[MS-ADTS]:
Active Directory Technical Specification

Intellectual Property Rights Notice for Open Specifications Documentation

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

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
02/22/2007	0.01		MCPP Milestone 3 Initial Availability
06/01/2007	1.0	Major	Included non-native content.
07/03/2007	1.0.1	Editorial	Revised and edited the technical content.
07/20/2007	1.0.2	Editorial	Revised and edited the technical content.
08/10/2007	1.0.3	Editorial	Revised and edited the technical content.
09/28/2007	2.0	Major	Adjusted bitfield diagrams for byte ordering; added bitflags.
10/23/2007	2.1	Minor	Updated the technical content.
11/30/2007	2.2	Minor	Updated the technical content.
01/25/2008	3.0	Major	Updated and revised the technical content.
03/14/2008	3.1	Minor	Deleted hexadecimal representations of little-endian bit flags.
05/16/2008	4.0	Major	Updated and revised the technical content.
06/20/2008	5.0	Major	Updated and revised the technical content.
07/25/2008	6.0	Major	Updated and revised the technical content.
08/29/2008	7.0	Major	Updated and revised the technical content.
10/24/2008	8.0	Major	Updated and revised the technical content.
12/05/2008	9.0	Major	Updated and revised the technical content.
01/16/2009	10.0	Major	Updated and revised the technical content.
02/27/2009	11.0	Major	Updated and revised the technical content.
04/10/2009	12.0	Major	Updated and revised the technical content.
05/22/2009	13.0	Major	Updated and revised the technical content.
07/02/2009	14.0	Major	Updated and revised the technical content.
08/14/2009	15.0	Major	Updated and revised the technical content.
09/25/2009	16.0	Major	Updated and revised the technical content.
11/06/2009	17.0	Major	Updated and revised the technical content.
12/18/2009	18.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
01/29/2010	19.0	Major	Updated and revised the technical content.
03/12/2010	20.0	Major	Updated and revised the technical content.
04/23/2010	21.0	Major	Updated and revised the technical content.
06/04/2010	22.0	Major	Updated and revised the technical content.
07/16/2010	23.0	Major	Significantly changed the technical content.
08/27/2010	24.0	Major	Significantly changed the technical content.
10/08/2010	25.0	Major	Significantly changed the technical content.
11/19/2010	26.0	Major	Significantly changed the technical content.
01/07/2011	27.0	Major	Significantly changed the technical content.
02/11/2011	28.0	Major	Significantly changed the technical content.
03/25/2011	29.0	Major	Significantly changed the technical content.
05/06/2011	30.0	Major	Significantly changed the technical content.
06/17/2011	30.1	Minor	Clarified the meaning of the technical content.
09/23/2011	31.0	Major	Significantly changed the technical content.
12/16/2011	32.0	Major	Significantly changed the technical content.
03/30/2012	33.0	Major	Significantly changed the technical content.
07/12/2012	34.0	Major	Significantly changed the technical content.
10/25/2012	35.0	Major	Significantly changed the technical content.
01/31/2013	36.0	Major	Significantly changed the technical content.
08/08/2013	37.0	Major	Significantly changed the technical content.

Contents

1 Introduction

This is the primary specification for **Active Directory**, both **Active Directory Domain Services (AD DS)** and **Active Directory Lightweight Directory Services (AD LDS)**. The state model for this specification is prerequisite to the other specifications for Active Directory: [MS-DRSR] and [MS-SRPL].

When no operating system version information is specified, information in this document applies to all relevant versions of Windows. Similarly, when no **DC functional level** is specified, information in this document applies to all DC functional levels.

Unless otherwise specified, information in this specification is also applicable to Active Directory Application Mode (ADAM). ADAM is a standalone application that provides AD LDS capabilities on Windows XP operating system and Windows Server 2003 operating system. There are two versions of ADAM, ADAM RTW and ADAM SP1; unless otherwise specified, where ADAM is discussed in this document it refers to both versions.

