePrincipalName, then adds a (possibly empty) set of SPNs to the object's servicePrincipalName.

- Removes a non-empty set of SPNs from the object's servicePrincipalName. If a member of the set is not present on the object's servicePrincipalName, it is ignored.

The effect of this method can be achieved by an LDAP Modify operation to the servicePrincipalName attribute of an object. Some manipulations of the servicePrincipalName attribute that can't be performed using this method can be performed using LDAP Modify. For example, an LDAP Modify can remove one specific SPN from the servicePrincipalName attribute while adding another SPN to the servicePrincipalName attribute in the same transaction; IDL_DRSWriteSPN can't do this.

```
ULONG
IDL_DRSWriteSPN(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_SPNREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_SPNREPLY *pmsgOut);
accountDN: unicodestring
account: DSName
operation: DS_SPN_OPERATION
cSPN: integer
spnSet: set of unicodestring
instanceName: unicodestring
pdwOutVersion^ := 1
pmsgOut^.V1.retVal := 0
/* Input parameter validation */
if dwInVersion ≠ 1 then
  pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
  return ERROR_INVALID_PARAMETER
endif
/* RODCs do not perform originating updates */
if AmIRODC() then
  pmsgOut^.V1.retVal := ERROR_DS_REFERRAL
  return ERROR_DS_REFERRAL
endif
accountDN := pmsgIn^.V1.pwszAccount
operation := pmsgIn^.V1.operation
cSPN := pmsgIn^.V1.cSPN
spnSet := pmsgIn^.V1.rpwszSPN
if accountDN = null or accountDN = "" then
  pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
  return ERROR_INVALID_PARAMETER
endif
if not operation in [DS_SPN_ADD_SPN_OP .. DS_SPN_DELETE_SPN_OP] then
  pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
  return ERROR_INVALID_PARAMETER
endif
/* DS_SPN_REPLACE_SPN_OP permits 0 SPNs to be specified (meaning
 * "delete all SPNs"). Other operations require >=1 SPNs to be
 * specified. */
if (operation ≠ DS_SPN_REPLACE_SPN_OP) and (cSPN = 0) then
  pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
  return ERROR_INVALID_PARAMETER
endif
/* The empty string is an invalid SPN. */
foreach spn in spnSet
  if spn = null or spn = "" then
```

```
    pmsgOut^.V1.retVal := ERROR_INVALID_PARAMETER
    return ERROR_INVALID_PARAMETER
  endif
endfor
account := GetDSNameFromDN(accountDN);
if not ObjExists(account) then
  pmsgOut^.V1.retVal := ERROR_DS_NO_SUCH_OBJECT
  return ERROR_DS_NO_SUCH_OBJECT
endif
/* Perform access checks */
if not AccessCheckWriteToSpnAttribute(account, spnSet) then
  pmsgOut^.V1.retVal := ERROR_ACCESS_DENIED
  return ERROR_ACCESS_DENIED
endif
if (operation = DS_SPN_DELETE_SPN_OP) then
  /* Remove specified SPNs */
  foreach spn in spnSet
    if spn in account!servicePrincipalName then
      account!servicePrincipalName :=
          account!servicePrincipalName - {spn}
    endif
  endfor
  return 0
endif
if (operation = DS_SPN_ADD_SPN_OP) then
  /* Add specified SPNs */
  foreach spn in spnSet
    account!servicePrincipalName :=
        account!servicePrincipalName + {spn}
  endfor
  return 0
endif
/* Must be DS_SPN_REPLACE_SPN_OP.
 * Remove all existing SPNs, then add in the specified SPNs. */
account!servicePrincipalName := {null}
foreach spn in spnSet
  account!servicePrincipalName :=
      account!servicePrincipalName + {spn}
endfor
return 0
```

5 Common Data Types, Variables, and Procedures

This section contains types that are used by drsuapi methods, or types used in this specification but normatively specified in other specifications. It also contains types and procedures used only within the specification. This section is arranged in order by type or procedure name.

The specification of message syntax in this section is normative for syntax only. The behavior descriptions for types representing messages are informative. Consult the behavior description for each method that uses a type for the normative specification of behavior related to that type.

"Hand-marshaled" types are types passed as BLOBs through RPC and types stored as blobs in the directory. Any type that is "hand-marshaled" is specified pictorially in this section to emphasize the layout of any multi-byte quantities it contains. The layout is always little-endian. If a type is both "hand-marshaled" and marshaled by RPC, then an IDL specification of the type is given, in addition to the pictorial specification.

This specification uses the definitions of RPC base types. Additional data types used in this protocol are specified in this section.

Note that values of some types are marshaled by RPC as structures in some cases and as little-endian byte arrays in other cases. Where such cases exist, the structure is defined both in **MIDL** syntax and in a byte diagram, and the byte array cases are clearly identified so that big-endian architectures can perform the necessary byte swapping.

5.1 AbstractPTFromConcretePT

```
procedure  AbstractPTFromConcretePT(
  concretePrefixTable: SCHEMA_PREFIX_TABLE): PrefixTable
```

Informative summary of behavior: Translates **SCHEMA_PREFIX_TABLE** structure to abstract PrefixTable.

```
prefixTable: PrefixTable
schemaSignature: sequence of BYTE
i: DWORD

for i := 0 to (concretePrefixTable.PrefixCount - 1)
  prefixTable[i].prefixString :=
    concretePrefixTable. pPrefixTableEntry[i].prefix
  prefixTable[i].prefixIndex :=
    concretePrefixTable.pPrefixTableEntry[i].ndx
endfor

return concretePrefixTable
```

5.2 AccessCheckAttr

```
procedure AccessCheckAttr(
    dsName: DSName, attr: ATTRTYP, right: Right): boolean
```

Returns true if dsName identifies an object within an NC replica hosted by the server, and the client's security context has the access indicated by the access right **right** to the attribute attr on that object; returns false otherwise.

See [MS-ADTS] section 5.1.3 for the specification of this procedure.

See [MS-ADTS] section 5.1.3.2 for the list of symbolic names for access rights (for example, RIGHT_DS_WRITE_PROPERTY) and the numeric value of each.

5.3 AccessCheckCAR

```
procedure AccessCheckCAR(dsName: DSName; right: Right): boolean
```

Returns true if dsName identifies an object within an NC replica hosted by the server, and the client's security context has the access indicated by the **control access right** **right** on that object; returns false otherwise.

See [MS-ADTS] section 5.1.3 for the specification of this procedure.

See [MS-ADTS] section 5.1.3.2.1 for the list of symbolic names for control access rights (for example, DS-Replication-Manage-Topology) and the numeric value of each.

5.4 AccessCheckObject

```
procedure AccessCheckObject(dsName: DSName, right: Right): boolean
```

Returns true if dsName identifies an object within an NC replica hosted by the server, and the client's security context has the access indicated by the access right **right** on that object; returns false otherwise.

See [MS-ADTS] section 5.1.3 for the specification of this procedure.

See [MS-ADTS] section 5.1.3.2 for the list of symbolic names for access rights (for example, RIGHT_DS_DELETE_CHILD) and the numeric value of each.

5.5 AccessCheckWriteToSpnAttribute

```
procedure AccessCheckWriteToSpnAttribute(
  obj: DSName, spnSet: set of unicodestring) : boolean
```

Performs an access check to determine if the client security context has the right to modify the servicePrincipalName attribute of object obj with the SPN values specified in spnSet, taking into consideration both regular and extended write property rights.

```
if AccessCheckAttr(obj,
                   servicePrincipalName,
                   RIGHT_DS_WRITE_PROPERTY) then
  return true
else
  if AccessCheckAttr(obj,
                     servicePrincipalName,
                     RIGHT_DS_WRITE_PROPERTY_EXTENDED) then
    /* Extended write access permits the attribute to be written */
     * provided the proposed SPNs meet certain constraints. */

    foreach spn in spnSet
      if not Is2PartSPN(spn) then
```

```
        if (Is3PartSPN(spn) and IsDCAccount(obj)) then

          /* Three part SPNs are permitted for DC computer accounts */
          /* However, in addition to the constraints on 2 part SPNs,*/
          /* the service name must meet additional constraints */

          serviceName := GetServiceNameFromSPN(spn)
          if not IsValidServiceName(obj, serviceName)
            return false
          endif
        else
          return false
        endif
      endif

      instanceName := GetInstanceNameFromSPN(spn)
      if (instanceName != obj!dNSHostName) and
         (not instanceName + "$" = obj!sAMAccountName) and
         (not instanceName in obj!msDS-AdditionalDnsHostName) and
         (not instanceName + "$" in
             obj!msDS-AdditionalSamAccountName) then
        /* If this is a DC computer account */
        /* the instance name may be a GUID based dns host name */
        if IsDCAccount(obj) then
          if not IsGUIDBasedDNSName(obj, instanceName)then
            return false
          endif
        else
          return false
        endif
      endif
    endfor
    return true
  endif

  return false
endif
```

5.6 AmIRODC

```
procedure AmIRODC() : Boolean
```

This procedure returns true if the DC is an RODC.

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

5.7 AttributeStamp

An abstract type that contains information about the last originating update to an attribute. It is a tuple of the following:

- **dwVersion**: 32-bit integer. Set to 1 when a value for the attribute is set for the first time, incremented by 1 on each subsequent originating update.

- **timeChanged**: The date/time at which the last originating update was made.
- **uuidOriginating**: The invocation-id of the DC that performed the last originating update.
- **usnOriginating**: The update sequence number (USN) assigned to the last originating update by the DC that performed it.

Comparisons

Values of AttributeStamp are partially ordered. Let d be the result of x.dwVersion - y.dwVersion, cast as a 32-bit integer. For example, if x.dwVersion is 1 and y.dwVersion is 3, d is -2. If x.dwVersion is 5 and y.dwVersion is 0xFFFFFFFA, d is 11. Then given two stamps *x* and *y*, *x* is said to be "greater than" *y* if any of the following is true:

- x is not null and y is null
- d > 0
- d = 0 and x.timeChanged > y.timeChanged
- d = 0 and x.timeChanged = y.timeChanged and x.uuidOriginating > y.uuidOriginating

5.8 AttributeSyntax

An abstract type that represents an LDAP attribute syntax. The valid values are the names from the "LDAP Syntax Name" column of the table in section 5.12, for example, "Object(DS-DN)" and "Object(DN-Binary)".

5.9 AttrStamp

```
procedure AttrStamp(o: DSName, attr: ATTRTYP) : AttributeStamp
```

Returns the AttributeStamp for the attribute whose **ATTRTYP** is attr on the object whose DSName is o.

5.10 ATTRTYP

ATTRTYP is a concrete type for a compact representation of an **object identifier (OID)**.

This type is declared as follows:

```
typedef ULONG ATTRTYP;
```

Section 5.13 specifies the procedures that map between ATTRTYP and OID with the aid of a **SCHEMA_PREFIX_TABLE**.

5.11 AttrtypFromSchemaObj

```
procedure AttrtypFromSchemaObj(o: DSName): ATTRTYP
```

Given the dsname o of an attributeSchema or classSchema object, returns the **ATTRTYP** identifying this schema object on this DC.

```
if o!msDS-IntId ≠ null then
  return o!msDS-IntId
endif
if attributeSchema in o!objectClass then
  return MakeAttid(dc.prefixTable, o!attributeID)
else
  return MakeAttid(dc.prefixTable, o!governsID)
endif
```

5.12 Abstract Value Representations

The abstract data model utilizes a representation of data values that is used by LDAP, minus the BER encoding. Several of these syntaxes are adopted from [RFC2252].

The following table lists all the supported syntaxes and how they are represented in the model. Some syntaxes share an OID, so here the syntaxes are identified by name, not OID.

LDAP syntax name (OID)	[RFC2252] name	Reference section in [RFC2252] or in this document
Boolean (2.2.5.8)	Boolean	[RFC2252] section 6.4
Enumeration (2.5.5.9)	INTEGER	[RFC2252] section 6.16
Integer (2.5.5.9)	INTEGER	[RFC2252] section 6.16
LargeInteger (2.5.5.16)	INTEGER	[RFC2252] section 6.16
Object(Presentation-Address) (2.5.5.13)	Presentation Address	[RFC2252] section 6.28
Object(Replica-Link) (2.5.5.10)	Binary	[RFC2252] section 6.2
String(IA5) (2.5.5.5)	IA5 String	[RFC2252] section 6.15
String(Numeric) (2.5.5.6)	Numeric String	[RFC2252] section 6.23
String(Object-Identifier) (2.5.5.2)	OID	[RFC2252] section 6.25
String(Octet) (2.5.5.10)	Binary	[RFC2252] section 6.2
String(Printable) (2.5.5.5)	Printable String	[RFC2252] section 6.29
String(Unicode) (2.5.5.12)	Directory String	[RFC2252] section 6.10
String(UTC-Time) (2.5.5.11)	UTC Time	[RFC2252] section 6.31
String(Generalized-Time) (2.5.5.11)	Generalized Time	[RFC2252] section 6.14
Object(DS-DN) (2.5.5.1)	-	Section 5.12.1
Object(DN-String) (2.5.5.14)	-	Section 5.12.2

LDAP syntax name (OID)	[RFC2252] name	Reference section in [RFC2252] or in this document
Object(DN-Binary) (2.5.5.7)	-	Section 5.12.3
Object(Access-Point) (2.5.5.14)	-	Section 5.12.4
Object(OR-Name) (2.5.5.7)	-	Section 5.12.5
String(NT-Sec-Desc) (2.5.5.15)	-	Section 5.12.6
String(SID) (2.5.5.17)	-	Section 5.12.7
String(Teletex) (2.5.5.4)	-	Section 5.12.8

The LDAP syntaxes that are not defined in [RFC2252] are described in the following subsections.

5.12.1 Object(DS-DN)

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

<GUID=*guid_value*>;<SID=*sid_value*>;*dn*

where *guid_value* is the value of the object's objectGUID attribute, *sid_value* is the value of the object's objectSid attribute in its binary format (as specified in [MS-DTYP] section **2.4.2**), and *dn* is the string representation of a DN (as specified by [RFC2252] section 6.9, and further specified by [RFC2253]).

For reference to objects which do not have an objectSid, the format is as follows:

<GUID=*guid_value*>;*dn*

where *guid_value* and *dn* have the same meaning as in the previous case.

5.12.2 Object(DN-String)

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

S:*char_count*:*string_value*:*object_DN*

where *char_count* is the number of characters in the *string_value* string, and *object_DN* is an object reference in the format of Object(DS-DN).

5.12.3 Object(DN-Binary)

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

B:*char_count*:*binary_value*:*object_DN*

where *char_count* is the number of hexadecimal digits in *binary_value*, *binary_value* is the hexadecimal representation of a binary value, and *object_DN* is an object reference in the format of Object(DS-DN).

5.12.4 Object(Access-Point)

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

presentation_address#X500:*object_DN*

where *presentation_address* is a value encoded in the Object(Presentation-Address) syntax and *object_DN* is an object reference in the format of Object(DS-DN).

5.12.5 Object(OR-Name)

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

X400:*OR_Name*#X500:*object_DN*

where *OR_Name* is an X.400 O/R name in the format described in [RFC1327] section 4.1, and *object_DN* is an object reference in the format of Object(DS-DN).

Alternatively, the *OR_Name* portion may be omitted, in which case the value contains only the *object_DN*.

5.12.6 String(NT-Sec-Desc)

A value with this syntax contains a Windows security descriptor in self-relative binary form. The binary form is that of a SECURITY_DESCRIPTOR structure and is documented in [MS-DTYP] section **2.4.6**.

5.12.7 String(Sid)

A value with this syntax is a Windows security identifier (SID) in binary form. The binary form is that of a SID structure and is specified in [MS-DTYP] section **2.4.2**.

5.12.8 String(Teletex)

A value with this syntax is a UTF-8 string restricted to characters with values between 0x20 and 0x7e, inclusive.

5.13 ATTRTYP-to-OID Conversion

This section describes the prefix mapping mechanism that allows the one to one mapping between object identifiers (OIDs) and a 32-bit integer (**ATTRTYP**).

An OID can be represented in the binary form, with a basic encoding rule (BER) encoding scheme. The standard BER encoding consists of four components. Here only the third component (contents octets) is being used, while other components are omitted.

Note The BER encoding of an object identifier is described in [ITUX690] section 8.19. To avoid ambiguity, the non-encoded form of the OID is referred to as the original form in this section.

The prefix of an OID is the binary OID, excluding the last 1 or 2 bytes. If the number following the final "." in the original form of the OID is less than 128, only the last byte is excluded; otherwise, the last two bytes are excluded.

A **PrefixTable** is a sequence of tuples defined as follows:

```
type PrefixTable = sequence of [
  prefixString: unicodestring,
  prefixIndex: integer
]
```

where prefixString is the prefix of an OID and prefixIndex is an integer in the range [0 .. 0x0000ffff]. The integer prefixIndex is called the prefix index of prefixString. To allow one to one mappings between the prefix strings and the prefix indexes in the table, each prefixString MUST occur at most once in the table, and each prefixIndex MUST occur at most once in the table.

An **ATTRTYP** is a 32-bit unsigned integer. If attr is an **ATTRTYP**, define attr.upperWord to be the most significant 16 bits, and attr.lowerWord to be the least significant 16 bits.

The following types and helper procedures are used for mapping between OIDs and **ATTRTYP**.

```
procedure ToBinary(st: unicodestring) : sequence of BYTE
```

Convert a string to a binary OID representation. For example, "\x55\x06" is the binary OID \x55\x06.

```
procedure CatBinary(o: sequence of BYTE, b: BYTE) : sequence of BYTE
```

Concatenate a byte onto a binary OID. For example, \x02 concatenated onto \x55\x06 is \x55\x06\x02.

```
procedure ToStringOID(o: sequence of BYTE) : unicodestring
```

Convert a binary OID to its string representation, as described in [ITUX690] section 8.19; return null if the conversion fails. For example, the binary OID \x55\x06\x02 is converted to the OID string "2.5.6.2".

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

Convert an OID string representation to a binary OID as described in [ITUX690] section 8.19; return null if the conversion fails. For example, the OID string "2.5.6.2" is converted to the binary form \x55\x06\x02.

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

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

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

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

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

Set t[t.length].prefixString to o. Generate a random number between 0 and 65535 that is unique in the values of prefixIndex in t, and set t[t.length].prefixIndex to the generated random number. Increase t.length by one.

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

Convert a string to its integer representation. For example, "127" is 127. Strings with non-numeric characters are not defined for this procedure.