Information that is applicable to AD LDS on Windows Server 2008 operating system is also applicable to Active Directory Lightweight Directory Services (AD LDS) for Windows Vista, except where it is explicitly specified that such information is not applicable to that product. AD LDS for Windows Vista is a standalone application that provides AD LDS capabilities for Windows Vista operating system. Similarly, unless it is explicitly specified otherwise, information that is applicable to AD LDS on Windows Server 2008 R2 operating system is also applicable to the standalone application Active Directory Lightweight Directory Services (AD LDS) for Windows 7, which provides AD LDS capabilities for Windows 7 operating system. Similarly, unless it is explicitly specified otherwise, information that is applicable to AD LDS on Windows Server 2012 operating system is also applicable to the stand-alone application Active Directory Lightweight Directory Services (AD LDS) for Windows 8 operating system, which provides AD LDS capabilities for Windows 8 operating system. Finally, unless it is explicitly specified otherwise, information that is applicable to AD LDS on Windows Server 2012 R2 operating system is also applicable to the stand-alone application Active Directory Lightweight Directory Services (AD LDS) for Windows 8.1 operating system, which provides AD LDS capabilities for Windows 8.1 operating system.

State is included in the state model for this specification only as necessitated by the requirement that a licensee implementation of Windows Server protocols be able to receive messages and respond 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 in the sections that follow, all of the behaviors are required for interoperability.

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.

For clarity, bit flags are sometimes shown as bit field diagrams. In the case of bit flags for **Lightweight Directory Access Protocol (LDAP)** attributes, these diagrams take on **big-endian** characteristics but do not reflect the actual byte ordering of integers over the wire, because LDAP transfers an integer as the **UTF-8** string of the decimal representation of that integer, as specified in [RFC2252].

Pervasive Concepts

The following concepts are pervasive throughout this specification.

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

replica: A variable containing a set of **objects**.

attribute: An identifier for a value or set of values. See also attribute in the Glossary section.

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

- Identifying attribute. A designated single-valued attribute appears on every object; the value of this attribute identifies the object. For the set of objects in a **replica**, the values of the identifying attribute are distinct.
- Parent-identifying attribute. A designated single-valued attribute appears on every object; the value of this attribute identifies the object's parent. That is, this attribute contains the value of the parent's identifying attribute, or a reserved value identifying no object (for more information, see section 3.1.1.1.3). For the set of objects in a replica, the values of this parent-identifying attribute define an **oriented tree** with objects as vertices and child-parent references as directed arcs, with the child as an arc's initial vertex and the parent as an arc's final vertex.

Note that 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 word replica suggests, it is often the case that two replicas 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 (**replication**) to converge the values of the remaining attributes. When the members of a set of replicas are considered to be the same, it is common to say "an object" as a shorthand way of 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 creating an object. An object class specifies a set of must-have attributes (every object of the class must have at least one value of each) and may-have attributes (every object of the class may have a value of each). An object class also specifies a set of possible superiors (the **parent object** of an object of the class must have one of these classes). An object class is defined by a classSchema object.

parent object: See "object", above.

child object, children: An object that is not the root of its oriented tree. The **children** of an object o is the set of all objects whose parent is o.

See section 3.1.1.1.3 for the particular use made of these definitions in this specification.

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

1.1 Glossary

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

88 object class
abstract class
abstract object class
access check
access control entry (ACE)
access control list (ACL)
access mask
account domain
ACID
ambiguous name resolution (ANR)
ancestor object
attribute syntax
AttributeStamp
authentication
authorization
auxiliary object class
back link attribute
back link value
backup domain controller (BDC)
big-endian
binary large object (BLOB)
bridgehead domain controller (bridgehead DC)
broadcast
canonical name
checksum
claim
code page
Component Object Model (COM)
constructed attribute
container
control access right
Coordinated Universal Time (UTC)
critical object
cyclic redundancy check (CRC)
digest
directory service (DS)
discretionary access control list (DACL)
distinguished name (DN)(4)
domain
domain name (3)
Domain Name System (DNS)
downlevel trust
endpoint
expunge

forward link value
FSMO role owner
full NC replica
fully qualified domain name (FQDN) (1) (2)
garbage collection
global catalog (GC)
global catalog server (GC server)
globally unique identifier (GUID)
group
Group Policy
GUIDString
inheritance
Lightweight Directory Access Protocol (LDAP)
LDAP connection
link attribute
link value
LinkValueStamp
local domain controller (local DC)
Lost and Found container
marshal
Messaging Application Programming Interface (MAPI)
mixed mode
multi-valued claim
name service provider interface (NSPI)
native mode
nonreplicated attribute
NULL GUID
object of class x (or x object)
operational attribute
originating update
partial attribute set (PAS)
privilege (1)
property set
RDN attribute
remote procedure call (RPC)
replicated update
replication
replication latency
replication traffic
RPC transport
schema
schema container
schema object
security identifier (SID)
security principal
security provider
service principal name (SPN)
single-valued claim
Simple Mail Transfer Protocol (SMTP)
SSL/TLS handshake
structural object class
system access control list (SACL)
ticket-granting ticket (TGT)
trust object
trust secret

trusted domain object (TDO)
Unicode
universally unique identifier (UUID)
uplevel trust
Windows error code

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

organization

The following terms are specific to this document:

active: A state of an attributeSchema or classSchema **object** that represents part of the **schema**. It is possible to instantiate an **active attribute** or an **active** class. The opposite term is **defunct**.

Active Directory: Either **Active Directory Domain Services (AD DS)** or **Active Directory Lightweight Directory Services (AD LDS)**. **Active Directory** is either deployed as **AD DS** or as **AD LDS**. This document describes both forms. When the specification does not refer specifically to **AD DS** or **AD LDS**, it applies to both.

Active Directory Domain Services (AD DS): AD DS is an operating system **directory service (DS)** implemented by a **domain controller (DC)**. The **DS** 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 Microsoft Windows 2000 and is available as part of Windows 2000 Server products and Windows Server 2003 products; in these products it is called "Active Directory". It is also available as part of Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2. **AD DS** is not present in Windows NT 4.0 or in Windows XP. For more information, see [MS-AUTHSOD] section 1.1.1.5.2.

Active Directory Lightweight Directory Services (AD LDS): AD LDS is a **directory service (DS)** implemented by a **domain controller (DC)**. The most significant difference between **AD LDS** and **Active Directory Domain Services (AD DS)** is that **AD LDS** does not host **domain naming contexts (domain NCs)**. A server can host multiple **AD LDS DCs**. Each **DC** is an independent **AD LDS** instance, with its own independent state. **AD LDS** can be run as an operating system **DS** or as a **directory** service provided by a standalone application (ADAM).

application NC: A specific type of **naming context (NC)**. An **application NC** cannot contain **security principal objects** in **Active Directory domain services (AD DS)** but can contain **security principals** in **Active Directory Lightweight Directory Service (AD LDS)**. In **AD DS** or **AD LDS**, a **forest** can have zero or more **application NCs**.

attribute: (Note: This definition is a specialization of the "**attribute**" concept that is described in section 1, Introduction, under Pervasive Concepts.) An identifier for a single-valued or multi-valued data element that is associated with an **object**. An **object** consists of its **attributes** and their values. For example, cn (common name), street (street address), and mail (email 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**.

ATTRTYP: A 32-bit quantity representing an **object identifier (OID)**. See [MS-DRSR] section 5.14.

auxiliary class: See **auxiliary object class**.

Basic Encoding Rules (BER): A specific set of rules for encoding data structures for transfer over a network. These encoding rules are defined in [ITUX690].

built-in domain: The **security identifier (SID)** namespace defined by the fixed **SID** S-1-5-32. Contains **groups** that define roles on a local machine such as "Backup Operators".

built-in domain SID: The fixed **SID** S-1-5-32.

child naming context (child NC): Given **naming contexts (NCs)** with their corresponding **distinguished names (DNs)** forming a child and parent relationship, the **NC** in the child relationship is referred as the **child NC**. The parent of a **child NC** must be an **NC** and is referred to as the **parent naming context (parent NC)**.

child object, children: See section 1, Introduction, under Pervasive Concepts.

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**.

configuration naming context (config NC): A specific type of **NC** or an instance of that type. A **forest** has a single **config NC**, which contains configuration information that is shared among all **DC** in the **forest**. A **config NC** cannot contain **security principal objects**.

crossRef object: An **object of class** crossRef. Each **crossRef object** is a child of the **Partitions container** in the **configuration naming context (Config NC)**. The class crossRef specifies the properties of a **naming context (NC)**, such as its **DNS name**, operational settings, and so on.

cross-forest trust: A relationship between two **forests** that enables **security principals** from any **domain** in one **forest** to authenticate to computers joined to any **domain** in the other **forest**.

cycle: See **replication cycle**.