The following procedures are used for mapping between object identifiers and **ATTRTYP** representations.

```
procedure MakeAttid(var t: PrefixTable, o: OID): ATTRTYP
```

Informative summary of behavior: This procedure converts an OID to a corresponding **ATTRTYP** representation.

```
lastValueString: unicodestring
lastValue, lowerWord: integer
binaryOID, oidPrefix: sequence of BYTE
attr: ATTRTYP
pos: integer
/* get the last value in the original OID: the value
 * after the last '.'*/
lastValueString := SubString(o,
                             FindCharRev(o, o.length,'.'),
                             o.length)
lastValue := ToInteger(lastValueString)
/* convert the dotted form of OID into a BER encoded binary
 * format. The BER encoding of OID is described in section
 * 8.19 of [ITUX690]*/
binaryOID := ToBinaryOid(o)
/* get the prefix of the OID*/
if lastValue < 128 then
  oidPrefix := SubBinary(binaryOID, 0, binaryOID.length - 1)
else
  oidPrefix := SubBinary(binaryOID, 0, binaryOID.length - 2)
endif
/* search the prefix in the prefix table, if none found, add
 * one entry for the new prefix.*/
fToAdd := true
for i := 0 to t.length
  if ToBinary(t[i].prefixString) = oidPrefix then
    fToAdd := false
    pos := i
  endif
endfor
if fToAdd then
  pos := t.length
  AddPrefixTableEntry(t, oidPrefix)
endif
/*compose the attid*/
lowerWord := lastValue mod 16384
if lastValue ≥ 16384 then
```

```
      /*mark it so we know it is not the whole lastValue*/
      lowerWord := lowerWord + 32768
    endif
    upperWord := t[pos].prefixIndex
    attr := upperWord * 65536 + lowerWord
    return attr
  procedure OidFromAttid(t: PrefixTable, attr: ATTRTYP): OID
```

Informative summary of behavior: This procedure converts an **ATTRTYP** representation to a corresponding OID.

```
    i, upperWord, lowerWord: integer
    binaryOID: sequence of BYTE
    binaryOID = null
    /* separate the ATTRTYP into two parts*/
    upperWord := attr / 65536
    lowerWord := attr mod 65536
    /* search in the prefix table to find the upperWord, if found,
     * construct the binary OID by appending lowerWord to the end of
     * found prefix.*/
    for i := 0 to t.length
      if t[i].prefixIndex = upperWord then
        if lowerWord < 128 then
          binaryOID := CatBinary(ToBinary(t[i].prefixString),
              ToByte(lowerWord))
        else
          if lowerWord ≥ 32768 then
            lowerWord := lowerWord - 32768
          endif
          binaryOID := CatBinary(ToBinary(t[i].prefixString),
              ToByte(((lowerWord / 128) mod 128) + 128))
          binaryOID := CatBinary(binaryOID, ToByte(lowerWord mod 128))
        endif
      endif
    endfor
    if binaryOID = null then
      return null
    else
      return ToStringOID(binaryOID)
    endif
  procedure NewPrefixTable( ): PrefixTable
```

This procedure creates a new **PrefixTable**, inserts the following tuples into the table, and returns the table as the result.

prefixString	Length of prefixString	prefixIndex
"\x55\x4"	2	0
"\x55\x6"	2	1
"\x2A\x86\x48\x86\xF7\x14\x01\x02"	8	2
"\x2A\x86\x48\x86\xF7\x14\x01\x03"	8	3

prefixString	Length of prefixString	prefixIndex
"\x60\x86\x48\x01\x65\x02\x02\x01"	8	4
"\x60\x86\x48\x01\x65\x02\x02\x03"	8	5
"\x60\x86\x48\x01\x65\x02\x01\x05"	8	6
"\x60\x86\x48\x01\x65\x02\x01\x04"	8	7
"\x55\x5"	2	8
"\x2A\x86\x48\x86\xF7\x14\x01\x04"	8	9
"\x2A\x86\x48\x86\xF7\x14\x01\x05"	8	10
"\x09\x92\x26\x89\x93\xF2\x2C\x64"	8	19
"\x60\x86\x48\x01\x86\xF8\x42\x03"	8	20
"\x09\x92\x26\x89\x93\xF2\x2C\x64\x01"	9	21
"\x60\x86\x48\x01\x86\xF8\x42\x03\x01"	9	22
"\x2A\x86\x48\x86\xF7\x14\x01\x05\xB6\x58"	10	23
"\x55\x15"	2	24
"\x55\x12"	2	25
"\x55\x14"	2	26

The following examples show the correspondence between OID and **ATTRTYP** by using the **PrefixTable** returned by the procedure NewPrefixTable.

```
OID: 2.5.4.6 (countryName attribute)
Binary: \x55\x04\x06
Prefix string: "\x55\x04"
Prefix index: 0
ATTRTYP: 0x00000006
OID: 2.5.6.2 (country class)
Binary: \x55\x06\x02
Prefix string: "\x55\x06"
Prefix index: 1
ATTRTYP: 0x00010002
OID: 1.2.840.113556.1.2.1 (instanceType attribute)
Binary: \x2A\x86\x48\x86\xF7\x14\x01\x02\x01
Prefix string: "\x2A\x86\x48\x86\xF7\x14\x01\x02"
Prefix index: 2
ATTRTYP: 0x00020001
OID: 1.2.840.113556.1.3.23 (container class)
Binary: \x2A\x86\x48\x86\xF7\x14\x01\x03\x17
Prefix string: "\x2A\x86\x48\x86\xF7\x14\x01\x03"
Prefix index: 3
ATTRTYP: 0x00030017
OID: 2.5.5.1 (attribute syntax: distinguished name)
Binary: \x55\x5\x1
Prefix string: "\x55\x5"
```

```
Prefix index: 8
ATTRTYP: 0x00080001
OID: 1.2.840.113556.1.4.1 (RDN attribute)
Binary: \x2A\x86\x48\x86\xF7\x14\x01\x04\x01
Prefix string: "\x2A\x86\x48\x86\xF7\x14\x01\x04"
Prefix index: 9
ATTRTYP: 0x00090001
OID: 1.2.840.113556.1.5.1 (securityObject class)
Binary: \x2A\x86\x48\x86\xF7\x14\x01\x05\x01
Prefix string: "\x2A\x86\x48\x86\xF7\x14\x01\x05"
Prefix index: 10
ATTRTYP: 0x000a0001
OID: 0.9.2342.19200300.100.1.1 (uid attribute)
Binary: \x09\x92\x26\x89\x93\xF2\x2C\x64\x01\x01
Prefix string: "\x09\x92\x26\x89\x93\xF2\x2C\x64\x01"
Prefix index: 21
ATTRTYP: 0x00150001
OID: 2.16.840.1.113730.3.1.1 (carLicense attribute)
Binary: \x60\x86\x48\x01\x86\xF8\x42\x03\x01\x01
Prefix string: "\x60\x86\x48\x01\x86\xF8\x42\x03\x01"
Prefix index: 22
ATTRTYP: 0x00160001
OID: 1.2.840.113556.1.5.7000.53 (crossRefContainer class)
Binary: \x2A\x86\x48\x86\xF7\x14\x01\x05\xB6\x58\x35
Prefix string: "\x2A\x86\x48\x86\xF7\x14\x01\x05\xB6\x58"
Prefix index: 23
ATTRTYP: 0x00170035
OID: 2.5.21.2 (ditContentRules attribute)
Binary: \x55\x15\x02
Prefix string: "\x55\x15"
Prefix index: 24
ATTRTYP: 0x00180002
OID: 2.5.18.1 (createTimeStamp attribute)
Binary: \x55\x12\x01
Prefix string: "\x55\x12"
Prefix index: 25
ATTRTYP: 0x00190001
OID: 2.5.20.1 (subSchema class)
Binary: \x55\x14\x01
Prefix string: "\x55\x14"
Prefix index: 26
ATTRTYP: 0x001a0001
```

5.14 BOOL

A concrete type for a boolean value, as specified in [MS-DTYP] section **2.2**.

5.15 BYTE

A concrete type for a single byte, as specified in [MS-DTYP] section **2.2**.

5.16 ClientExtensions

```
procedure ClientExtensions(hDrs: DRS_HANDLE): DRS_EXTENSIONS_INT
```

This server method gets the client extensions presented in the IDL_DRSBind call that created hDrs. Any fields not specified by the client in the pextClient parameter to IDL_DRSBind (such that pextClient^.cb is less than the offset of the end of the field of DRS_EXTENSIONS_INT) are set to 0.

5.17 ConcretePTFromAbstractPT

```
procedure ConcretePTFromAbstractPT(
  prefixTable: PrefixTable): SCHEMA_PREFIX_TABLE
```

Informative summary of behavior: Translates abstract PrefixTable to a **SCHEMA_PREFIX_TABLE** structure.

```
prefixCount: ULONG
concretePrefixTable: SCHEMA_PREFIX_TABLE
schemaSignature: sequence of BYTE
prefixCount := prefixTable.length
concretePrefixTable.PrefixCount := prefixCount
for i := 0 to (prefixTable.length - 1)
  concretePrefixTable.pPrefixTableEntry[i].prefix :=
    prefixTable[i].prefixString
  concretePrefixTable.pPrefixTableEntry[i].ndx :=
    prefixTable[i].prefixIndex
endfor
return concretePrefixTable
```

5.18 ConfigNC

```
procedure ConfigNC(): DSName
```

This procedure returns the dsname of dc.configNC.

5.19 dc, DC

A global variable that represents the state of a domain controller (DC) as defined in [MS-ADTS] section 3.1.1.1.9, and the type of that variable. That definition is repeated here for convenience:

type **DC** = [

serverGuid: GUID,

usn: 64-bit integer,

prefixTable: PrefixTable,

defaultNC: full domain NC replica,

configNC: config NC replica,

schemaNC: schema NC replica,

partialDomainNCs: set of partial domain NC replica,

appNCs: set of application NC replica,

ldapConnections: LDAPConnections,

replicationQueue: ReplicationQueue,

kccFailedConnections: KCCFailedConnections,

kccFailedLinks: KCCFailedLinks,

rpcClientContexts: RPCClientContexts,

rpcOutgoingContexts: RPCOutgoingContexts

]

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

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

```
dc: DC
```

5.20 DefaultNC

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

This procedure returns the dsname of the dc.defaultNC.

5.21 DefaultNCOfDC

```
procedure DefaultNCOfDC(dcName: unicodestring): DSName
```

Returns the dsname of the default NC of the DC named by dcName, where dcName is the DNS name or the NetBIOS name of the DC.

5.22 DescendantObject

```
procedure DescendantObject(
    ancestor: DSName, rdns: unicodestring): DSName
```

This procedure constructs a DN string by concatenating rdns and ancestor.dn, and then verifies the existence of the descendant object. It returns the DSName if the descendant exists, and null otherwise.

5.23 DN

An abstract type that is a *unicodestring* (section 3.4.3) containing a distinguished name (DN) of the form specified in [RFC2253].

5.24 DNBinary

An abstract type representing the concrete type SYNTAX_DISTNAME_BINARY. It consists of the following tuple:

type DNBinary = [dn: DSName, binary: sequence of BYTE]

5.25 DomainNameFromNT4AccountName

```
procedure DomainNameFromNT4AccountName(
  nt4AccountName: unicodestring): unicodestring
```

If nt4AccountName is a name in Windows NT 4.0 account name format, that is, two components separated by a backslash (for example, "DOMAIN\username"), this procedure returns the first component (the domain name, or "DOMAIN" in this example). If the nt4AccountName is not in this format, null is returned.

5.26 DRS_EXTENSIONS

The **DRS_EXTENSIONS** structure defines a concrete type for capabilities information used in version negotiation.

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

cb: Size in bytes of the **rgb** array.

rgb: To RPC this field is a string of **cb** bytes. It is interpreted by client and server as the first **cb** bytes of a DRS_EXTENSIONS_INT structure that follow the **cb** field of that structure. The fields of the DRS_EXTENSIONS_INT are in little-endian byte order. Since both **DRS_EXTENSIONS** and DRS_EXTENSIONS_INT begin with a **DWORD cb**, a field in DRS_EXTENSIONS_INT is at the same offset in **DRS_EXTENSIONS** as it is in DRS_EXTENSIONS_INT.

5.27 DRS_EXTENSIONS_INT

The DRS_EXTENSIONS_INT structure is a concrete type for structured capabilities information used in version negotiation.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
cb																															
dwFlags																															
SiteObjGuid																															
...																															
...																															
...																															
Pid																															
dwReplEpoch																															
dwFlagsExt																															

cb (4 bytes): The count of bytes in the fields **dwFlags** through **dwReplEpoch** inclusive. Allows extended versions of this structure to carry more information in future product versions.

dwFlags (4 bytes): The dwFlags field contains individual bit flags that describe the capabilities of the DC that produced the DRS_EXTENSIONS_INT structure.<10>

The following table lists the bit flags.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
BAS	AS	RM	MV	DF	DC	UO	AE	KE	AE2	LVR	DC2	INR	CB	GRI	SE	DCF	TM	SH	PB3	GC5	GM2	GC6	ANC	GC8	GR5	GR6	WB3	DF2	R1	R2	R3

BAS (DRS_EXT_BASE): Unused, MUST be 1 and ignored.

AS (DRS_EXT_ASYNCREPL): If present, signifies that the DC supports DRS_MSG_REPADD_V2 and a version for the server-to-server replication implementation.

RM (DRS_EXT_REMOVEAPI): If present, signifies that the DC supports IDL_DRSRemoveDsServer and IDL_DRSRemoveDsDomain.

MV (DRS_EXT_MOVEREQ_V2): If present, signifies that the DC supports a version for the server-to-server replication implementation.

DF (DRS_EXT_GETCHG_DEFLATE): If present, signifies that the DC supports a version for the server-to-server replication implementation.

DC (DRS_EXT_DCINFO_V1): If present, signifies that the DC supports IDL_DRSDomainControllerInfo.

UO (DRS_EXT_RESTORE_USN_OPTIMIZATION): Unused, MUST be 1 and ignored.

AE (DRS_EXT_ADDENTRY): If present, signifies that the DC supports a version for the server-to-server implementation of creating objects remotely.

KE (DRS_EXT_KCC_EXECUTE): If present, signifies that the DC supports IDL_DRSExecuteKCC.

AE2 (DRS_EXT_ADDENTRY_V2): If present, signifies that the DC supports a version for the server-to-server implementation of creating objects remotely.

LVR (DRS_EXT_LINKED_VALUE_REPLICATION): If present, signifies that the DC supports link value replication, and this support is enabled.

DC2 (DRS_EXT_DCINFO_V2): If present, signifies that the DC supports DRS_MSG_DCINFOREPLY_V2.

INR (DRS_EXT_INSTANCE_TYPE_NOT_REQ_ON_MOD): Unused, MUST be 1 and ignored.

CB (DRS_EXT_CRYPTO_BIND): A server-to-server implementation-specific flag. If present, it indicates that the security provider used for the connection supports encryption through RPC.

GRI (DRS_EXT_GET_REPL_INFO): If present, signifies that the DC supports IDL_DRSGetReplInfo.

SE (DRS_EXT_STRONG_ENCRYPTION): If present, signifies that the DC supports a version for the server-to-server replication implementation that uses 128-bit encryption.

DCF (DRS_EXT_DCINFO_VFFFFFFFF): If present, signifies that the DC supports DRS_MSG_DCINFOREPLY_VFFFFFFFF.

TM (DRS_EXT_TRANSITIVE_MEMBERSHIP): If present, signifies that the DC supports a version for the server-to-server implementation of group expansion.

SH (DRS_EXT_ADD_SID_HISTORY): If present, signifies that the DC supports IDL_DRSAddSidHistory.

PB3 (DRS_EXT_POST_BETA3): Unused, MUST be 1 and ignored.

GC5 (DRS_EXT_GETCHGREQ_V5): If present, signifies that the DC supports a version for the server-to-server replication implementation.

GM2(DRS_EXT_GETMEMBERSHIPS2): If present, signifies that the DC supports a version for the server-to-server implementation of group expansion.

GC6 (DRS_EXT_GETCHGREQ_V6): If present, signifies that the DC supports a version for the server-to-server replication implementation.

ANC (DRS_EXT_NONDOMAIN_NCS): If present, signifies that the DC supports application NCs.

GC8 (DRS_EXT_GETCHGREQ_V8): If present, signifies that the DC supports a version for the server-to-server replication implementation.

GR5 (DRS_EXT_GETCHGREPLY_V5): If present, signifies that the DC supports a version for the server-to-server replication implementation.

GR6 (DRS_EXT_GETCHGREPLY_V6): If present, signifies that the DC supports a version for the server-to-server replication implementation.

WB3 (DRS_EXT_WHISTLER_BETA3): If present, signifies that the DC supports DRS_MSG_REPVERIFYOBJ, the server-to-server implementation of creating objects remotely, and a version of the server-to-server replication implementation method.

DF2 (DRS_EXT_W2K3_DEFLATE): If present, signifies that the DC supports a version for the server-to-server replication implementation.

R1 (DRS_EXT_RESERVED_FOR_WIN2K_OR_DOTNET_PART1): Unused, MUST be 0 and ignored.

R2 (DRS_EXT_RESERVED_FOR_WIN2K_OR_DOTNET_PART2): Unused, MUST be 0 and ignored.

R3 (DRS_EXT_RESERVED_FOR_WIN2K_OR_DOTNET_PART3): Unused, MUST be 0 and ignored.

SiteObjGuid (16 bytes): A GUID. The objectGUID of the site object of which the DC's DSA object is a descendant. For non-DC client callers, this field SHOULD be set to zero.

Pid (4 bytes): 32-bit signed integer value specifying the process identifier of the client. This is for informational and debugging purposes only. The assignment of this field is implementation-specific.<11>

dwReplEpoch (4 bytes): 32-bit unsigned integer value. This value is set to zero by all client callers. The server sets this value by assigning the value of msDS-ReplicationEpoch from its nTDSDSAobject.

dwFlagsExt (4 bytes): Extension of the dwFlags field that contains individual bit flags that describe the capabilities of the DC that produced the DRS_EXTENSIONS_INT structure. For non-DC client callers, no bits SHOULD be set. The following table lists the bit flags.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
DA	LH	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

DA (DRS_EXT_ADAM): If present, signifies that the DC supports DRS_MSG_REPSYNC_V2, DRS_MSG_UPDREFS_V2, DRS_MSG_INIT_DEMOTIONREQ_V1, DRS_MSG_REPLICA_DEMOTIONREQ_V1, and DRS_MSG_FINISH_DEMOTIONREQ_V1.

LH (DRS_EXT_LH_BETA2): If present, signifies that the DC supports DRS_FULL_REP and DRS_SELECT_SECRET_REP flags and supports infolevel 3 in DRS_MSG_DCINFOREQ_V1.

5.28 DRS_HANDLE

A concrete type for an RPC context handle (as specified in [C706]) for use in calls to methods in the drsuapi RPC interface.

This type is declared as follows:

```
typedef [context_handle] void* DRS_HANDLE;
```

For the specification of IDL_DRSBind, see section 4.1.2.

5.29 DRS_OPTIONS

A concrete type for a set of options sent to and received from various drsuapi methods.

This type is declared as follows:

```
typedef unsigned long DRS_OPTIONS;
```

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

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
AS	X	AR	ALL/DR	WR	IS	PS	MR	ASR/IE	TS	CO	GA	GS/LO	FR	SN/RF	FS/NS	X	FSP	X	X	X	X	X	SS	X	SF	DAS	DPS	UC	NN	SP	X

X: Unused. MUST be zero and ignored.

AS (DRS_ASYNC_OP): Perform the operation asynchronously.

AR (DRS_ADD_REF): Register a client DC for notifications of updates to the NC replica.

ALL (DRS_SYNC_ALL): Replicate from all server DCs.

DR (DRS_DEL_REF): Deregister a client DC from notifications of updates to the NC replica.

WR (DRS_WRIT_REP): Replicate a writable replica, not a read-only partial replica or read-only full replica.

IS (DRS_INIT_SYNC): Perform replication at startup.