DC functional level: A specification of functionality available in a **domain controller (DC)**. For **AD DS**, possible values are DS_BEHAVIOR_WIN2000 (for Windows 2000 Server **DCs**), DS_BEHAVIOR_WIN2003 (for Windows Server 2003 **DCs**), DS_BEHAVIOR_WIN2008 (for Windows Server 2008 **DCs**), DS_BEHAVIOR_WIN2008R2 (for Windows Server 2008 R2 **DCs**), DS_BEHAVIOR_WIN2012 (for Windows Server 2012 **DCs**), and DS_BEHAVIOR_WIN2012R2 (for Windows Server 2012 R2 **DCs**). For **AD LDS**, possible values are DS_BEHAVIOR_WIN2003 (for Windows Server 2003 **DCs**), DS_BEHAVIOR_WIN2008 (for Windows Server 2008 **DCs**), DS_BEHAVIOR_WIN2008R2 (for Windows Server 2008 R2 **DCs**), DS_BEHAVIOR_WIN2012 (for Windows Server 2012 **DCs**), and DS_BEHAVIOR_WIN2012R2 (for Windows Server 2012 R2 **DCs**).

default domain naming context (default domain NC): When **Active Directory** is operating as **Active Directory Domain Services (AD DS)**, this is the **default naming context (default NC)** of the **domain controller (DC)**. When operating as **Active Directory Lightweight Directory Services (AD LDS)**, this **NC** is not defined.

default naming context (default NC): When **Active Directory** is operating as **Active Directory Domain Services (AD DS)**, the **default naming context (default NC)** is the **domain naming context (domain NC)** whose full **replica** is hosted by a **domain controller (DC)**, except when the **DC** is a **read-only domain controller (RODC)**, in which case the **default NC** is a **filtered partial NC replica**. When operating as **AD DS**, the **default NC** contains the **DC's computer object**. When **Active Directory** is operating as **AD LDS**, the **default NC** is the **naming context (NC)** specified by the msDS-DefaultNamingContext **attribute** on the nTDSDSA **object** for the **DC**. See **nTDSDSA object**.

default schema: The **schema** of a given version of **Active Directory**, as defined by [MS-ADSC], [MS-ADA1], [MS-ADA2], and [MS-ADA3] for **AD DS**, and as defined by [MS-ADLS] for **Active Directory Lightweight Directory Services (AD LDS)**.

defunct: A state of an attributeSchema or classSchema **object** that represents part of the **schema**. It is not possible to instantiate a **defunct attribute** or a **defunct class**. The opposite term is **active**.

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

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

directory: A **forest**.

directory object (or object): A **Lightweight Directory Access Protocol (LDAP) object** [RFC2251], which is a specialization of the "object" concept that is described in section 1, Introduction, under Pervasive Concepts. An **Active Directory object** can be identified by a **dsname** according to the matching rules defined in [MS-DRSR] section 5.50, DSNAME.

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

DNS name: A **fully qualified domain name (FQDN) (1)**.

domain controller (DC): The service, running on a server, that implements **Active Directory**, or the server hosting this service. The service hosts the data store for **objects** and interoperates with other **DCs** to ensure that a local change to an **object** replicates correctly across all **DCs**. When **Active Directory** is operating as **Active Directory Domain Services (AD DS)**, the **DC** contains **full NC replicas** of the **configuration naming context (config NC)**, **schema naming context (schema NC)**, and one of the **domain NCs** in its **forest**. If the **AD DS DC** is a **global catalog server (GC server)**, it contains **partial NC replicas** of the remaining **domain NCs** in its **forest**. For more information, see [MS-AUTHSOD] section 1.1.1.5.2. When **Active Directory** is operating as **Active Directory Lightweight Directory Services (AD LDS)**, several **AD LDS DCs** can run on one server. When **Active Directory** is operating as **AD DS**, only one **AD DS DC** can run on one server. However, several **AD LDS DCs** can coexist with one **AD DS DC** on one server. The **AD LDS DC** contains **full NC replicas** of the **config NC** and the **schema NC** in its **forest**.

domain functional level: A specification of functionality available in a **domain**. Must be less than or equal to the **DC functional level** of every **domain controller (DC)** that hosts a **replica** of the **domain's naming context (NC)**. Possible values in Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 are DS_BEHAVIOR_WIN2000, DS_BEHAVIOR_WIN2003_WITH_MIXED_DOMAINS, DS_BEHAVIOR_WIN2003, DS_BEHAVIOR_WIN2008, DS_BEHAVIOR_WIN2008R2, DS_BEHAVIOR_WIN2012, and DS_BEHAVIOR_WIN2012R2. See section 6.1.4.3 for information on how the **domain functional level** is determined. When **Active Directory** is operating as **Active Directory Lightweight Directory Services (AD LDS)**, **domain functional level** does not exist.

domain joined: A relationship between a machine and some **domain naming context (domain NC)** in which they share a secret. The shared secret allows the machine to authenticate to a **domain controller (DC)** for the **domain**.

domain local group: An **Active Directory group** that allows **user objects**, **global groups**, and **universal groups** from any **domain** as members. It also allows other **domain local groups** from within its **domain** as members. A **group object** g is a **domain local group** if and only if GROUP_TYPE_RESOURCE_GROUP is present in g!groupType. A security-enabled **domain local group** is valid for inclusion within **access control lists (ACLs)** from its own **domain**. If a **domain** is in **mixed mode**, then a security-enabled **domain local group** in that **domain** allows only **user objects** as members.

domain naming context (domain NC): A specific type of **naming context (NC)**, or an instance of that type. A **domain NC** can contain **security principal objects**. **Domain NCs** appear in the **global catalog (GC)**. A **domain NC** is hosted by one or more **domain controllers (DCs)** operating as **AD DS**. In **AD DS**, a **forest** has one or more **domain NCs**. The root of a **domain NC** is an **object of class** domainDNS. A **domain NC** cannot exist in **AD LDS**.

domain prefix: A **domain security identifier (SID)**, minus the **relative identifier (RID)** portion.

DSE: An acronym for a **directory service agent (DSA)**-specific **entry**.

DSA object: See **nTDSDSA object**.

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

dsname: A tuple that contains between one and three identifiers for an **object**. The possible identifiers are the **object's globally unique identifier (GUID)** (**attribute** objectGUID), **security identifier (SID)** (**attribute** objectSid), and **distinguished name (DN)** (**attribute** distinguishedName). A **dsname** can appear in a protocol message and as an **attribute** value (for example, a value of an **attribute** with **syntax** Object(DS-DN)).

dynamic object: An **object** with a time-to-die, **attribute** msDS-Entry-Time-To-Die. The **directory service (DS)** garbage-collects a **dynamic object** immediately after its time-to-die has passed. The **constructed attribute** entryTTL gives a **dynamic object's** current time-to-live, that is, msDS-Entry-Time-To-Die minus the current system time. For more information, see [RFC2589].

entry: A synonym for **object**. See also the "object" concept that is described in section 1, Introduction, under Pervasive Concepts.

existing-object: An **object** that is not a **tombstone**, **deleted-object**, or **recycled-object**.

Extended-Rights container: A **container** holding **objects** that correspond to **control access rights**. The **container** is a child of **configuration naming context (config NC)** and has **relative distinguished name (RDN)** CN=Extended-Rights.

File Replication Service (FRS): One of the services offered by a **domain controller (DC)**. The running/paused state of the **FRS** on a **DC** is available through protocols documented in section 6.3.

filter: One of the parameters in an **Lightweight Directory Access Protocol (LDAP)** search request. The **filter** specifies matching constraints for the candidate **objects**.

filtered attribute set: The subset of **attributes** that are not replicated to the **filtered partial NC replica** and the **filtered GC partial NC replica**. The **filtered attribute set** is part of the state of the **forest** and is used to control the **attributes** that replicate to a **read-only domain controller (RODC)**. The searchFlags **schema attribute** is used to define this set.

filtered GC partial NC replica: An **NC replica** that contains a **schema**-specified subset of **attributes** for the **objects**. The **attributes** consist of the **attributes** in the **GC partial attribute set (PAS)**, excluding those present in the **filtered attribute set**. A **filtered GC partial NC replica** is not writable.

filtered partial NC replica: An **NC replica** that contains all the **attributes** of the **objects**, excluding those **attributes** in the **filtered attribute set**. A **filtered partial NC replica** is not writable.

flexible single master operation (FSMO): A read or **update** operation on an **naming context (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 also **FSMO role**, **FSMO role owner**.

foreign principal object (FPO): A foreignSecurityPrincipal **object**.

forest: For **Active Directory Domain Services (AD DS)**, a set of **naming contexts (NCs)** consisting of one **schema naming context (schema NC)**, one **configuration naming context (config NC)**, one or more **domain naming contexts (domain NCs)**, and zero or more **application naming contexts (application NCs)**. Because a set of **NCs** can be arranged into a tree structure, a **forest** is also a set containing one or several trees of **NCs**. For **AD LDS**, a set of **NCs** consisting of one **schema NC**, one **config NC**, and zero or more **application NCs**. (In Microsoft documentation, an **AD LDS forest** is called a "configuration set".)