PS (DRS_PER_SYNC): Perform replication periodically.

MR (DRS_MAIL_REP): Perform replication using SMTP as a transport.

ASR (DRS_ASYNC_REP): Populate the NC replica asynchronously.

IE (DRS_IGNORE_ERROR): Ignore errors.

TS (DRS_TWOWAY_SYNC): Inform server DC to replicate from the client DC.

CO (DRS_CRITICAL_ONLY): Replicate only system-critical objects.

GA (DRS_GET_ANC): Include updates to ancestor objects before updates to their descendants.

GS (DRS_GET_NC_SIZE): Get the approximate size of the server NC replica.

LO (DRS_LOCAL_ONLY): Perform the operation locally, without contacting any other DC.

FR (DRS_FULL_REP): Replicate a read-only full replica. Not a writable or partial replica.

SN (DRS_SYNC_BYNAME): Choose the source server by network name.

RF (DRS_REF_OK): Allow NC replica to be removed even if other DCs use this DC as a replication server DC.

FS (DRS_FULL_SYNC_NOW): Replicate all updates now, even those that would normally be filtered.

NS (DRS_NO_SOURCE): NC replica has no server DCs.

FSP (DRS_FULL_SYNC_PACKET): Replicate all updates in the replication request, even those that would normally be filtered.

SS (DRS_SELECT_SECRET_REP): Do not replicate attribute values of attributes containing **secret data**.

SF (DRS_SYNC_FORCED): Force replication, even if the replication system is otherwise disabled.

DAS (DRS_DISABLE_AUTO_SYNC): Disable replication induced by update notifications.

DPS (DRS_DISABLE_PERIODIC_SYNC): Disable periodic replication.

UC (DRS_USE_COMPRESSION): Compress response messages.

NN (DRS_NEVER_NOTIFY): Do not send update notifications.

SP (DRS_SYNC_PAS): Expand the partial attribute set of the partial replica.

5.30 DRS_SPN_CLASS

A *unicodestring* constant (section 3.4.3) that is used as the service class in the SPN for a DC. It has the value "E3514235-4B06-11D1-AB04-00C04FC2DCD2". This SPN is used for mutual authentication in the server-to-server replication implementation.

5.31 DS_REPL_OP_TYPE

A concrete type for the replication operation type.

```
typedef  enum
{
  DS_REPL_OP_TYPE_SYNC = 0x00000000,
  DS_REPL_OP_TYPE_ADD = 0x00000001,
  DS_REPL_OP_TYPE_DELETE = 0x00000002,
  DS_REPL_OP_TYPE_MODIFY = 0x00000003,
  DS_REPL_OP_TYPE_UPDATE_REFS = 0x00000004
} DS_REPL_OP_TYPE;
```

DS_REPL_OP_TYPE_SYNC: Sync NC replica from server DC.

DS_REPL_OP_TYPE_ADD: Add NC replica server DC.

DS_REPL_OP_TYPE_DELETE: Remove NC replica server DC.

DS_REPL_OP_TYPE_MODIFY: Modify NC replica server DC.

DS_REPL_OP_TYPE_UPDATE_REFS: Update NC replica client DC.

5.32 DSAObj

```
procedure DSAObj(): DSName
```

Returns the dsname of the DC's nTDSDSA object.

```
return select one o from children ConfigNC()
  where o!objectGUID = dc.serverGUID
```

5.33 DSName

An abstract type for representing a dsname. It corresponds to the concrete representation **DSNAME**. It consists of a tuple which identifies an object in the directory. This tuple is discussed in [MS-ADTS] section 3.1.1.1.5. For this document, the fields of the tuple are defined as follows:

```
type DSName =  [dn: NameString , guid: GUID, sid: Sid]
```

The *dn* field corresponds to the NameString field of the **DSNAME**, and contains the distinguished name (DN) of the object. The *guid* field corresponds to the GUID field of the **DSNAME** and contains the value of the object's objectGUID attribute. The *sid* field corresponds to the Sid field of the **DSNAME**. If the object possesses an objectSid attribute, it contains the value of the object's objectSid attribute. If the object does not possess an objectSid attribute, the field is null.

5.34 DSNAME

A concrete type for representing a DSName, identifying a directory object using the values of one or more of its LDAP attributes: objectGUID, objectSid, or distinguishedName.

```
typedef struct {
  unsigned long structLen;
  unsigned long SidLen;
  GUID Guid;
  NT4SID Sid;
  unsigned long NameLen;
  [size_is(NameLen + 1)] WCHAR StringName[];
} DSNAME;
```

structLen: Length, in bytes, of the entire data structure.

SidLen: Number of bytes in the Sid field used to represent the object's objectSid attribute value. Zero indicates that the DSNAME does not identify the objectSid value of the directory object.

Guid: The value of the object's objectGUID attribute specified as a **GUID** structure, which is defined in [MS-DTYP] section **2.3.2**. Zero values for all fields in the **GUID** structure indicates that the **DSNAME** does not identify the objectGUID value of the directory object.

Sid: The value of the object's objectSid attribute, its security identifier (see [MS-SECO] section 2.1.2), specified as a **SID** structure, which is defined in [MS-DTYP] section **2.4.2**. The size of this field is exactly 28 bytes, regardless of the value of **SidLen**, which specifies how many bytes in this field are used. Note that this is smaller than the theoretical size limit of a **SID**, which is 68 bytes. While Windows publishes a general SID format, Windows never uses that format in its full generality. 28 bytes is sufficient for a Windows SID.

NameLen: Number of characters in the **StringName** field, not including the null terminator, used to represent the object's distinguishedName attribute value. Zero indicates that the DSNAME does not identify the distinguishedName value of the directory object.

StringName: Null-terminated Unicode value of the object's distinguishedName attribute, as specified in [MS-ADTS] section 3.1.1.1.4. This field always contains at least one character: the null terminator. Each Unicode value is encoded as 2 bytes. The byte ordering is little-endian.

The following table shows an alternative representation of this structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
structLen																															
SidLen																															
Guid.Data1																															
Guid.Data2																Guid.Data3															
Guid.Data4...																															
...Guid.Data4																															
Sid...																															
...Sid...																															
...Sid...																															
...Sid...																															
...Sid...																															
...Sid...																															
...Sid																															
NameLen																															
StringName (Variable Length) ...																															

Note All fields have little-endian byte ordering.

5.34.1 DSNAME Equality

When comparing **DSNAME** elements for equality, an implementation must be aware that multiple attributes may be specified. **DSNAME** values x and y are equal if and only if one of the following conditions holds:

- x.Guid is not zeros and y.Guid is not zeros and x.Guid = y.Guid
- All of the following are true:

- x.Guid is zeros or y.Guid is zeros
- x.StringLen ≠ 0
- The number of RDNs in x is the same as that of y
- For each RDN x_i in x and RDN y_i in y (see [RFC2253]):
  - AttributeType of x_i = AttributeType of y_i
  - AttributeValue of x_i = AttributeValue of y_i, without regard to case differences, Hiragana and Katakana character differences, and non-spacing characters

- All of the following are true:
  - x.Guid is zeros
  - y.Guid is zeros
  - x.StringLen = 0
  - y.StringLen = 0
  - x.SidLen ≠ 0
  - x.SidLen = y.SidLen
  - x.Sid and y.Sid contain identical values in the first x.SidLen array items

5.35 DSTIME

A concrete type for time expressed as the number of seconds since January 1, 1601, 12:00:00am.

This type is declared as follows:

```
typedef LONGLONG DSTIME;
```

The following table shows an alternative representation of this structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
cSeconds...																															
...cSeconds																															

Note Byte ordering is little-endian.

5.36 DWORD

A concrete type for a 32-bit unsigned integer, as specified in [MS-DTYP] section **2.2.7**.

5.37 Expunge

```
procedure Expunge(obj: DSName)
```

Physically removes an object whose DSName is obj from the directory, without enforcing referential integrity constraints. The object is immediately removed without undergoing conversion to a tombstone.

5.38 FILETIME

A concrete type for a time, as specified in [MS-DTYP] section **2.3.1**.

5.39 FindCharRev

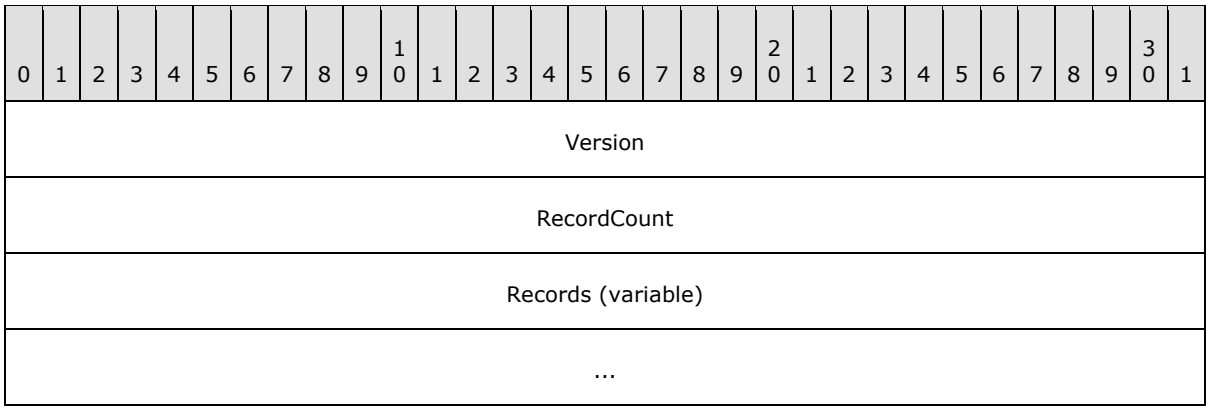
```
procedure FindCharRev(
    s: unicodestring,
    start: integer,
    c: UCHAR): integer
```

Informative summary of behavior: Returns the zero-based index of the last occurrence of c in the portion of s between start and the end of s.

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

5.40 FOREST_TRUST_INFORMATION

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



Version (4 bytes): Version of the data structure. The only supported version of the data structure is 1.

RecordCount (4 bytes): Number of records present in the data structure.

Records (variable): Variable-length records each containing specific type of data about the forest trust relationship.

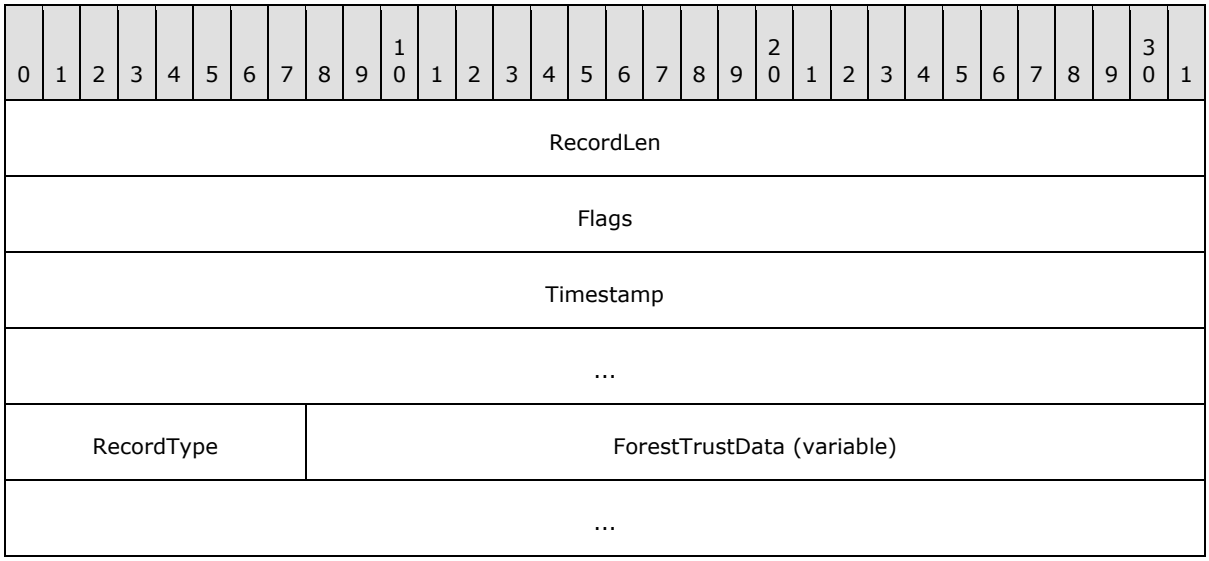
Note Records are not necessarily aligned to 32-bit boundaries. Each record starts at the next byte after the previous record ends.

Each record is represented as described in section 5.40.1.

Note All fields have little-endian byte ordering.

5.40.1 Record

Each Record is represented in the following manner:



RecordLen (4 bytes): Length, in bytes, of the entire record.

Flags (4 bytes): Individual bit flags that control how the forest trust information in this record can be used.

If RecordType = 0 or 1, the **Flags** field can have one or more of the following bits:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TDN	TDA	TDC	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X: Unused. Must be zero and ignored.

TDN (LSA_TLN_DISABLED_NEW): Entry is not yet enabled.

TDA (LSA_TLN_DISABLED_ADMIN): Entry is disabled by administrator.

TDC (LSA_TLN_DISABLED_CONFLICT): Entry is disabled due to conflict with another trusted domain.

If RecordType = 2, the **Flags** field can have one or more of the following bits:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SDA	SDC	NDA	NDC	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X: Unused. MUST be zero and ignored.

SDA (LSA_SID_DISABLED_ADMIN): Entry is disabled for SID based matches by administrator.

SDC (LSA_SID_DISABLED_CONFLICT): Entry is disabled due to SID conflict with another trusted domain.

NDA (LSA_NB_DISABLED_ADMIN): Entry is disabled for NetBIOS name-based matches by administrator.

NDC (LSA_NB_DISABLED_CONFLICT): Entry is disabled due to NetBIOS domain name conflict with another trusted domain.

For RecordType = 2, NETBIOS_DISABLED_MASK is defined as a mask on the lower 4 bits of the Flags field.

For all record types, LSA_FTRECORD_DISABLED_REASONS is defined as a mask on the lower 16 bits of the Flags field. Unused bits covered by the mask are reserved for future use.

Timestamp (8 bytes): 64-bit timestamp value indicating when this entry was created.

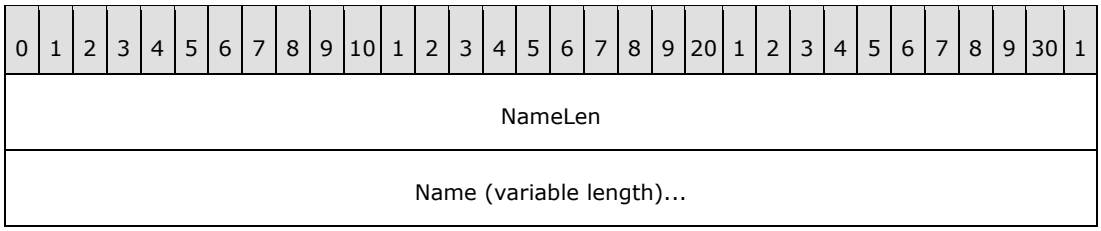
RecordType (1 byte): 8-bit value specifying the type of record contained in this specific entry. The allowed values are specified in section 5.41.

ForestTrustData (variable): Variable length type-specific record, depending on the RecordType value, containing specific type of data about the forest trust relationship.

IMPORTANT NOTE: The type-specific ForestTrustData record is not necessarily aligned to a 32-bit boundary. Each record starts at the byte following the RecordType field.

There are three different type-specific records. Depending on the value of the RecordType field, the structure of the type-specific record differs as described below.

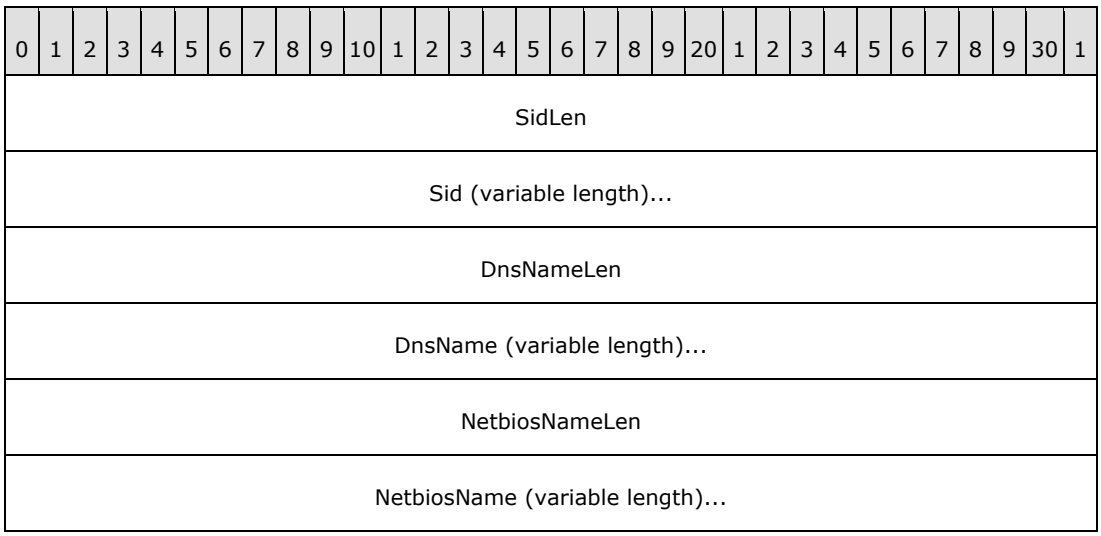
- If RecordType = 0 or RecordType = 1, then the type-specific record is represented in the following manner:



NameLen: Length, in bytes, of the Name field below.

Name: Top-level name of the trusted forest in UTF-8 format.

- If RecordType = 2, then the type-specific record is represented in the following manner. Note that the record contains the following structures one after another. It is important to note here that none of the data shown below is necessarily aligned to 32-bit boundaries.



SidLen: Length, in bytes, of the Sid field below.

Sid: The SID of a domain in the trusted forest, specified as a SID structure, which is defined in [MS-DTYP] section **2.4.2**.

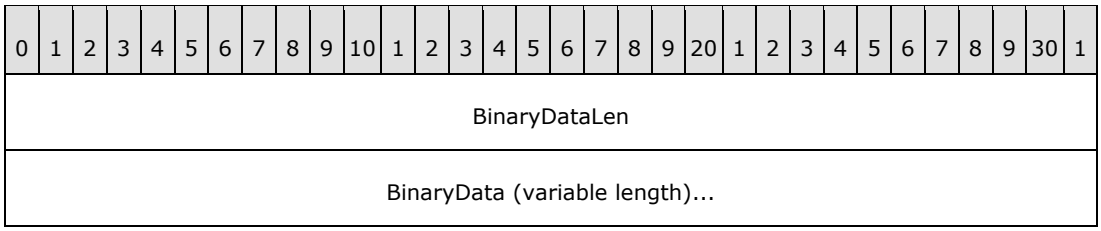
DnsNameLen: Length, in bytes, of the DnsName field below.

DnsName: The DNS name of a domain in the trusted forest in UTF-8 format.

NetbiosNameLen: Length, in bytes, of the NetbiosName field below.

NetbiosName: The NetBIOS name of a domain in the trusted forest in UTF-8 format.

- If RecordType is not one of the values above, then the type-specific record is represented in the following manner:



BinaryDataLen: Length, in bytes, of the BinaryData field below.

BinaryData: Trusted forest data.

5.40.2 Determining If a Name Is In a Trusted Forest

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

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

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

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

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

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

The X_FOREST_TRUST_INFORMATION structure defined above is used by the procedure to determine if a given domain is in a trusted forest. To unmarshal the content of the msDS-TrustForestTrustInfo attribute into this structure, the procedure UnmarsallForestTrustInfo described below can be used.

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

The sequence [index .. index + size] of bytes in buffer is interpreted as a UTF8 string, and a corresponding *unicodestring* (section 3.4.3) is returned.

```
procedure ExtractSid(
    buffer: sequence of BYTE,
    index: DWORD, size: DWORD): SID;
```

The sequence [index .. index + size] of bytes in buffer is converted into a SID structure and returned.

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

The sequence [index .. index + size] of bytes in buffer is returned.

```
procedure UnmarshalForestTrustInfo
    (inputBuffer: sequence of BYTE,
     var forestTrustInfo: X_FOREST_TRUST_INFORMATION): boolean
```

Informative summary of behavior: This procedure unmarshals byte stream inputBuffer, which holds the content of a msDS-TrustForestTrustInfo attribute containing forest trust information as described in FOREST_TRUST_INFORMATION, into the forestTrustInfo structure.

```
  index: DWORD
  pdwVersion: ADDRESS OF DWORD
  pdwRecordCount: ADDRESS OF DWORD
  i: DWORD
  pwdRecordLength: ADDRESS OF DWORD
  pTrustRecord: ADDRESS OF X_FOREST_TRUST_RECORD
  pulTime: ADDRESS OF ULONGLONG
  pType: ADDRESS OF BYTE
  pSid: ADDRESS OF SID
  pString: ADDRESS OF unicodestring
  pdwSize: ADDRESS OF DWORD

  index := 0

  pdwVersion := ADR(inputBuffer[index])
  if pdwVersion^ ≠ 1 then
    return false
  endif

  index := index + 4

  pdwRecordCount := ADR(inputBuffer[index])
  forestTrustInfo.RecordCount := pdwRecordCount^
  index := index + 4

  /* Extract each record */
  for i:= 0 to pdwRecordCount^

    /* First 4 bytes of the record is the length */
```

```
pdwRecordLength := ADR(inputBuffer[index])
index := index + 4

pTrustRecord := forestTrustInfo.Entries[i]

/* Next 4 bytes of the record are the flags */
pdwFlags := ADR(inputBuffer[index])
pTrustRecord^.Flags := pdwFlags^
index := index + 4

/* Next 8 bytes of the record represent the Time field */
pulTime := ADR(inputBuffer[index])
pTrustRecord^.Time := pulTime^
index := index + 8

/* Next byte represents trust type */
pType := ADR(inputBuffer[index])
pTrustRecord^.ForestTrustType := pType^
index := index + 1

if (pTrustRecord^.ForestTrustType = ForestTrustTopLevelName or
    pTrustRecord^.ForestTrustType = ForestTrustTopLevelNameEx)
        then

  /* Next 4 bytes represent the size of the top level name */
  pdwSize := ADR(inputBuffer[index])
  index := index + 4

  /* Extract the top level name; index is at the start of name */
  pTrustRecord^.TopLevelName :=
      ExtractString(inputBuffer, index, pdwSize^)
  index := index + pdwSize^
else
   if (pTrustRecord^.ForestTrustType = ForestTrustDomainInfo)
          then
     /* Next 4 bytes represent the size of the sid */
     pdwSize := ADR(inputBuffer[index])
     index := index + 4

     /* Extract the sid; index is at the start of sid */
     pTrustRecord^.DomainInfo.Sid :=
         ExtractSid(inputBuffer, index, pdwSize^)
     index := index + pdwSize^

     /* Next 4 bytes represent the size of the dns domain name */
     pdwSize := ADR(inputBuffer[index])
     index := index + 4

     /* Extract the dns domain name; index is at start of name */
     pTrustRecord^.DomainInfo.DnsName :=
         ExtractString(inputBuffer, index, pdwSize^)
     index := index + pdwSize^

     /* Next 4 bytes represent the size of the netbios
      * domain name */
     pdwSize := ADR(inputBuffer[index])
     index := index + 4
```

```
        /* Extract the netbios domain name; index is at the start
         * of name */
        pTrustRecord^.DomainInfo.NetbiosName :=
            ExtractString(inputBuffer, index, pdwSize^)
        index := index + pdwSize^
      else
        /* Next 4 bytes represent the size of the binary data */
        pdwSize := ADR(inputBuffer[index])
        pTrustRecord^.Data.Length := pdwSize^
        index := index + 4

        /* Extract the binary data; index is at the start of data */
        pTrustRecord^.Data.Buffer :=
            ExtractBinaryData(inputbuffer, index, pdwSize^)
        index := index + pdwSize^
      endif

    endif

    /* index is now at the beginning of the next record */
  endfor

  return true
```

The following procedure is used to determine if a given domain is in a trusted forest. Since it makes use of forest trust information data stored in objects in the NC replica of the forest root domain (see FOREST_TRUST_INFORMATION), this function only works on GC servers or DCs in the forest root domain.

```
procedure IsDomainNameInTrustedForest(name: unicodestring): boolean
```

Informative summary of behavior: This procedure determines if the domain with the name given by name is in a trusted forest. The input name may be a DNS or a NetBIOS name.

```
  if IsDomainDnsNameInTrustedForest(name) then
    return true
  endIf

  if IsDomainNetbiosNameInTrustedForest(name) then
    return true
  endIf

  return false
```

The IsDomainNameInTrustedForest procedure uses the following helper procedures to determine if a domain is in a trusted forest.

```
procedure ForestTrustOwnsName(o: DSName, name: unicodestring): boolean

  f: X_FOREST_TRUST_INFORMATION

  /* o is the DSName identifying the object containing the forest
   * trust information being evaluated to determine if the input
```

```
     * name is owned by the trusted forest represented by o. */

    if not UnmarshalForestTrustInfo(o!msDS-TrustForestTrustInfo, f)
      /* unable to extract forest trust information from binary form */
      return false
    endif

    /* if a suffix of the name is in the exclusion list, the
     * forest does not own this name */
    foreach e in f.Entries
      if (e.ForestTrustType = ForestTrustTopLevelNameEx and
            e.TopLevelName is a suffix for name) then
        return false
      endif
    endfor

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

    return false

  procedure IsDomainDnsNameInTrustedForest(name: unicodestring)
      : boolean

    tdos: set of DSName
    f: X_FOREST_TRUST_INFORMATION

    /* Get all the objects that represent trusted domains */
    tdos := select all o in Children ForestRootDomainNC() where
                    trustedDomain in o!objectClass and
                    o!trustAttributes & 0x00000008 ≠ 0 and
                    o!msDS-TrustForestTrustInfo ≠ null

    foreach o in tdos
      if not UnmarshalForestTrustInfo(o!msDS-TrustForestTrustInfo, f)
          then
        return false
      else
        foreach e in f.Entries
          if (e.ForestTrustType = ForestTrustDomainInfo and
                e.DomainInfo.DnsName = name and
                LSA_SID_DISABLED_ADMIN not in e.Flags and
                LSA_SID_DISABLED_CONFLICT not in e.Flags and
                ForestTrustOwnsName(o, e.DomainInfo.DnsName) then
            return true
          endif
        endfor
      endif
    endfor

    return false
```

```
procedure IsDomainNetbiosNameInTrustedForest
            (name: unicodestring): boolean

  tdos: set of DSName
  f: X_FOREST_TRUST_INFORMATION

  /* Get all the objects that represent trusted domains */
  tdos := select all o in Children ForestRootDomainNC() where
                  trustedDomain in o!objectClass and
                  o!trustAttributes & 0x00000008 ≠ 0 and
                  o!msDS-TrustForestTrustInfo ≠ null

  foreach o in tdos
    if not UnmarshalForestTrustInfo(o!msDS-TrustForestTrustInfo, f)
        then
      return false
    else
      foreach e in f.Entries
        if (e.ForestTrustType = ForestTrustDomainInfo and
              e.DomainInfo.NetbiosName = name and
              NETBIOS_DISABLED_MASK not in e.Flags and
              ForestTrustOwnsName(o, e.DomainInfo.DnsName) then
          return true
        endif
      endfor
    endif
  endfor

  return false
```

5.41 FOREST_TRUST_RECORD_TYPE

A concrete type for specifying the type of record contained in a forest trust information (FOREST_TRUST_INFORMATION) entry. The allowed values are specified by the following enumerated list.

```
typedef  enum
{
  ForestTrustTopLevelName = 0,
  ForestTrustTopLevelNameEx = 1,
  ForestTrustDomainInfo = 2,
  ForestTrustDomainInfo = ForestTrustDomainInfo
} FOREST_TRUST_RECORD_TYPE;
```

5.42 ForestRootDomainNC

```
procedure ForestRootDomainNC(): DSName
```

Returns the DSName of the forest root domain NC.

5.43 FullReplicaExists

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

Returns true if the NC replica with root nc is a full replica:

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

5.44 GetAttrVals

```
procedure GetAttrVals(
  o: DSName,
  att: ATTRTYP,
  includeDeletedLinks: boolean): set of attribute value
```

Constructs a set V containing each value of the attribute att from object o.

If att is not a link attribute, the value of includeDeletedLinks is ignored. If att is a link attribute and includeDeletedLinks = false, the set includes only those values v of att such that LinkStamp(o, att, v).timeDeleted = 0. If att is a link attribute and includeDeletedLinks = true, the set contains all values v of att, even those such that LinkStamp(o, att, v).timeDeleted ≠ 0.

If the V is empty, null is returned. Otherwise, V is returned.

5.45 GetDefaultObjectCategory

```
procedure GetDefaultObjectCategory(class: ATTRTYP): DSName
```

Returns the defaultObjectCategory value for the object class *class*.

```
classObj: DSName
classObj := SchemaObj(class)
return classObj!defaultObjectCategory
```

5.46 GetDSNameFromDN

```
procedure GetDSNameFromDN(dn: unicodestring): DSName
```

Produces a DSName from the DN dn. Let d represent the returned DSName. It is the case that d.dn = dn and d.guid = dn!objectGUID. Furthermore, if dn!objectSid ≠ null, then d.sid = dn!objectSid, otherwise d.sid has no value.

5.47 GetFSMORoleOwner

```
procedure GetFSMORoleOwner(role: integer): DSName
```

Returns the DSName of the nTDSDSA object of the DC that owns the FSMO role specified by role. The valid values for role are:

Symbolic constant	Value
FSMO_SCHEMA	1
FSMO_DOMAIN_NAMING	2
FSMO_PDC	3
FSMO_RID	4
FSMO_INFRASTRUCTURE	5

5.48 GetInstanceNameFromSPN

```
procedure GetInstanceNameFromSPN(spn: unicodestring): unicodestring
```

Syntactically extracts and returns the instance name from a two-part or three-part SPN. The instance name is the second part of the SPN. For example, dc-01.fabrikam.com is the instance name in the two-part SPN "ldap/dc-01.fabrikam.com" and in the three-part SPN "ldap/dc-01.fabrikam.com/fabrikam.com".

5.49 GetObjectNC

```
procedure GetObjectNC(o: DSName): DSName
```

Returns the DSName of the NC in which the object whose DSName is o is located.

5.50 GetServiceClassFromSPN

```
procedure GetServiceClassFromSPN(spn: unicodestring): unicodestring
```

Syntactically extracts and returns the service class from a two-part or three-part SPN. The service class is the first part of the SPN. For example, "ldap" is the service class in the two-part SPN "ldap/dc-01.fabrikam.com" and in the three-part SPN "ldap/dc-01.fabrikam.com/fabrikam.com".

5.51 GetServiceNameFromSPN

```
procedure GetServiceNameFromSPN(spn: unicodestring): unicodestring
```

Syntactically extracts and returns the service name from a three-part SPN. If the supplied SPN is a two-part SPN, it will return null. The service name is the third part of the SPN. For example, "fabrikam.com" is the service name in the three-part SPN "ldap/dc-01.fabrikam.com/fabrikam.com".

5.52 groupType Bit Flags

Bits which may appear in values of the groupType attribute defining a group type.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	AG	RG	UG	BG	QG	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	SE

X: Unused. MUST be zero and ignored.

AG (GROUP_TYPE_ACCOUNT_GROUP): Account group type.

RG (GROUP_TYPE_RESOURCE_GROUP): Resource group type.

UG (GROUP_TYPE_UNIVERSAL_GROUP): Universal group type.

BG (GROUP_TYPE_APP_BASIC_GROUP): Application basic group type.

QG (GROUP_TYPE_APP_QUERY_GROUP): Application query group type.

SE (GROUP_TYPE_SECURITY_ENABLED): Group is security enabled.

5.53 GUID

A concrete type, as specified in [C706] and [MS-DTYP] section **2.3.2**.

The type GUID has a well-defined null value, which is all zeros. The constant NULLGUID is equal to this value.

When comparing two GUID values, each GUID value is treated as an octet string in little-endian byte order.

Two GUID values g1 and g2 are equal if they are octet-for-octet identical.

Value g1 is less than value g2 if and only if there exists a N (where N is less than the size of GUID type in octets) such that octets 0...N-1 of g1 and g2 are identical, and octet N of g1 is less than octet N of g2.

Value g1 is greater than value g2 if and only if there exists a N (where N is less than the size of GUID type in octets) such that octets 0...N-1 of g1 and g2 are identical, and octet N of g1 is greater than octet N of g2.

5.54 GuidFromString

```
procedure GuidFromString(strGuid: unicodestring): GUID
```

Converts the string representation of a GUID specified in strGuid (for example, "{12AA5F43-C776-4D63-B347-1175DF806200}" or "12aa5f43-c776-4d63-b347-1175df806200") to a binary GUID. The string representation is that defined in [RFC4122] section 3, or such a representation prefixed with "{" and suffixed with "}". If strGuid is not a valid string representation of a GUID, null is returned.

5.55 GuidToString

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

Converts guid to the concatenation of "{", the string representation defined in [RFC4122] section 3, and "}"; for example, "{12aa5f43-c776-4d63-b347-1175df806200}".

5.56 handle_t

A concrete type for a RPC binding handle, as specified in [C706] section 4.2.9.7 and [MS-DTYP] section **2.1.2**.

5.57 instanceType Bit Flags

Bits which may appear in values of the instanceType attribute.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
NH	UI	WR	NA	NC	NG	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X: Unused. MUST be zero and ignored.

NH (IT_NC_HEAD): The object is the root of an NC.

UI (IT_UNINSTANT): The NC replica has not yet been instantiated.

WR (IT_WRITE): The object is writable.

NA (IT_NC_ABOVE): The DC hosts the NC above this one. IT_NC_HEAD must also be set.

NC (IT_NC_COMING): The NC replica is in the process of being constructed for the first time via replication. IT_NC_HEAD must also be set.

NG (IT_NC_GOING): The NC replica is in the process of being removed from the DC. IT_NC_HEAD must also be set.

5.58 Is2PartSPN

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

Returns true if spn is an SPN consisting of two parts, and false otherwise.

5.59 Is3PartSPN

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

Returns true if spn is an SPN consisting of three parts, and false otherwise.

5.60 IsBuiltinPrincipal

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

Returns true if sid is the SID of a **built-in security principal**, and returns false if it is not.

5.61 IsDomainNameInTrustedForest

```
procedure IsDomainNameInTrustedForest(name: unicodestring) : boolean
```

Returns true if the domain identified by *name* is in a forest trusted by the caller's forest as determined by FOREST_TRUST_INFORMATION state of the caller's forest, and false otherwise. *name* can be either a DNS name or a NetBIOS name of a domain.

See section 5.40 for the specification of this procedure.

5.62 IsDCAccount

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

Returns true if the object represents the computer account of a DC.

5.63 IsGC

```
procedure IsGC(): boolean
```

Returns true if the DC on which it is called is a global catalog server, and false if it is not.

5.64 IsGUIDBasedDNSName

```
procedure IsGUIDBasedDNSName(o: DSName, instanceName: unicodestring):
  boolean
```

Returns true if instanceName is the DNS host name of the DC, identified by o, constructed in the form "<DSA GUID>._msdcs.<DNS forest name>".

5.65 IsMemberOfBuiltinAdminGroup

```
procedure IsMemberOfBuiltinAdminGroup(): boolean
```

This function returns true if the client security context is a member of the BUILTIN\Administrators group, and false if it is not. The BUILTIN\Administrators group is the group with the SID S-1-5-32-544.

5.66 IsValidServiceName

```
procedure IsValidServiceName(o: DSName, serviceName: unicodestring):
```

```
boolean
```

Returns true if the name serviceName is a valid service name in an SPN for the DC represented by computer object o.

A valid service name can be one of the following:

1. For GC SPNs, the service name must be the DNS forest name.
2. For other classes of SPNs, the service name must be either the DNS domain name of the DC's default domain or the DNS domain name of an application NC hosted by the DC.

5.67 KCCFailedConnections

An abstract type consisting of a sequence of tuples, one tuple per each DC for which the connection attempt failed. Each tuple contains the following fields:

- **DsaDN**: A *unicodestring* (section 3.4.3) containing the DN of the nTDSDSA object corresponding to the DC.
- **UUIDDsa**: A GUID containing the DSA GUID of the DC.
- **TimeFirstFailure**: A FILETIME containing the time when KCC noticed the first failure while contacting the DC.
- **FailureCount**: An integer containing the total number of failures KCC has encountered while contacting the DC.
- **LastResult**: A DWORD containing a Windows error code indicating the reason for the last failure.

The global variable dc for a DC has an associated field dc.kccFailedConnections, which maintains the DC's KCCFailedConnections state.

5.68 KCCFailedLinks

An abstract type consisting of a sequence of tuples, one tuple for each neighboring DC for which a connection attempt failed.

The fields of the tuple are the same as the fields of the KCCFailedConnections tuple.

The global variable dc for a DC has an associated field dc.kccFailedLinks, which maintains the DC's KCCFailedLinks state.

5.69 LARGE_INTEGER

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

5.70 LDAP_CONN_PROPERTIES

A concrete type containing bit flags which identify properties of an LDAP connection.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
BND	SSL	UDP	GC	GSS	NGO	SPL	MD5	SGN	SL	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X: Unused. MUST be zero and ignored.

BND: A bind has been performed on this LDAP connection.

SSL: The LDAP connection corresponds to an SSL connection.

UDP: The LDAP connection corresponds to a UDP connection.

GC: The LDAP connection was made through the GC port.

GSS: GSSAPI security package was used for authentication.

NGO: SPNEGO security package was used for authentication.

SPL: The LDAP bind corresponds to LDAP simple bind.

MD5: **Digest**-MD5 security package was used for authentication.

SGN: Signing is enabled on the LDAP connection.

SL: Sealing is enabled on the LDAP connection.

5.71 LDAPConnections

An abstract type for the LDAP connections associated with a DC. It is a sequence of tuples, one tuple per LDAP connection currently open. Each tuple contains these fields:

- **iPAddress**: A DWORD containing the IPv4 address of the client machine that established this connection.
- **notificationCount**: An integer containing the number of LDAP notifications enabled on this connection.
- **secTimeConnected**: An integer containing the time in seconds that this connection has been open.
- **flags**: A DWORD containing LDAP_CONN_PROPERTIES bit flags which identify properties of this connection.
- **totalRequests**: An integer containing the total number of LDAP requests processed on this connection.
- **userName**: A *unicodestring* (section 3.4.3) containing the name of security principal that opened this connection.