forest functional level: A specification of functionality available in a **forest**. It must be less than or equal to the **DC functional level** of every **DC** in the **forest**. For **Active Directory Domain Services (AD DS)**, possible values in Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 are DS_BEHAVIOR_WIN2000, DS_BEHAVIOR_WIN2003_WITH_MIXED_DOMAINS, DS_BEHAVIOR_WIN2003, DS_BEHAVIOR_WIN2008, DS_BEHAVIOR_WIN2008R2, DS_BEHAVIOR_WIN2012, and DS_BEHAVIOR_WIN2012R2. For **AD LDS**, the possible values in Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2 are DS_BEHAVIOR_WIN2003, DS_BEHAVIOR_WIN2008, DS_BEHAVIOR_WIN2008R2, DS_BEHAVIOR_WIN2012, and DS_BEHAVIOR_WIN2012R2. See section 6.1.4.4 for information on how the **forest functional level** is determined.

forest root domain NC: For **Active Directory Domain Services (AD DS)**, the **domain naming context (domain NC)** within a **forest** whose child is the **forest's configuration naming context (config NC)**. The **DNS name** of this **domain** serves as the **forest** name.

forward link attribute: A type of **attribute** whose values include **object references** (for example, an **attribute** of **syntax** Object(DS-DN)). The **forward link values** can be used to compute the values of a related **attribute**, a **back link attribute**, on other **objects**. A **forward link attribute** can exist with no corresponding **back link attribute**, but not vice versa.

FSMO role: A set of **objects** that can be **updated** in only one **NC replica** (the **FSMO role owner's replica**) at any given time.

FSMO role object: The **object** in the **directory** that represents a specific **FSMO role**. This **object** is an element of the **FSMO role** and contains the fSMORoleOwner attribute.

GC partial attribute set (PAS): The subset of **attributes** that replicate to a **GC partial NC replica**. The **partial attribute set** is part of the state of the **forest** and is used to control the **attributes** that replicate to **global catalog servers (GC servers)**. The isMemberOfPartialAttributeSet **schema attribute** is used to define this set.

GC partial NC replica: An **NC replica** that contains a **schema**-specified subset of **attributes** for the **objects** it contains. The subset of **attributes** consists of the **attributes** in the **GC partial attribute set (PAS)**.

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 and only if GROUP_TYPE_ACCOUNT_GROUP is present in g!groupType. A **global group** that is also a **security-enabled group** is valid for inclusion within ACLs anywhere in the **forest**. If a **domain** is in **mixed mode**, then a **global group** in that **domain** that is also a **security-enabled group** allows only **user object** as members. See also **domain local group**, **security-enabled group**.

group object: An **object of class** group representing a **group**. A class **group** has a **forward link attribute** member; the values of this **attribute** represent either elements of the **group** (for example, **objects of class** user or computer) or subsets of the **group** (**objects of class** group). 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 instance, used to represent email distribution lists).

GUID-based DNS name: A **DNS name** published for a **domain controller (DC)**. If a **DC's DSA GUID** is "52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8", and the **forest root domain NC's DNS name** is "fabrikam.com", then the **GUID-based DNS name** of the **DC** is "52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8._msdcs.fabrikam.com"..

inbound trust: A **trust** relationship between two **domains**, from the perspective of the **domain** that is trusted to perform **authentication**.

interdomain trust account: An account that stores information associated with a **domain trust** in the **domain controllers (DCs)** of the **domain** that is trusted to perform **authentication**.

intersite topology generator (ISTG): A **domain controller (DC)** within a given **site** that computes an **NC replica graph** for each **NC replica** on any **DC** in its **site**. This **DC** creates, **updates**, and deletes corresponding nTDSConnection **objects** for edges directed from **NC replicas** in other **sites** to **NC replicas** in its **site**.

invocationId: The invocationId **attribute**. An **attribute** of an **nTDSDSA object**. Its value is a unique identifier for a function that maps from **update sequence numbers (USNs)** to **updates** to the **NC replicas** of a **domain controller (DC)**. See also **nTDSDSA object**.

Knowledge Consistency Checker (KCC): An internal Windows component of the **Active Directory replication** used to create spanning trees for **domain controller (DC)**-to-**DC replication** and to translate those trees into settings of variables that implement the **replication** topology.