The global variable dc for a DC has an associated field dc.ldapConnections, which maintains the DC's LDAPConnections state.

5.72 LinkStamp

```
procedure LinkStamp(
  o: DSName;
  att: ATTRTYP;
  val: attribute value): LinkValueStamp
```

Returns the LinkValueStamp associated with the last update to add or remove value val from the forward link attribute att of object o. If val was last updated when the forest functional level was DS_BEHAVIOR_WIN2000 (see [MS-ADTS] section 7.1.4.4), no LinkValueStamp is associated with val, and LinkStamp returns null.

5.73 LinkValueStamp

An abstract type which denotes information about the last update to a link value of an object. It is a tuple consisting of all the fields in AttributeStamp, plus the following additional fields:

- **timeCreated**: The date/time at which the first originating update was made.
- **timeDeleted**: The date/time at which the last replicated update was made that deleted the value, or 0 if the value is not currently deleted.

Comparisons

Values of LinkValueStamp are partially ordered. Let d be the result of x.dwVersion - y.dwVersion, cast as a 32-bit integer. Then given two stamps *x* and *y*, *x* is said to be "greater than" *y* if any of the following is true:

- x is not null and y is null
- x.timeCreated > y.timeCreated
- x.timeCreated = y.timeCreated and d > 0
- x.timeCreated = y.timeCreated and d = 0 and x.timeChanged > y.timeChanged
- x.timeCreated = y.timeCreated and d = 0 and x.timeChanged = y.timeChanged and x.uuidOriginating > y.uuidOriginating

5.74 LONG

A concrete type for a 32-bit signed integer, as specified in [MS-DTYP] section **2.2.25**.

5.75 LONGLONG

A concrete type for a 64-bit signed integer, as specified in [MS-DTYP] section **2.2.26**.

5.76 LPWSTR

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

5.77 MakeAttid

```
procedure MakeAttid(var t: PrefixTable, o: OID) : ATTRTYP
```

Translates an abstract OID o to a concrete **ATTRTYP**, using the **prefix table** specified by t. This procedure may mutate the supplied prefix table. See section 5.13 for the specification of this procedure.

5.78 MergeUTD

```
procedure MergeUTD(
  utd1: UPTODATE_VECTOR_V1_EXT,
  utd2: UPTODATE_VECTOR_V1_EXT): UPTODATE_VECTOR_V1_EXT
```

Informative summary of behavior: The client does not want to include objects in the inconsistency-detection process that have not yet replicated. To meet this goal, it uses MergeUTD to compute an **UPTODATE_VECTOR_V1_EXT** that has minimal pairwise values for each uuidDsa.

MergeUTD is specified by the following normative semantics:

For every uuidDsa that is in both utd1 and utd2, add a uuidDsa to the returned **UPTODATE_VECTOR_V1_EXT** *with a corresponding* USN *value such that the* USN *is smaller than the* USNs *corresponding to the uuidDsa in utd1 and utd2*.

5.79 MTX_ADDR

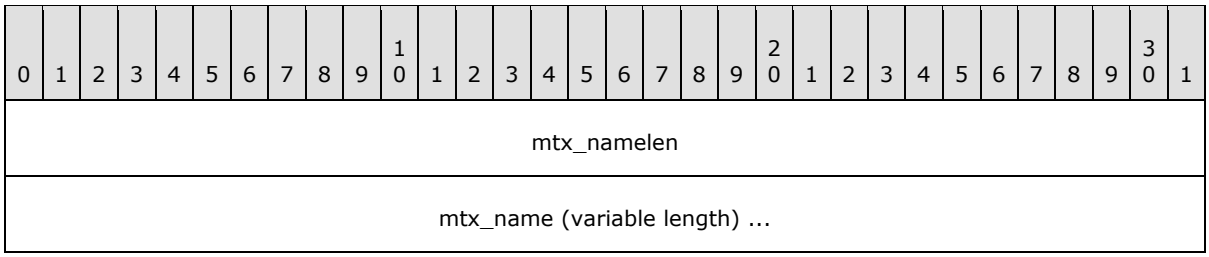
The **MTX_ADDR** structure defines a concrete type for the network name of a DC.

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

mtx_namelen: A 32-bit unsigned integer specifying the number of bytes in **mtx_name**, including a null terminating character.

mtx_name: UTF-8 encoding of a NetworkAddress.

The following table shows an alternative representation of this structure.



5.80 NetworkAddress

An abstract type for the transport-specific address of a DC represented as a UTF-8 string. For the RPC transport, the address is a DNS name (see [MS-ADTS] section 7.3.2.1). For the SMTP transport, the address is an SMTP address (as specified in [RFC2821]).

In AD/LDS, the NetworkAddress also includes the LDAP port of the destination AD/LDS DC. This data is used as RPC endpoint annotation to select the endpoint corresponding to the selected AD/LDS DC, since a machine can hold multiple AD/LDS DCs.

A NetworkAddress is stored as mtx_name within an MTX_ADDR structure, which in turn is stored within a REPS_TO structure.

5.81 NewPrefixTable

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

Creates a new PrefixTable that contains a set of default prefixes. See section 5.13 for the specification of this procedure.

5.82 NT4SID

The **NT4SID** structure defines a concrete type for a security identifier (SID).

```
typedef struct {
  char Data[28];
} NT4SID;
```

Data: Bytes that make up a SID structure, as specified in [MS-DTYP] section **2.4.2**, in little-endian byte order.

5.83 NTDSTRANSPORT_OPT Values

The valid system flags used on directory objects are specified in [MS-ADTS] section 7.1.

5.84 NULLGUID

A value of type GUID that is entirely zero, that is, {00000000-0000-0000-0000-000000000000}.

5.85 ObjExists

```
procedure ObjExists(dsName: DSName): boolean
```

Returns true if *dsName* identifies an object in some NC replica hosted by the server.

5.86 OID

An abstract type for representing values of type String(Object-Identifier). Values of this type are a dotted decimal *unicodestring* (section 3.4.3) (for example, "1.2.840.113556.1.4.159").

5.87 OID_t

The **OID_t** structure defines a concrete type for an OID or a prefix of an OID. A component of type **PrefixTableEntry**.

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

length: Size in bytes of the elements array.

elements: Array of bytes that constitute an OID or a prefix of an OID.

5.88 OidFromAttid

```
procedure OidFromAttid(t: PrefixTable, attr: ATTRTYP) : OID
```

Translates a concrete **ATTRTYP** attr to an abstract OID, using the prefix table specified by t. See section 5.13 for the specification of this procedure.

5.89 parent

This is an abstract attribute, present on every object, as specified in [MS-ADTS] section 3.1.1.1.3. This attribute is part of the state model but is not exposed in the Active Directory schema.

5.90 PARTIAL_ATTR_VECTOR_V1_EXT

The **PARTIAL_ATTR_VECTOR_V1_EXT** structure defines a concrete type for a set of attributes to be replicated to a given partial replica.

```
typedef struct {
  DWORD dwVersion;
  DWORD dwReserved1;
  [range(1,1048576)] DWORD cAttrs;
  [size_is(cAttrs)] ATTRTYP rgPartialAttr[];
} PARTIAL_ATTR_VECTOR_V1_EXT;
```

dwVersion: Version of this structure; MUST be 1.

dwReserved1: Reserved; MUST be 0 and ignored.

cAttrs: Count of attributes in the rgPartialAttr array.

rgPartialAttr: Attributes in the set.

5.91 partialAttributeSet

The abstract non-replicated single-valued attribute partialAttributeSet is an optional attribute on the NC root of every partial replica.

The abstract type set of **ATTRTYP** simplifies the specification of methods that read and write the attribute partialAttributeSet. Reading the attribute partialAttributeSet returns a single value, which is of type set of **ATTRTYP**. Each element in the set is an attribute that is in the subset of attributes replicated to the partial replica.

5.92 PartialGCReplicaExists

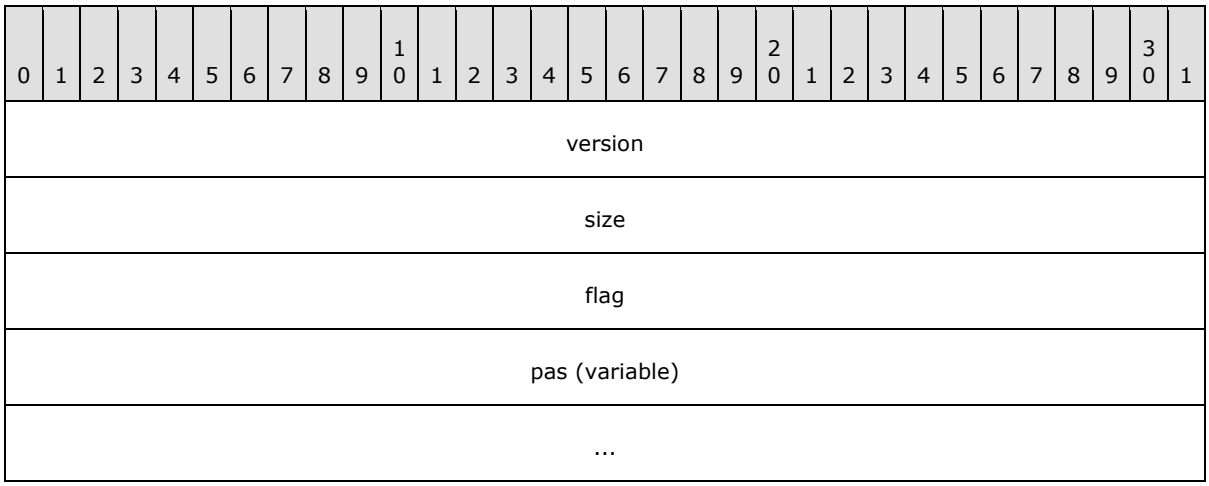
```
procedure PartialGCReplicaExists(nc : DSName) : boolean
```

Returns true if the NC replica with root nc is a partial NC replica.

```
if not ObjExists(nc) then
  return false
endif
return nc in DSAObj()!hasPartialReplicaNCs
```

5.93 PAS_DATA

A concrete type for a list of attributes in a **partial attribute set**.



version (4 bytes): Version of the structure; MUST be 1.

size (4 bytes): Size of the entire structure.

flag (4 bytes): Not used; MUST be 0.

pas (variable): A **PARTIAL_ATTR_VECTOR_V1_EXT** structure, which specifies the additional attributes being requested as part of a PAS **cycle**.

5.94 PerformReplication

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

Executes the tasks necessary for replicating the specified NC from the specified DC.

5.95 PrefixTable

An abstract type for a prefix table. See section 5.13 for the specification of this type.

5.96 PrefixTableEntry

The **PrefixTableEntry** structure defines a concrete type for mapping of a range of **ATTRTYP** values to and from OIDs. It is a component of the type **SCHEMA_PREFIX_TABLE**.

```
typedef struct {
  unsigned long ndx;
  OID_t prefix;
} PrefixTableEntry;
```

ndx: Index assigned to the prefix.

prefix: An OID or a prefix of an OID.

5.97 RDN

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

5.98 rdnType

An abstract attribute present on every object. The rdnType of an object is the RDN attribute of the object, i.e. an **ATTRTYP** identifying the attributeSchema object of the RDN attribute. rdnType is not represented in the schema and does not replicate in the normal way.

On an originating Add, the new object's rdnType is derived from the most specific structural object class of the new object.

On a replicated Add, rdnType is derived as follows:

- If the forest functional level is less than DS_BEHAVIOR_WIN2003, rdnType is derived from the objectClass of the object, which replicates.
- If the forest functional level is DS_BEHAVIOR_WIN2003 or greater, rdnType is derived from the DN of the object, which replicates.

5.99 RemoveObj

```
procedure RemoveObj(o: DSName)
```

Deletes (into a tombstone) the object whose DSName is o from the directory.

5.100 ReplicationQueue

The server-to-server replication implementation is synchronized on a queue for inbound replication tasks. This is an abstract type for queued pending replication operations. It is a sequence of tuples, one tuple per each queued replication operation that is pending. Each tuple contains the following fields:

- **TimeEnqueued**: A FILETIME containing the time when the operation was enqueued.
- **SerialNumber**: A ULONG containing a unique identifier associated with the operation.
- **Priority**: A ULONG containing the priority of the operation.
- **OperationType**: An integer that indicates the type of operation as defined in **DS_REPL_OP_TYPE**.
- **Options**: A **DRS_OPTIONS** containing options associated with the replication operation.
- **NamingContext**: A *unicodestring* (section 3.4.3) containing the NC root of the NC replica associated with the operation.
- **DsaDN**: A *unicodestring* (section 3.4.3) containing the DN of the nTDSDSA object of the DC associated with the operation.
- **DsaAddress**: A *unicodestring* (section 3.4.3) containing the network address of the DC associated with the operation.
- **UUIDNC**: A GUID containing the objectGUID of the NC root of the NC replica associated with the operation.
- **UUIDDsa**: A GUID containing the DSA GUID of the DC associated with the operation.

The global variable dc for a DC has an associated field dc.replicationQueue, which maintains the DC's ReplicationQueue state.

5.101 REPLTIMES

The **REPLTIMES** structure defines a concrete type for times at which periodic replication occurs.

```
typedef struct {
  UCHAR rgTimes[84];
} REPLTIMES;
```

rgTimes: An 84-byte structure used to set periodic replication times. Each bit in this structure represents a 15-minute period for which replication can be scheduled within a one-week period. The replication schedule begins on Sunday at 12:00am UTC.

The following diagram shows an alternative representation of this structure.

5.102 replUpToDateVector, ReplUpToDateVector

The abstract non-replicated multi-valued attribute replUpToDateVector is an optional attribute on the NC root of every NC replica.

The abstract type ReplUpToDateVector simplifies the specification of methods that read and write the attribute replUpToDateVector. Reading the attribute replUpToDateVector produces one or more ReplUpToDateVector values.

The type ReplUpToDateVector is a tuple with these three fields:

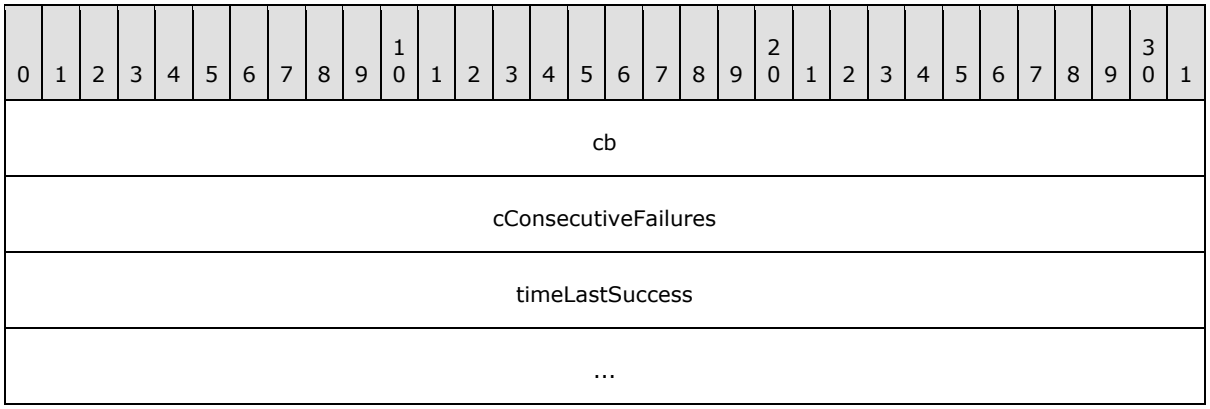
- **uuidDsa**: **Invocation ID** of the DC which assigned usnHighPropUpdate.
- **usnHighPropUpdate**: A USN at which an update was applied on the DC identified by uuidDsa.
- **timeLastSyncSuccess**: Time at which the last successful replication occurred from the DC identified by uuidDsa; for replication latency reporting only.

Given an NC replica r, if c is an element of r!replUpToDateVector then all updates made by c.uuidDsa with USN <= c.usnHighPropUpdate have been applied to r.

5.103 REPS_FROM

A concrete type. The non-replicated multi-valued attribute repsFrom is an optional attribute on the root object of every NC replica. It is stored with the structure REPS_FROM, which is represented below as a packet diagram. This attribute exposes a structure that controls the server-to-server replication implementation—specifically, the options for inbound replication of changes.

Note In the below field descriptions, the source DC refers to the DC identified by the uuidDsaObj.



timeLastAttempt
...
ulResultLastAttempt
cbOtherDraOffset
cbOtherDra
ulReplicaFlags
rtSchedule
...
...
...
...
...
...
...
(rtSchedule cont'd for 13 rows)
usnVec
...
...
...
...
...

uuidDsaObj
...
...
...
uuidInvocId
...
...
...
uuidTransportObj
...
...
...
dwReserved
cbPasDataOffset
data (variable)
...

cb (4 bytes): Total number of bytes in the REPS_FROM structure.

cConsecutiveFailures (4 bytes): An unsigned long. Contains the number of consecutive failures that have occurred while replicating from the source DC.

timeLastSuccess (8 bytes): A **DSTIME**. Contains the time of the last successful replication with the source DC.

timeLastAttempt (8 bytes): A **DSTIME**. Contains the time of the last replication attempt with the source DC.

ulResultLastAttempt (4 bytes): An unsigned long. Contains the result of the last replication attempt with the source DC.

cbOtherDraOffset (4 bytes): Offset from the start of the structure to a location in the data field, specifying the start of an **MTX_ADDR** structure containing a NetworkAddress for the source DC.

cbOtherDra (4 bytes): Size of the **MTX_ADDR** structure pointed to by **cbOtherDraOffset**.

ulReplicaFlags (4 bytes): A **ULONG**. This field contains a set of **DRS_OPTIONS** that are applicable when replicating from the source DC.

rtSchedule (84 bytes): A **REPLTIMES** structure. If periodic replication is enabled (ulReplicaFlags contains DRS_PER_SYNC), this field identifies the 15-minute intervals within each week when a **replication cycle** is initiated with the source DC.

usnVec (24 bytes): A **USN_VECTOR** structure.

uuidDsaObj (16 bytes): A GUID. This is the DSA GUID of the source DC.

uuidInvocId (16 bytes): A GUID. Contains the invocation ID of the source DC.

uuidTransportObj (16 bytes): A GUID. Contains the objectGUID of the interSiteTransport object corresponding to the transport used for communication with the source DC.

dwReserved (4 bytes): Not used; MUST be 0.