LDAP ping: A specific **Lightweight Directory Access Protocol (LDAP)** search that returns information about whether services are live on a **domain controller (DC)**.

lingering object: An **object** that still exists in an **NC replica** even though it has been deleted and garbage-collected from other **replicas**. This occurs, for instance, when a **domain controller (DC)** goes offline for longer than the **tombstone lifetime**.

mailslot: A form of datagram communication using the Server Message Block (SMB) protocol, as specified in [MS-MAIL].

mailslot ping: A specific **mailslot** request that returns information about whether services are live on a **domain controller (DC)**.

most specific object class: In a sequence of **object classes** related by **inheritance**, the class that none of the other classes inherits from. The special **object class** top is less specific than any other class.

naming context (NC): An **NC** is a set of objects organized as a tree. It is referenced by a **DSName**. 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 **security identifier (SID)** of the **DSName**, if present, is the objectSid **attribute** of the tree root; for **Active Directory Domain Services (AD DS)**, the **SID** is present if and only if the **NC** is a **domain naming context (domain NC)**. **Active Directory** supports organizing several **NCs** into a tree structure.

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

NC replica graph: A directed graph containing **NC replicas** as nodes and repsFrom tuples as inbound edges by which **originating updates** replicate from each full **replica** of a given **naming context (NC)** to all other **NC replicas** of the **NC**, directly or transitively.

NetBIOS: A protocol family including name resolution, datagram, and connection services. For more information, see [RFC1001] and [RFC1002].

NetBIOS domain name: The name registered by **domain controllerS (DCs)** on [1C] records of the **NBNS** (WINS) server (see section 6.3.4). For details of NetBIOS name registration, see [MS-WPO] sections 7.1.4 and 10.4.

NetBIOS Name Service (NBNS): The name service for **NetBIOS**. For more information, see [RFC1001] and [RFC1002].

Netlogon: A component of Windows that authenticates a computer and provides other services. The running/paused state of **Netlogon** on a **domain controller (DC)** is available through protocols documented in section 6.3.

nTDSDSA object: An **object of class** nTDSDSA, representing a **domain controller (DC)** in the **configuration naming context (config NC)**.

object: See section 1, Introduction, under Pervasive Concepts.

object class: See section 1, Introduction, under Pervasive Concepts.

object class name: The lDAPDisplayName of the classSchema **object** of an **object class**. This document consistently uses **object class names** to denote **object classes**; for example, user and group are both **object classes**. The correspondence between **Lightweight Directory Access Protocol (LDAP)** display names and numeric **object identifiers (OIDs)** in the **Active Directory schema** is defined in the appendices of these documents: [MS-ADSC], [MS-ADA1], [MS-ADA2], and [MS-ADA3].

object identifier (OID): A sequence of numbers in a format defined by [RFC1778].

object reference: An **attribute** value that references an **object**; reading a reference gives the **distinguished name (DN)** or full **dsname** of the **object**.

optional feature: A non-default behavior that modifies the **Active Directory** state model. An **optional feature** is enabled or disabled in a specific scope, such as a **forest** or a **domain**.

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.

outbound trust: A **trust** relationship between two **domains**, from the perspective of the **domain** that trusts another **domain** to perform **authentication**.

parent naming context (parent NC): Given **naming contexts (NCs)** with their corresponding **distinguished names (DNs)** forming a child and parent relationship, the **NC** in the parent relationship is referred as the **parent NC**.

parent object: See section 1, Introduction, under Pervasive Concepts.

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

Partitions container: A **child object** of the **configuration naming context (config NC)** root. The **relative distinguished name (RDN)** of the **Partitions container** is "cn=Partitions" and its class is crossRefContainer. See also **crossRef object**.

prefix table: A data structure that is used to translate between an **object identifier (OID)** and an **ATTRTYP**.

primary domain controller (PDC): A **domain controller (DC)** designated to track changes made to the accounts of all computers on a **domain**. It 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**.

primary group: The group **object** identified by the primaryGroupID **attribute** of a **user object**. The **primary group's** objectSid equals the user's objectSid, with its **relative identifier (RID)** portion replaced by the primaryGroupID value. The user is considered a member of its **primary group**.

principal: A unique entity identifiable by a **security identifier (SID)** that is typically the requester of access to securable **objects** or resources. It often corresponds to a human user but can also be a computer or service. It is sometimes referred to as a **security principal**.

read permission: **Authorization** to read an **attribute** of an **object**.

read-only domain controller (RODC): A **domain controller (DC)** that does not accept **originating updates**. Additionally, an **RODC** does not perform outbound **replication**.

read-only full NC replica: An **NC replica** that contains all **attributes** of the **objects** it contains, and does not accept **originating updates**.