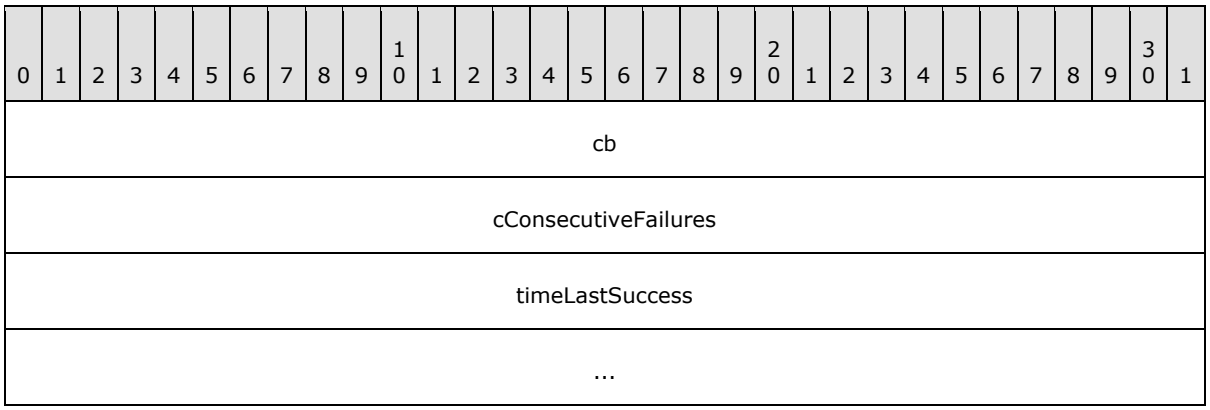
cbPasDataOffset (4 bytes): Offset from the start of the structure to a location in the data field, specifying the start of a PAS_DATA structure.

data (variable): Storage for the rest of the structure. The **MTX_ADDR** structure and the PAS_DATA structure pointed to by **cbOtherDraOffset** and **cbPasDataOffset** are packed into this field and can be referenced using the offsets.

5.104 REPS_TO

A concrete type. The non-replicated multi-valued attribute repsTo is an optional attribute on the root object of every NC replica. It is stored as the structure REPS_TO, which is represented below as a packet diagram. This attribute exposes a structure that controls the server-to-server replication implementation—specifically, the options for outbound replication of changes.

This structure is used for both repsTo values and repsFrom values. Many of the fields are unused in repsTo values, and some of the field names are misleading (for example, **timeLastSuccess**).



timeLastAttempt
...
ulResultLastAttempt
cbOtherDraOffset
cbOtherDra
ulReplicaFlags
rtSchedule
...
...
...
...
...
...
...
(rtSchedule cont'd for 13 rows)
usnVec
...
...
...
...
...

uuidDsaObj
...
...
...
uuidInvocId
...
...
...
uuidTransportObj
...
...
...
dwReserved
cbPasDataOffset
data (variable)
...

cb (4 bytes): Total number of bytes in the REPS_TO structure.

cConsecutiveFailures (4 bytes): An unsigned long. Contains the number of consecutive attempts to send a replication notification to the DC identified by uuidDsaObj that have been unsuccessful.

timeLastSuccess (8 bytes): A **DSTIME** structure. Contains the time when the last successful replication notification to the DC identified by uuidDsaObj was sent, or 0 if no replication notification has been sent successfully.

timeLastAttempt (8 bytes): A **DSTIME** structure. Contains the last time when an attempt was made to send a replication notification to the DC identified by uuidDsaObj, or 0 if no attempt has been made.

ulResultLastAttempt (4 bytes): An unsigned long. Contains the result of the last attempt to send a replication notification to the DC identified by uuidDsaObj. It has a value 0 if the last notification was sent successfully, a Windows error code otherwise.

cbOtherDraOffset (4 bytes): Offset from the start of the structure to a location in the data field, specifying the start of an **MTX_ADDR** structure containing a NetworkAddress.

cbOtherDra (4 bytes): Size of the **MTX_ADDR** structure pointed to by **cbOtherDraOffset**.

ulReplicaFlags (4 bytes): A **ULONG**. This set contains DRS_WRIT_REP if this replica is writable. This set never contains any other elements.

rtSchedule (84 bytes): A **REPLTIMES** structure. Not used.

usnVec (24 bytes): A **USN_VECTOR** structure. Not used.

uuidDsaObj (16 bytes): A GUID. A DSA GUID that identifies a DC.

uuidInvocId (16 bytes): A GUID. Not used.

uuidTransportObj (16 bytes): A GUID. Not used.

dwReserved (4 bytes): Not used; MUST be 0.

cbPasDataOffset (4 bytes): Not used.

data (variable): Storage for the rest of the structure. The **MTX_ADDR** structure pointed to by **cbOtherDraOffset** is packed into this field and can be referenced using the offset.

5.105 repsFrom, RepsFrom

The non-replicated multi-valued attribute repsFrom is an optional attribute on the root object of every NC replica. It is stored with the structure REPS_FROM.

The abstract type RepsFrom simplifies the specification of methods that read and write the attribute repsFrom. Reading the attribute repsFrom produces one or more RepsFrom values using the conversions from REPS_FROM specified below. Writing a RepsFrom value to the attribute repsFrom stores a REPS_FROM using the reverse conversion.

The type RepsFrom is a tuple with the following fields:

- **naDsa**: A NetworkAddress, corresponds to cbOtherDraOffset and cbOtherDra in REPS_FROM. This is a NetworkAddress of the DC.
- **uuidDsa**: A GUID, corresponds to uuidDsaObj in REPS_FROM. This is the DSA GUID of the DC.
- **options**: A ULONG, corresponds to ulReplicaFlags in REPS_FROM. These flags are used by the server-to-server replication implementation. This set contains one or more of the following values chosen from **DRS_OPTIONS**:
  - DRS_WRIT_REP: The replica is a full (read/write) replica of the NC.
  - DRS_INIT_SYNC: The replica must be replicated from the DC identified by uuidDsa when the DC hosting this replica is started.
  - DRS_PER_SYNC: Replicate the NC replica from DC identified by uuidDsa periodically, as defined by the periodic replication schedule.

- DRS_MAIL_REP: Replicate the NC replica from DC identified by uuidDsa via SMTP.
- DRS_DISABLE_AUTO_SYNC: Disable notification–based replication of the NC replica from DC identified by uuidDsa.
- DRS_DISABLE_PERIODIC_SYNC: Disable periodic replication of the NC replica from DC identified by uuidDsa.
- DRS_USE_COMPRESSION: Replication response messages sent along this communication path should be compressed.
- DRS_TWOWAY_SYNC: At the end of a replication attempt, replication should be triggered in the opposite direction.

- **schedule**: A **REPLTIMES**, corresponds to rtSchedule in REPS_FROM. This contains the periodic replication schedule.
- **uuidInvocId**: A GUID. Contains the invocation ID of the source DC.
- **usnVec**: A **USN_VECTOR**, corresponds to the usnVec in REPS_FROM.
- **uuidTransport**: A GUID, corresponds to uuidTransportObj in REPS_FROM. This is the objectGUID of the interSiteTransport object corresponding to the transport used for communication with the DC identified by uuidDsa.
- **consecutiveFailures**: A DWORD, corresponds to cbConsecutiveFailures in REPS_FROM. It is the number of consecutive failures during replication from the DC identified by uuidDsa.
- **timeLastSuccess**: A DWORD, corresponds to timeLastSuccess in REPS_FROM. It is the time of the last successful replication from the DC identified by uuidDsa.
- **timeLastAttempt**: A DWORD, corresponds to timeLastAttempt in REPS_FROM. It is the time of the last replication attempt with the DC identified by uuidDsa.
- **resultLastAttempt**: Result of last replication attempt with the DC identified by uuidDsa.
- **pasData**: A PAS_DATA value that corresponds to **cbPasDataOffset** in REPS_FROM. Contains the list of attributes (being added to the partial attribute set for the NC on this DC) that are being requested from the DC identified by uuidDsa.

When converting a RepsFrom to a REPS_FROM, assign zeros to all unused fields of REPS_FROM. If naDsa is an empty string, set cbOtherDra to 0 and cbOtherDraOffset to 0. If pasData is not present, set cbPasDataOffset to 0.

5.106 repsTo, RepsTo

The non-replicated multi-valued attribute repsTo is an optional attribute on the root object of every NC replica. It is stored as the structure REPS_TO.

The abstract type RepsTo simplifies the specification of methods that read and write the attribute repsTo. Reading the attribute repsTo produces one or more RepsTo values using the conversions from REPS_TO specified below. Writing a RepsTo value to the attribute repsTo stores a REPS_TO using the reverse conversion.

The type RepsTo is a tuple with the following fields:

- **naDsa**: A NetworkAddress, corresponding to cbOtherDraOffset and cbOtherDra in REPS_TO. This is the NetworkAddress of a DC.

- **uuidDsa**: A GUID, corresponding to uuidDsaObj in REPS_TO. This is the DSA GUID of the target DC.
- **options**: Bit flags chosen from **DRS_OPTIONS**, corresponding to ulReplicaFlags in REPS_TO. This set contains the DRS_WRIT_REP value if this replica is writable. This set never contains any other elements.
- **resultLastAttempt**: A DWORD, corresponding to ulResultLastAttempt in REPS_TO. Contains the result of the last attempt to send a replication notification to the DC identified by uuidDsa. It has a value 0 if the last notification was sent successfully and a Windows error code otherwise.
- **consecutiveFailures**: A DWORD, corresponding to cConsecutiveFailures in REPS_TO. Contains the number of consecutive attempts to send a replication notification to the DC identified by uuidDsa that have been unsuccessful.
- **timeLastAttempt**: A **DSTIME**, corresponding to timeLastAttempt in REPS_TO. Contains the last time when an attempt was made to send a replication notification to the DC identified by uuidDsa, or 0 if no attempt has been made.
- **timeLastSuccess**: A **DSTIME**, corresponding to timeLastSuccess in REPS_TO. Contains the time when the last successful replication notification to the DC identified by uuidDsa was sent, or 0 if no replication notification has been sent successfully.

When converting a RepsTo to a REPS_TO, assign zeros to all unused fields of REPS_TO. If naDsa is an empty string, set cbOtherDra to 0 and cbOtherDraOffset to 0.

5.107 Rid

An abstract type consisting of an integer that represents the relative identifier (RID) component of a SID, as specified in [MS-DTYP] section **2.4.2** and [MS-SECO] section 2.1.2.

5.108 Right

An abstract type which represents an access right (for example, RIGHT_DS_WRITE_PROPERTY) or a control access right (for example, DS-Replication-Manage-Topology) on an object. The complete set of access right values is specified in [MS-ADTS] section 5.1.3.2, and the complete set of control access right values is specified in [MS-ADTS] section 5.1.3.2.1. Note that since access rights and control access rights are non-overlapping sets, there is no ambiguity in having one type represent rights of both kinds.

5.109 RIGHT Values

The valid access rights used in ACLs in security descriptors are defined in [MS-ADTS] section 5.1.3.2.

5.110 RPCClientContexts

An abstract type that is a sequence of tuples, one tuple per RPC context for an incoming RPC session to the DC. Each tuple contains these fields:

- **BindingContext**: A ULONGLONG containing a unique identifier for the context.
- **RefCount**: An integer used to reference count the number of references to the context.
- **IsBound**: A Boolean value that is true if IDL_DRSUnbind has not yet been called on the RPC context represented by this tuple, and false otherwise.

- **UUIDClient**: A GUID containing the value which was passed in as the puuidClientDsa argument of DL_DRSBind while establishing the context.
- **TimeLastUsed**: A FILETIME containing the last time a session corresponding to the context was used in an RPC method call.
- **IPAddress**: A DWORD containing the IPv4 address of the client associated with the context.
- **PID**: An integer containing the process ID passed in by the client as the pextClient argument of IDL_DRSBind while establishing the context.

The global variable dc for a DC has an associated field dc.rpcClientContexts, which maintains the DC's RPCClientContexts state.

5.111 RPCOutgoingContexts

An abstract type that is a sequence of tuples, one tuple per RPC context for an outgoing server-to-server RPC session from the DC to another DC. This implementation uses a cache, some of the fields in the tuples reference this implementation. Each tuple contains the following fields:

- **ServerName**: A *unicodestring* (section 3.4.3) that contains the host name of the server.
- **IsBound**: A Boolean value that is true if IDL_DRSUnbind has not yet been called on the RPC context represented by this tuple, and false otherwise.
- **HandleFromCache**: A Boolean value that is true if the context handle was retrieved from the cache, and false otherwise.
- **HandleInCache**: A Boolean value that is true if the context handle is still in the cache, and false otherwise.
- **ThreadId**: An integer that contains an implementation specific thread ID of the thread that is using the context.
- **BindingTimeOut**: An integer. If the context is set to be canceled, then this field contains the timeout in minutes.
- **CreateTime**: A **DSTIME** value that contains the time when the context was created.
- **CallType**: An integer that indicates the type of RPC call that the DC is waiting on. See DS_REPL_SERVER_OUTGOING_CALL for possible values.

The global variable dc for a DC has an associated field dc.rpcOutgoingContexts, which maintains the DC's RPCOutgoingContexts state.

5.112 sAMAccountType Values

Values that describe information about various account type objects.

Symbolic name	Value
SAM_GROUP_OBJECT	0x10000000
SAM_NON_SECURITY_GROUP_OBJECT	0x10000001
SAM_ALIAS_OBJECT	0x20000000

Symbolic name	Value
SAM_NON_SECURITY_ALIAS_OBJECT	0x20000001
SAM_USER_OBJECT	0x30000000
SAM_NORMAL_USER_ACCOUNT	0x30000000
SAM_MACHINE_ACCOUNT	0x30000001
SAM_TRUST_ACCOUNT	0x30000002

5.113 SCHEMA_PREFIX_TABLE

The **SCHEMA_PREFIX_TABLE** structure defines the concrete type for a table to map **ATTRTYP** values to and from OIDs.

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

PrefixCount: Count of items in the **pPrefixEntry** array.

pPrefixEntry: Array of **PrefixTableEntry** items in the table.

5.114 SchemaNC

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

This procedure returns the DSName of dc.schemaNC.

5.115 SchemaObj

```
procedure SchemaObj(att: ATTRTYP): DSName
```

Given the **ATTRTYP** att of an attributeSchema or classSchema object on this DC, returns the dsname of the attributeSchema or classSchema object.

```
return select one o from children SchemaNC()
  where AttrtypFromSchemaObj(o) = att
```

5.116 Server Extensions

```
procedure ServerExtensions(hDrs: DRS_HANDLE): DRS_EXTENSIONS_INT
```

Returns the server extensions presented in the IDL_DRSBind call that created hDrs. Any fields not specified by the server in the *ppextServer^* parameter of IDL_DRSBind are set to 0.

5.117 SID

A concrete type for the Windows NT **SID** structure, as specified in [MS-DTYP] section **2.4.2**.

5.118 SidFromStringSid

```
procedure SidFromStringSid(stringSID: unicodestring): SID
```

Converts the string representation of a SID specified in stringSID (for example, S-1-5-3) to the SID type, as specified in [MS-DTYP] section **2.4.2**. See [MS-SECO] section 2.1.2 for the string representation of a SID.

5.119 StampLessThanOrEqualUTD

```
procedure StampLessThanOrEqualUTD(
  stamp: AttributeStamp,
  utd: UPTODATE_VECTOR_V1_EXT) : boolean
```

Informative summary of behavior: The StampLessThanOrEqualUTD procedure is used to determine if an attribute has already replicated (or should have already replicated) in the server-to-server replication implementation.

```
i: integer

for i := 0 to utd.cNumCursors - 1
  if utd.rgCursors[i].uuidDsa = stamp.uuidOriginating) and
      (utd.rgCursors[i].usn >= stamp.usnOriginating) then
    return true
  endif
endfor
return false
```

5.120 StartsWith

```
procedure StartsWith(s: unicodestring, p: unicodestring): boolean
```

Returns true if the string p is a prefix of string s, and returns false otherwise.

5.121 STATUS Codes

Windows status code. Other nonzero codes are fatal errors.

Symbolic name	Value
STATUS_ACCESS_DENIED	0xC0000022
STATUS_MORE_ENTRIES	0x00000105
STATUS_TOO_MANY_CONTEXT_IDS	0xC000015A
STATUS_INSUFFICIENT_RESOURCES	0xC000009A

Symbolic name	Value
STATUS_BUFFER_TOO_SMALL	0xC0000023
STATUS_INVALID_PARAMETER	0xC000000D

5.122 StringSidFromSid

```
procedure StringSidFromSid(sid: SID): unicodestring
```

Converts a binary SID specified in sid to the string representation of a SID (for example, S-1-5-3). See [MS-SECO] section 2.1.2 for the string representation of a SID.

5.123 SubString

```
procedure SubString(
  s: unicodestring, start: integer, length: integer): unicodestring
```

Returns the portion of s beginning at zero-based index start and containing length characters. If start is less than zero or greater than s.length-1, returns null. If length + start is greater than s.length, then length is treated as if it equals s.length - start.

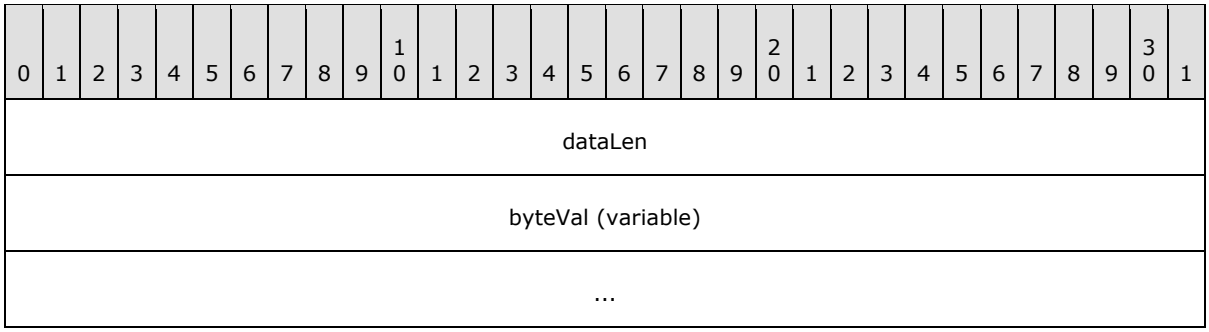
5.124 Syntax

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

Returns the syntax of the attribute attr.

5.125 SYNTAX_ADDRESS

The SYNTAX_ADDRESS packet is the concrete type for a sequence of bytes or Unicode characters.



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

byteVal (variable): Byte or character data.

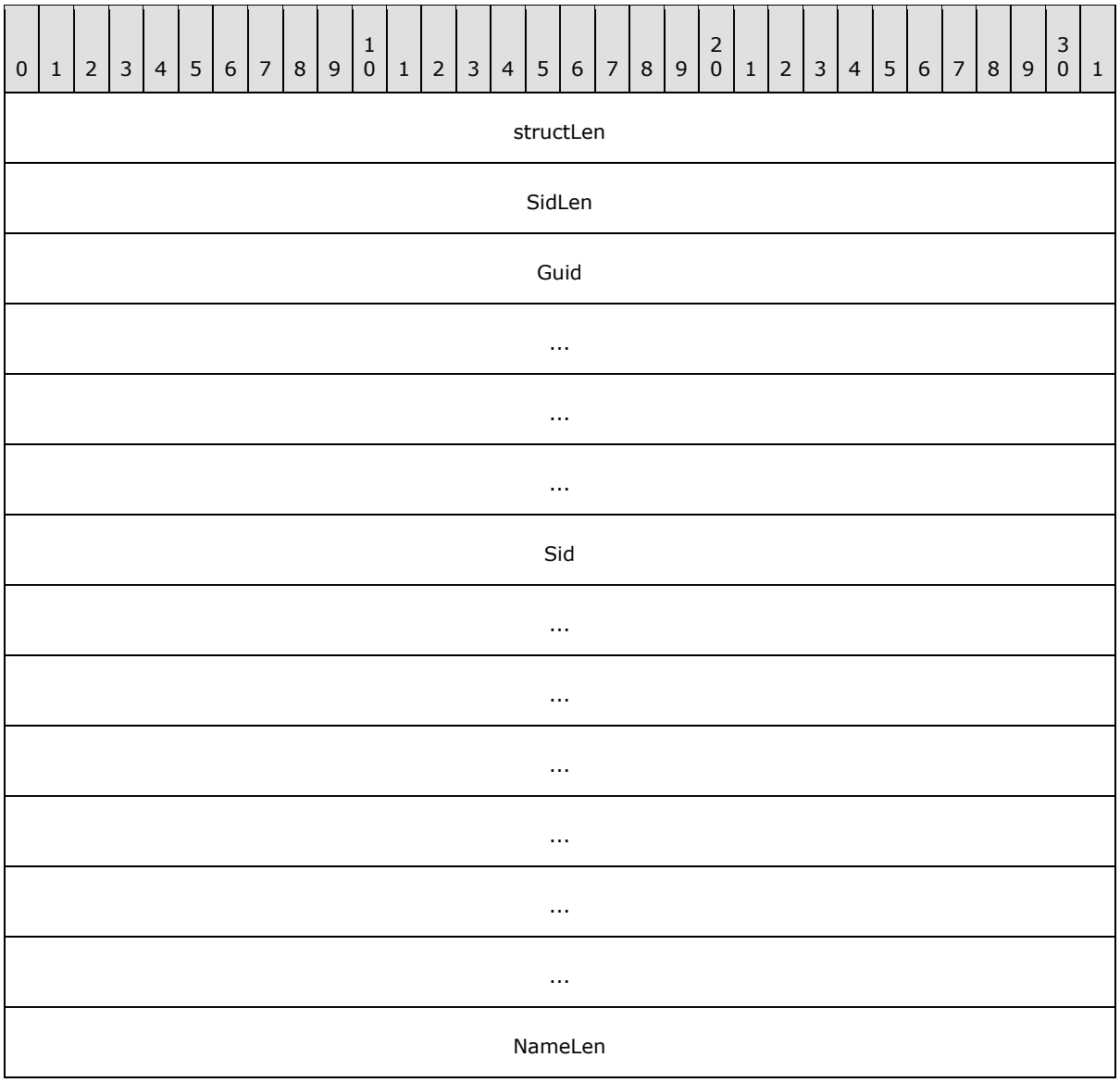
The following structure definition shows an alternative representation of this data type.

```
typedef struct {
```

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

5.126 SYNTAX_DISTNAME_BINARY

The SYNTAX_DISTNAME_BINARY packet is the concrete type for a combination of a **DSNAME** and a binary or character data buffer.



StringName (variable)
...
Padding (variable)
...
dataLen
byteVal (variable)
...

structLen (4 bytes): Length of the structure, in bytes, up to and including the field StringName.

SidLen (4 bytes): Number of bytes in the Sid field used to represent the object's objectSid attribute value. Zero indicates that the SYNTAX_DISTNAME_BINARY does not identify the objectSid value of the directory object.

Guid (16 bytes): The value of the object's objectGUID attribute specified as a GUID structure, which is defined in [MS-DTYP] section **2.3.2**. Zero values for all fields in the GUID structure indicates that the SYNTAX_DISTNAME_BINARY does not identify the objectGUID value of the directory object.

Sid (28 bytes): The value of the object's objectSid attribute, its security identifier (see [MS-SECO] section 2.1.2), specified as a SID structure, which is defined in [MS-DTYP] section **2.4.2**. The size of this field is exactly 28 bytes, regardless of the value of SidLen, which specifies how many bytes in this field are used.

NameLen (4 bytes): Number of characters in the StringName field, not including the null terminator, used to represent the object's distinguishedName attribute value. Zero indicates that the SYNTAX_DISTNAME_BINARY does not identify the distinguishedName value of the directory object.

StringName (variable): Null-terminated Unicode value of the object's distinguishedName attribute, as specified in [MS-ADTS] section 3.1.1.1.4. This field always contains at least one character: the null terminator. Each Unicode value is encoded as 2 bytes. The byte ordering is little-endian.

Padding (variable): Padding (bytes with value zero) to align the field dataLen at a double word boundary.

dataLen (4 bytes): Length of the remaining structure including this field, in bytes.

byteVal (variable): Array of bytes.

Note All fields have little-endian byte ordering.

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

```
typedef struct {
    DSNAME Name;
    SYNTAX_ADDRESS Data;
} SYNTAX_DISTNAME_BINARY;
```

5.127 systemFlags Values

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

5.128 UCHAR

A concrete type, as defined in [MS-DTYP] section **2.2.45**. A UCHAR is an 8-bit unsigned quantity.

5.129 ULONG

A concrete type for a 32-bit unsigned integer, as specified in [MS-DTYP] section **2.2.51**.

5.130 ULONGLONG

A concrete type for a 64-bit unsigned integer, as specified in [MS-DTYP] section **2.2.55**.

5.131 UNICODE_STRING

A concrete type for a counted double-byte Unicode string, as specified in [MS-DTYP] section **2.3.9**.

5.132 UPTODATE_CURSOR_V1

The **UPTODATE_CURSOR_V1** structure is a concrete type for the replication state relative to a given DC.

```
typedef struct {
  UUID uuidDsa;
  USN usnHighPropUpdate;
} UPTODATE_CURSOR_V1;
```

uuidDsa: invocationId of the DC performing the update.

usnHighPropUpdate: USN of the update on the updating DC.

A cursor c with c.uuidDsa = x and c.usnHighPropUpdate = y indicates a replication state that is inclusive of all changes originated by DC x at USN less than or equal to y.

5.133 UPTODATE_CURSOR_V2

The **UPTODATE_CURSOR_V2** structure defines a concrete type for the replication state relative to a given DC.

```
typedef struct {
  UUID uuidDsa;
  USN usnHighPropUpdate;
  DSTIME timeLastSyncSuccess;
} UPTODATE_CURSOR_V2;
```

uuidDsa: invocationId of the DC performing the update.

usnHighPropUpdate: Update sequence number (USN) of the update on the updating DC.

timeLastSyncSuccess: Time at which the last successful replication occurred from the DC identified by **uuidDsa**; for replication latency reporting only.

A cursor c with c.uuidDsa = x and c.usnHighPropUpdate = y indicates a replication state that is inclusive of all changes originated by DC x at USN less than or equal to y.

5.134 UPTODATE_VECTOR_V1_EXT

The **UPTODATE_VECTOR_V1_EXT** structure defines a concrete type for the replication state relative to a set of DCs.

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

dwVersion: Version of this structure; MUST be set to 1.

dwReserved1: Reserved; MUST be set to 0 and ignored.

cNumCursors: Count of items in the **rgCursors** array.

dwReserved2: Reserved; MUST be set to 0 and ignored.

rgCursors: An array of **UPTODATE_CURSOR_V1**. The items in this field MUST be sorted in increasing order of the **uuidDsa** field.

5.135 UPTODATE_VECTOR_V2_EXT

The **UPTODATE_VECTOR_V2_EXT** structure defines a concrete type for the replication state relative to a set of DCs.

```
typedef struct {
  DWORD dwVersion;
  DWORD dwReserved1;
  [range(0,1048576)] DWORD cNumCursors;
  DWORD dwReserved2;
  [size_is(cNumCursors)] UPTODATE_CURSOR_V2 rgCursors[];
} UPTODATE_VECTOR_V2_EXT;
```

dwVersion: Version of this structure; MUST be set to 2.

dwReserved1: Reserved; MUST be set to 0 and ignored.

cNumCursors: Count of items in the **rgCursors** array.

dwReserved2: Reserved; MUST be set to 0 and ignored.

rgCursors: An array of **UPTODATE_CURSOR_V2**. The items in this field MUST be sorted in increasing order of the **uuidDsa** field.

5.136 userAccountControl Bits

Bit flags describing various qualities of a security account.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	AD	X	X	LO	X	X	X	DA	NA	X	ID	WT	ST	X	X	X	X	X	X	X	X	X	X	X	X	X	PS	X	X	X	X

X: Unused. MUST be zero and ignored.

AD (ADS_UF_ACCOUNTDISABLE): The account is disabled.

LO (ADS_UF_LOCKOUT): The account is temporarily locked out.

DA (ADS_UF_TEMP_DUPLICATE_ACCOUNT): This is an account for a user whose primary account is in another domain.

NA (ADS_UF_NORMAL_ACCOUNT): Default account type that represents a typical user.

ID (ADS_UF_INTERDOMAIN_TRUST_ACCOUNT): Account for a domain-to-domain trust.

WT (ADS_UF_WORKSTATION_ACCOUNT): Computer account for a computer that is a member of this domain.

ST (ADS_UF_SERVER_TRUST_ACCOUNT): Computer account for a DC.

PS (ADS_UF_PARTIAL_SECRETS_ACCOUNT): Computer account for an RODC.

5.137 UserNameFromNT4AccountName

```
procedure UserNameFromNT4AccountName(
  nt4AccountName: unicodestring): unicodestring
```

If nt4AccountName is a name in Windows NT 4.0 account name format, that is, two components separated by a backslash (for example, "DOMAIN\username"), this procedure returns the second component (the user name, or "username" in this example). If the nt4AccountName is not in this format, null is returned.

5.138 USHORT

A concrete type for a 16-bit unsigned integer, as specified in [MS-DTYP] section **2.2**.

5.139 USN

A concrete type for the variable *usn* specified in [MS-ADTS] section 3.1.1.1.9 and present in the dc global variable.

This type is declared as follows:

```
typedef LONGLONG USN;
```

5.140 USN_VECTOR

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

```
typedef struct {
  USN usnHighObjUpdate;
  USN usnReserved;
  USN usnHighPropUpdate;
} USN_VECTOR;
```

usnHighObjUpdate: An update sequence number (USN).

usnReserved: A USN.

usnHighPropUpdate: A USN.

The **USN_VECTOR** type as shown is used in the DRS IDL. All uses of this structure are implementation-specific.

5.141 UUID

A type that is equivalent to the GUID type.

5.142 Value

An abstract type for attribute values used for abstract value representation (see section 5.12).

5.143 WCHAR

A concrete type, as specified in [MS-DTYP] section **2.2**. A WCHAR is a 16-bit unsigned integer, in little-endian order, used to store a double-byte Unicode character. A WCHAR * is a pointer to a null terminated Unicode string.

6 Security

The following sections specify security considerations for implementers.

6.1 Security Considerations for Implementers

General security considerations for this protocol are specified in section 2.2. Security considerations for an individual method are specified in the subsection of section 4 that describes the behavior of that method.

6.2 Index of Security Parameters

Security parameter	Section
SPNs for client-to-DC authentication	Section 2.2.3

7 Appendix A: Full IDL

For ease of implementation the full IDL is provided below, where "ms-dtyp.idl" refers to the IDL found in [MS-DTYP] Appendix A.

```
import "ms-dtyp.idl";
[
    uuid (e3514235-4b06-11d1-ab04-00c04fc2dcd2), version(4.0),
    pointer_default (unique)
]
interface drsuapi
{

typedef LONGLONG DSTIME;

typedef [context_handle] void * DRS_HANDLE;

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

typedef struct {
    unsigned long structLen;
    unsigned long SidLen;
    GUID Guid;
    NT4SID Sid;
    unsigned long NameLen;
    [size_is(NameLen + 1)] WCHAR StringName[];
} DSNAME;

typedef LONGLONG USN;

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

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

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

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

```
typedef struct {
    unsigned long ndx;
    OID_t prefix;
} PrefixTableEntry;

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

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

typedef struct {
    [range(0,10485760)] ULONG valLen;
    [size_is(valLen)] UCHAR *pVal;
} ATTRVAL;

typedef struct {
    UUID uuidDsa;
    USN usnHighPropUpdate;
    DSTIME timeLastSyncSuccess;
} UPTODATE_CURSOR_V2;

typedef struct {
    DWORD dwVersion;
    DWORD dwReserved1;
    [range(0,1048576)] DWORD cNumCursors;
    DWORD dwReserved2;
    [size_is(cNumCursors)] UPTODATE_CURSOR_V2 rgCursors[];
} UPTODATE_VECTOR_V2_EXT;

typedef struct {
    UCHAR rgTimes[84];
} REPLTIMES;

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

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

typedef struct {
    [string,unique] WCHAR *NetbiosName;
    [string,unique] WCHAR *DnsHostName;
    [string,unique] WCHAR *SiteName;
    [string,unique] WCHAR *ComputerObjectName;
    [string,unique] WCHAR *ServerObjectName;
    BOOL fIsPdc;
    BOOL fDsEnabled;
```

```
} DS_DOMAIN_CONTROLLER_INFO_1W;

typedef struct {
    [string,unique] WCHAR *NetbiosName;
    [string,unique] WCHAR *DnsHostName;
    [string,unique] WCHAR *SiteName;
    [string,unique] WCHAR *SiteObjectName;
    [string,unique] WCHAR *ComputerObjectName;
    [string,unique] WCHAR *ServerObjectName;
    [string,unique] WCHAR *NtdsDsaObjectName;
    BOOL fIsPdc;
    BOOL fDsEnabled;
    BOOL fIsGc;
    GUID SiteObjectGuid;
    GUID ComputerObjectGuid;
    GUID ServerObjectGuid;
    GUID NtdsDsaObjectGuid;
} DS_DOMAIN_CONTROLLER_INFO_2W;

typedef struct {
  [string, unique] WCHAR* NetbiosName;
  [string, unique] WCHAR* DnsHostName;
  [string, unique] WCHAR* SiteName;
  [string, unique] WCHAR* SiteObjectName;
  [string, unique] WCHAR* ComputerObjectName;
  [string, unique] WCHAR* ServerObjectName;
  [string, unique] WCHAR* NtdsDsaObjectName;
  BOOL fIsPdc;
  BOOL fDsEnabled;
  BOOL fIsGc;
  BOOL fIsRodc;
  GUID SiteObjectGuid;
  GUID ComputerObjectGuid;
  GUID ServerObjectGuid;
  GUID NtdsDsaObjectGuid;
} DS_DOMAIN_CONTROLLER_INFO_3W;

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

typedef struct {
    [string] LPWSTR pszNamingContext;
    [string] LPWSTR pszSourceDsaDN;
    [string] LPWSTR pszSourceDsaAddress;
    [string] LPWSTR pszAsyncIntersiteTransportDN;
    DWORD dwReplicaFlags;
    DWORD dwReserved;
    UUID uuidNamingContextObjGuid;
    UUID uuidSourceDsaObjGuid;
    UUID uuidSourceDsaInvocationID;
    UUID uuidAsyncIntersiteTransportObjGuid;
```

```
    USN usnLastObjChangeSynced;
    USN usnAttributeFilter;
    FILETIME ftimeLastSyncSuccess;
    FILETIME ftimeLastSyncAttempt;
    DWORD dwLastSyncResult;
    DWORD cNumConsecutiveSyncFailures;
} DS_REPL_NEIGHBORW;

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

typedef struct {
    UUID uuidSourceDsaInvocationID;
    USN usnAttributeFilter;
} DS_REPL_CURSOR;

typedef struct {
    DWORD       cNumCursors;
    DWORD       dwReserved;
    [size_is(cNumCursors)] DS_REPL_CURSOR rgCursor[];
} DS_REPL_CURSORS;

typedef struct {
    [string] LPWSTR pszAttributeName;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
} DS_REPL_ATTR_META_DATA;

typedef struct {
    [string] LPWSTR pszDsaDN;
    UUID uuidDsaObjGuid;
    FILETIME ftimeFirstFailure;
    DWORD cNumFailures;
    DWORD dwLastResult;
} DS_REPL_KCC_DSA_FAILUREW;

typedef struct {
    DWORD cNumEntries;
    DWORD dwReserved;
    [size_is(cNumEntries)] DS_REPL_KCC_DSA_FAILUREW rgDsaFailure[];
} DS_REPL_KCC_DSA_FAILURESW;

typedef struct {
    DWORD cNumEntries;
    DWORD dwReserved;
    [size_is(cNumEntries)] DS_REPL_ATTR_META_DATA rgMetaData[];
} DS_REPL_OBJ_META_DATA;

typedef enum {
    DS_REPL_OP_TYPE_SYNC = 0,
    DS_REPL_OP_TYPE_ADD,
    DS_REPL_OP_TYPE_DELETE,
```

```
    DS_REPL_OP_TYPE_MODIFY,
    DS_REPL_OP_TYPE_UPDATE_REFS
} DS_REPL_OP_TYPE;

typedef struct {
    FILETIME ftimeEnqueued;
    ULONG ulSerialNumber;
    ULONG ulPriority;
    DS_REPL_OP_TYPE OpType;
    ULONG ulOptions;
    [string] LPWSTR pszNamingContext;
    [string] LPWSTR pszDsaDN;
    [string] LPWSTR pszDsaAddress;
    UUID uuidNamingContextObjGuid;
    UUID uuidDsaObjGuid;
} DS_REPL_OPW;

typedef struct {
    FILETIME ftimeCurrentOpStarted;
    DWORD cNumPendingOps;
    [size_is(cNumPendingOps)] DS_REPL_OPW rgPendingOp[];
} DS_REPL_PENDING_OPSW;

typedef struct {
    [string] LPWSTR pszAttributeName;
    [string] LPWSTR pszObjectDn;
    DWORD cbData;
    [size_is(cbData), ptr] BYTE *pbData;
    FILETIME ftimeDeleted;
    FILETIME ftimeCreated;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
} DS_REPL_VALUE_META_DATA;

typedef struct {
    DWORD cNumEntries;
    DWORD dwEnumerationContext;
    [size_is(cNumEntries)] DS_REPL_VALUE_META_DATA rgMetaData[];
} DS_REPL_ATTR_VALUE_META_DATA;

typedef struct {
    UUID uuidSourceDsaInvocationID;
    USN usnAttributeFilter;
    FILETIME ftimeLastSyncSuccess;
} DS_REPL_CURSOR_2;

typedef struct {
    DWORD cNumCursors;
    DWORD dwEnumerationContext;
    [size_is(cNumCursors)] DS_REPL_CURSOR_2 rgCursor[];
} DS_REPL_CURSORS_2;

typedef struct {
    UUID uuidSourceDsaInvocationID;
    USN usnAttributeFilter;
```

```
    FILETIME ftimeLastSyncSuccess;
    [string] LPWSTR pszSourceDsaDN;
} DS_REPL_CURSOR_3W;

typedef struct {
    DWORD cNumCursors;
    DWORD dwEnumerationContext;
    [size_is(cNumCursors)] DS_REPL_CURSOR_3W rgCursor[];
} DS_REPL_CURSORS_3W;

typedef struct {
    [string] LPWSTR pszAttributeName;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
    [string] LPWSTR pszLastOriginatingDsaDN;
} DS_REPL_ATTR_META_DATA_2;

typedef struct {
    DWORD cNumEntries;
    DWORD dwReserved;
    [size_is(cNumEntries)] DS_REPL_ATTR_META_DATA_2 rgMetaData[];
} DS_REPL_OBJ_META_DATA_2;

typedef struct {
    [string] LPWSTR pszAttributeName;
    [string] LPWSTR pszObjectDn;
    DWORD cbData;
    [size_is(cbData), ptr] BYTE *pbData;
    FILETIME ftimeDeleted;
    FILETIME ftimeCreated;
    DWORD dwVersion;
    FILETIME ftimeLastOriginatingChange;
    UUID uuidLastOriginatingDsaInvocationID;
    USN usnOriginatingChange;
    USN usnLocalChange;
    [string] LPWSTR pszLastOriginatingDsaDN;
} DS_REPL_VALUE_META_DATA_2;

typedef struct {
    DWORD cNumEntries;
    DWORD dwEnumerationContext;
    [size_is(cNumEntries)] DS_REPL_VALUE_META_DATA_2 rgMetaData[];
} DS_REPL_ATTR_VALUE_META_DATA_2;

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

typedef struct {
    [ref] DSNAME *pNC;
    UUID uuidDsaSrc;
    [unique] [string] char *pszDsaSrc;
    ULONG ulOptions;
} DRS_MSG_REPSYNC_V1;
```

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

typedef struct {
    [ref] DSNAME *pNC;
    [ref] [string] char *pszDsaDest;
    UUID uuidDsaObjDest;
    ULONG ulOptions;
} DRS_MSG_UPDREFS_V1;

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

typedef struct {
    [ref] DSNAME *pNC;
    [ref] [string] char *pszDsaSrc;
    REPLTIMES rtSchedule;
    ULONG ulOptions;
} DRS_MSG_REPADD_V1;

typedef struct {
    [ref] DSNAME *pNC;
    [unique] DSNAME *pSourceDsaDN;
    [unique] DSNAME *pTransportDN;
    [ref] [string] char *pszSourceDsaAddress;
    REPLTIMES rtSchedule;
    ULONG ulOptions;
} DRS_MSG_REPADD_V2;

typedef  [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_REPADD_V1 V1;
    [case(2)] DRS_MSG_REPADD_V2 V2;
} DRS_MSG_REPADD;

typedef struct {
    [ref] DSNAME *pNC;
    [string] char *pszDsaSrc;
    ULONG ulOptions;
} DRS_MSG_REPDEL_V1;

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

typedef struct {
    [ref] DSNAME *pNC;
    UUID uuidSourceDRA;
    [unique, string] char *pszSourceDRA;
    REPLTIMES rtSchedule;
    ULONG ulReplicaFlags;
    ULONG ulModifyFields;
    ULONG ulOptions;
} DRS_MSG_REPMOD_V1;
```

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

typedef struct {
    ULONG CodePage;
    ULONG LocaleId;
    DWORD dwFlags;
    DWORD formatOffered;
    DWORD formatDesired;
    [range(1,10000)] DWORD cNames;
    [string, size_is(cNames)] WCHAR **rpNames;
} DRS_MSG_CRACKREQ_V1;

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

typedef struct {
    DS_NAME_RESULTW *pResult;
} DRS_MSG_CRACKREPLY_V1;

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

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

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

typedef struct {
    DWORD retVal;
} DRS_MSG_SPNREPLY_V1;

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

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

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

typedef struct {
    BOOL fLastDcInDomain;
```

```
} DRS_MSG_RMSVRREPLY_V1;

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

typedef struct {
    [string] LPWSTR DomainDN;
} DRS_MSG_RMDMNREQ_V1;

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

typedef struct {
    DWORD Reserved;
} DRS_MSG_RMDMNREPLY_V1;

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

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

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

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

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

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

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

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_DCINFOREPLY_V1 V1;
    [case(2)] DRS_MSG_DCINFOREPLY_V2 V2;
    [case(3)] DRS_MSG_DCINFOREPLY_V3 V3;
    [case(0xFFFFFFFF)] DRS_MSG_DCINFOREPLY_VFFFFFFFF VFFFFFFFF;
} DRS_MSG_DCINFOREPLY;
```

```
typedef struct {
    DWORD dwTaskID;
    DWORD dwFlags;
} DRS_MSG_KCC_EXECUTE_V1;

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

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

typedef struct {
    [range(0,10000)] DWORD cNumContexts;
    DWORD dwReserved;
    [size_is(cNumContexts)] DS_REPL_CLIENT_CONTEXT rgContext[];
} DS_REPL_CLIENT_CONTEXTS;

typedef struct {
    [string] LPWSTR pszServerName;
    BOOL fIsHandleBound;
    BOOL fIsHandleFromCache;
    BOOL fIsHandleInCache;
    DWORD dwThreadId;
    DWORD dwBindingTimeoutMins;
    DSTIME dstimeCreated;
    DWORD dwCallType;
} DS_REPL_SERVER_OUTGOING_CALL;

typedef struct {
    [range(0, 256)] DWORD cNumCalls;
    DWORD dwReserved;
    [size_is(cNumCalls)] DS_REPL_SERVER_OUTGOING_CALL rgCall[];
} DS_REPL_SERVER_OUTGOING_CALLS;

typedef struct {
    DWORD InfoType;
    [string] LPWSTR pszObjectDN;
    UUID uuidSourceDsaObjGuid;
} DRS_MSG_GETREPLINFO_REQ_V1;

typedef struct {
    DWORD InfoType;
    [string] LPWSTR pszObjectDN;
    UUID uuidSourceDsaObjGuid;
    DWORD ulFlags;
    [string] LPWSTR pszAttributeName;
    [string] LPWSTR pszValueDN;
    DWORD dwEnumerationContext;
} DRS_MSG_GETREPLINFO_REQ_V2;
```

```
typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_GETREPLINFO_REQ_V1 V1;
    [case(2)] DRS_MSG_GETREPLINFO_REQ_V2 V2;
} DRS_MSG_GETREPLINFO_REQ;

typedef [switch_type(DWORD)] union {
    [case(0)] DS_REPL_NEIGHBORSW *pNeighbors;
    [case(1)] DS_REPL_CURSORS *pCursors;
    [case(2)] DS_REPL_OBJ_META_DATA *pObjMetaData;
    [case(3)] DS_REPL_KCC_DSA_FAILURESW *pConnectFailures;
    [case(4)] DS_REPL_KCC_DSA_FAILURESW *pLinkFailures;
    [case(5)] DS_REPL_PENDING_OPSW *pPendingOps;
    [case(6)] DS_REPL_ATTR_VALUE_META_DATA *pAttrValueMetaData;
    [case(7)] DS_REPL_CURSORS_2 *pCursors2;
    [case(8)] DS_REPL_CURSORS_3W *pCursors3;
    [case(9)] DS_REPL_OBJ_META_DATA_2 *pObjMetaData2;
    [case(10)] DS_REPL_ATTR_VALUE_META_DATA_2 *pAttrValueMetaData2;
    [case(0xFFFFFFFA)]
        DS_REPL_SERVER_OUTGOING_CALLS *pServerOutgoingCalls;
    [case(0xFFFFFFFB)] UPTODATE_VECTOR_V1_EXT *pUpToDateVec;
    [case(0xFFFFFFFC)] DS_REPL_CLIENT_CONTEXTS *pClientContexts;
    [case(0xFFFFFFFE)] DS_REPL_NEIGHBORSW *pRepsTo;
} DRS_MSG_GETREPLINFO_REPLY;

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

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

typedef struct {
    DWORD dwWin32Error;
} DRS_MSG_ADDSIDREPLY_V1;

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

typedef struct {
    [ref] DSNAME *pNC;
    UUID uuidDsaSrc;
    ULONG ulOptions;
} DRS_MSG_REPVERIFYOBJ_V1;

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

```
    [case(1)] DRS_MSG_REPVERIFYOBJ_V1 V1;
} DRS_MSG_REPVERIFYOBJ;

typedef struct {
    [string] const WCHAR *pwszFromSite;
    [range(1,10000)] DWORD cToSites;
    [string, size_is(cToSites)] WCHAR **rgszToSites;
    DWORD dwFlags;
} DRS_MSG_QUERYSITESREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_QUERYSITESREQ_V1 V1;
} DRS_MSG_QUERYSITESREQ;

typedef struct {
    DWORD dwErrorCode;
    DWORD dwCost;
} DRS_MSG_QUERYSITESREPLYELEMENT_V1;

typedef struct {
    [range(0,10000)] DWORD cToSites;
    [size_is(cToSites)] DRS_MSG_QUERYSITESREPLYELEMENT_V1 *rgCostInfo;
    DWORD dwFlags;
} DRS_MSG_QUERYSITESREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_QUERYSITESREPLY_V1 V1;
} DRS_MSG_QUERYSITESREPLY;

typedef struct {
    DWORD dwReserved;
} DRS_MSG_INIT_DEMOTIONREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_INIT_DEMOTIONREQ_V1 V1;
} DRS_MSG_INIT_DEMOTIONREQ;

typedef struct {
    DWORD dwOpError;
} DRS_MSG_INIT_DEMOTIONREPLY_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_INIT_DEMOTIONREPLY_V1 V1;
} DRS_MSG_INIT_DEMOTIONREPLY;

typedef struct {
    DWORD dwFlags;
    UUID uuidHelperDest;
    DSNAME* pNC;
} DRS_MSG_REPLICA_DEMOTIONREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_REPLICA_DEMOTIONREQ_V1 V1;
} DRS_MSG_REPLICA_DEMOTIONREQ;

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

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

typedef struct {
    DWORD dwOperations;
    UUID uuidHelperDest;
    LPWSTR szScriptBase;
} DRS_MSG_FINISH_DEMOTIONREQ_V1;

typedef [switch_type(DWORD)] union {
    [case(1)] DRS_MSG_FINISH_DEMOTIONREQ_V1 V1;
} DRS_MSG_FINISH_DEMOTIONREQ;

typedef struct {
    DWORD dwOperationsDone;
    DWORD dwOpFailed;
    DWORD dwOpError;
} DRS_MSG_FINISH_DEMOTIONREPLY_V1;

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


// opnum 0
ULONG
IDL_DRSBind(
    [in] handle_t rpc_handle,
    [in, unique] UUID *puuidClientDsa,
    [in, unique] DRS_EXTENSIONS *pextClient,
    [out] DRS_EXTENSIONS **ppextServer,
    [out, ref] DRS_HANDLE *phDrs);

// opnum 1
ULONG
IDL_DRSUnbind(
    [in, out, ref] DRS_HANDLE *phDrs);

// opnum 2
ULONG
IDL_DRSReplicaSync(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPSYNC *pmsgSync);

// Opnum 3
void Opnum3ServerToServerOnly(void);

// opnum 4
ULONG
IDL_DRSUpdateRefs(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_UPDREFS *pmsgUpdRefs);

// opnum 5
```

```
ULONG
IDL_DRSReplicaAdd(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPADD *pmsgAdd);

// opnum 6
ULONG
IDL_DRSReplicaDel(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPDEL *pmsgDel);

// opnum 7
ULONG
IDL_DRSReplicaModify(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPMOD *pmsgMod);

// Opnum 8
void Opnum8ServerToServerOnly(void);

// Opnum 9
void Opnum9ServerToServerOnly(void);

// Opnum 10
void Opnum10ServerToServerOnly(void);

// Opnum 11
void Opnum11ServerToServerOnly(void);

// opnum 12
ULONG
IDL_DRSCrackNames(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_CRACKREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_CRACKREPLY *pmsgOut);

// opnum 13
ULONG
IDL_DRSWriteSPN(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_SPNREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)] DRS_MSG_SPNREPLY *pmsgOut);

// opnum 14
ULONG
IDL_DRSRemoveDsServer(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_RMSVRREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
```

```
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_RMSVRREPLY *pmsgOut);

// opnum 15
ULONG
IDL_DRSRemoveDsDomain(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_RMDMNREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_RMDMNREPLY *pmsgOut);

// opnum 16
ULONG
IDL_DRSDomainControllerInfo(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_DCINFOREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_DCINFOREPLY *pmsgOut);

// Opnum 17
void Opnum17ServerToServerOnly(void);

// opnum 18
ULONG
IDL_DRSExecuteKCC(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_KCC_EXECUTE *pmsgIn);

// opnum 19
ULONG
IDL_DRSGetReplInfo(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_GETREPLINFO_REQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_GETREPLINFO_REPLY *pmsgOut);

// opnum 20
ULONG
IDL_DRSAddSidHistory(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_ADDSIDREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_ADDSIDREPLY *pmsgOut);

// Opnum 21
void Opnum21ServerToServerOnly(void);

// opnum 22
ULONG
```

```
IDL_DRSReplicaVerifyObjects(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwVersion,
    [in, ref, switch_is(dwVersion)] DRS_MSG_REPVERIFYOBJ *pmsgVerify);

// Opnum 23
void Opnum23ServerToServerOnly(void);

// opnum 24
ULONG
IDL_DRSQuerySitesByCost(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)] DRS_MSG_QUERYSITESREQ *pmsgIn,
    [out, ref] DWORD *pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_QUERYSITESREPLY *pmsgOut);

// opnum 25
ULONG
IDL_DRSInitDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_INIT_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_INIT_DEMOTIONREPLY* pmsgOut);

// opnum 26
ULONG
IDL_DRSReplicaDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_REPLICA_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_REPLICA_DEMOTIONREPLY* pmsgOut);

// opnum 27
ULONG
IDL_DRSFinishDemotion(
    [in, ref] DRS_HANDLE hDrs,
    [in] DWORD dwInVersion,
    [in, ref, switch_is(dwInVersion)]
        DRS_MSG_FINISH_DEMOTIONREQ* pmsgIn,
    [out, ref] DWORD* pdwOutVersion,
    [out, ref, switch_is(*pdwOutVersion)]
        DRS_MSG_FINISH_DEMOTIONREPLY* pmsgOut);

}
```

8 Appendix B: Windows Behavior

All IDL methods and their associated concrete types have existed in the drsuapi RPC interface since Windows 2000 except those listed in the following table.

Data type or IDL method	Section	Windows version introduced
DRS_MSG_GETREPLINFO_REQ_V2	4.1.7.1.3	Windows Server 2003
IDL_DRSReplicaVerifyObjects	4.1.17	Windows Server 2003
IDL_DRSFinishDemotion	4.1.6	Windows Server 2008
IDL_DRSInitDemotion	4.1.8	Windows Server 2008
IDL_DRSReplicaDemotion	4.1.14	Windows Server 2008

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2008
- Windows Vista
- Windows XP
- Windows Server 2003
- Windows 2000

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.1: Windows servers listen only on the RPC over TCP protocol sequence. Windows clients attempt to connect only via the RPC over TCP protocol sequence.

<2> Section 2.2.4: Windows implements DC-to-DC interaction with an SPN with service class "E3514235-4B06-11D1-AB04-00C04FC2DCD2". See DRS_SPN_CLASS.

<3> Section 3.6: The client reuses an association for multiple invocations.

<4> Section 4.1: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
3	Server-to-Server implementation only.
8	Server-to-Server implementation only.
9	Server-to-Server implementation only.
10	Server-to-Server implementation only.
11	Server-to-Server implementation only.
17	Server-to-Server implementation only.

Opnum	Description
21	Server-to-Server implementation only.
23	Server-to-Server implementation only.

<5> Section 4.1.1.2.6: The function determines if auditing is enabled on the server by querying the LSA information policy on the server associated with ctx and confirming that the information policy is set to generate both success and failure audits for the "account management" audit category. To achieve this, the LsarOpenPolicy2, LsarQueryInformationPolicy, and LsarClose messages in [MS-LSAD] are used ([MS-LSAD] sections **3.1.4.4.1**, **3.1.4.4.4**, and **3.1.4.9.5**). The *SystemName* parameter to LsarOpenPolicy2 is generated using the **srcDomainController** variable in the **IDL_DRSAddSidHistory** method and the *DesiredAccess* parameter to LsarOpenPolicy2 is set to (POLICY_VIEW_AUDIT_INFORMATION | POLICY_VIEW_LOCAL_INFORMATION). On success, the *PolicyHandle* acquired from the LsarOpenPolicy2 message is passed to LsarQueryInformationPolicy with PolicyAuditEventsInformation as the information class. The check to determine if success and failure audits are enabled for "account management" is achieved by performing the following evaluation:

```
PolicyInformation^.PolicyAuditEventsInfo.EventAuditingOptions[6] ∩
  { POLICY_AUDIT_EVENT_SUCCESS, POLICY_AUDIT_EVENT_FAILURE } =
{ POLICY_AUDIT_EVENT_SUCCESS, POLICY_AUDIT_EVENT_FAILURE }
```

where *PolicyInformation* is the result from the LsarQueryInformationPolicy message. *PolicyHandle* is then closed using the LsarClose message.

The function generates an audit on the DC associated with ctx by adding the source principal (pmsgIn^.V1.SrcPrincipal, where pmsgIn is a parameter to the **IDL_DRSAddSidHistory** function which in turn calls this method) to the group *srcDomainFlatName*$$$ on the DC associated with ctx, where *srcDomainFlatName* is the NetBIOS name of the domain to which the source principal belongs. After adding the principal to the group, it then removes the principal from the group, leaving the group in its original state but having generated an audit event as a side effect of manipulating the group's membership.

<6> Section 4.1.1.2.12: This test is implemented in two steps. First, it is determined if the DC associated with ctx is running at least Windows 2000. This is determined by whether a SamrConnect5 or SamrConnect4 API call (as specified in [MS-SAMR]) to the DC is successful. If it is, the DC is running at least Windows 2000, and the function returns true.

Otherwise, the DC is considered to be running Windows NT 4.0. The function then connects to the registry service on the DC named in ctx and queries the value of the "CSDVersion" registry value on the "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion" registry key. If the value is not equal to any of the following strings, the function returns true, otherwise, it returns false:

- Service Pack 0
- Service Pack 1
- Service Pack 2
- Service Pack 3

If the registry service on the DC could not be contacted, or if the registry key or registry value do not exist, the function returns false.

<7> Section 4.1.2.1: Windows non-DC client callers always use the following UUID for puuidClientDsa: e24d201a-4fd6-11d1-a3da-0000f875ae0d.

<8> Section 4.1.2.1: Windows non-DC client callers always set the **dwFlags** field of the DRS_EXTENSIONS_INT structure to zero. Windows non-DC client callers always set the **SiteObjGuid** field of the DRS_EXTENSIONS_INT structure to the NULL GUID value. Windows non-DC client callers always set the Pid field of the DRS_EXTENSIONS_INT structure to an implementation-specific, client-local process identifier.

<9> Section 4.1.5.3: The Knowledge Consistency Checker (KCC) is an internal component that runs on all DCs. The KCC generates and maintains the replication topology between DCs within sites and between sites. The KCC performs two major tasks:

- Configure replication connections (nTDSConnection objects) between DCs. Each nTDSConnection object defines a replication link between the **local DC** and a replication partner DC. The local DC sends server-to-server replication requests to the replication partner DC.
- Converts the connection objects that represent replication to the local DC into an repsFrom values used by DC replication.

The KCCs implementation requirements are outlined in [MS-ADTS] section 7.2. Upon receiving this message, the server will trigger the KCC to perform the above two tasks.

<10> Section 5.27: Clients callers set dwFlags to zero.

<11> Section 5.27: This field contains the process ID of the client.

9 Index

E

F

G

H

I

K

L

M

N

V

W