Recycle Bin: An **optional feature** that modifies the state model of **object** deletions and undeletions, making undeletion of **deleted-objects** possible without loss of the **object's attribute** values.

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

relative distinguished name (RDN): The name of an **object** relative to its parent. This is the leftmost attribute-value pair in the **distinguished name (DN)** of an **object**. For example, in the **DN** "cn=Peter Houston, ou=NTDEV, dc=microsoft, dc=com", the **RDN** is "cn=Peter Houston". For more information, see [RFC2251].

relative identifier (RID): The last item in the series of **SubAuthority** values in a **security identifier (SID)** (see [MS-DTYP] section 2.3). Differences in the **RID** are what distinguish the different **SIDs** generated within a **domain**.

replica: See section 1, Introduction, under Pervasive Concepts.

replicated attribute: An **attribute** whose values are replicated to other **NC replicas**. An **attribute** is replicated if its attributeSchema **object** *o* does not have a value for the systemFlags **attribute**, or if the FLAG_ATTR_NOT_REPLICATED bit (bit 0) of *o*!systemFlags is zero.

replication cycle: A series of one or more **replication** responses associated with the same **invocationId**, concluding with the return of a new up-to-date vector.

root domain: The unique **domain naming context (domain NC)** of an **Active Directory forest** that is the parent of the **forest's configuration naming context (config NC)**. The **config NC's relative distinguished name (RDN)** is "cn=Configuration" relative to the root **object** of the **root domain**.

root DSE (rootDSE): A nameless **entry** containing the configuration status of the **Lightweight Directory Access Protocol (LDAP)** server. Typically, access to at least a portion of the **root DSE** is available to unauthenticated clients, allowing them to determine the **authentication** methods supported by the server.

schema NC: A specific type of **NC** or an instance of that type. A **forest** has a single **schema NC**, which is replicated to each **domain controller (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**. A schema NC cannot contain **security principal objects**.

Secure Sockets Layer (SSL): A means of providing privacy and data protection between a client and a server. It may also be used to provide **authentication** between the two systems. For more information, see [SSL3].

secret attribute: Any of the following **attributes**: currentValue, dBCSPwd, initialAuthIncoming, initialAuthOutgoing, lmPwdHistory, ntPwdHistory, priorValue, supplementalCredentials, trustAuthIncoming, trustAuthOutgoing, and unicodePwd.

security context: A data structure containing **authorization** information for a particular **security principal** in the form of a collection of **security identifiers (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 (SD): A data structure containing the security information associated with a securable **object**. A **security descriptor (SD)** identifies an **object's** owner by **security identifier (SID)**. If access control is configured for the **object**, its **SD** contains a **discretionary access control list (DACL)** with **SIDs** for the **security principals** that are allowed or denied access. The **SD** format is specified in [MS-DTYP] section 2.4.6; a string representation of **SDs**, called Security Descriptor Definition Language (SDDL), is specified in [MS-DTYP] section 2.5.1.

security-enabled group: A **group object** with GROUP_TYPE_SECURITY_ENABLED present in its groupType **attribute**. Only **security-enabled groups** are added to a **security context**. See also **group object**.

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 that is 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 **object classes** are examples of **security principal object** classes (though not every group **object** is a **security principal object**). In **AD LDS**, any **object** containing the msDS-BindableObject **auxiliary class** is a **security principal**. See also **computer object**, **group object**, and **user object**.

server object: A class of **object** in the **configuration naming context (config NC)**. A **server object** can have an **nTDSDSA object** as a child. See also **nTDSDSA object**.

Simple Authentication and Security Layer (SASL): An **authentication** mechanism that is used by **Lightweight Directory Access Protocol (LDAP)** and is defined in [RFC2222].

simple bind: A bind with the simple **authentication** option enabled according to [RFC2251].

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 users log in, **Active Directory** clients find **domain controllers (DCs)** that are in the same **site** as the user, or near the same **site** if there is no **DC** in the **site**. See also **Knowledge Consistency Checker (KCC)**.

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

site settings object: For a given **site** with **site object** *s*, its **site settings object** *o* is the child of *s* such that *o* is of class nTDSSiteSettings and the **relative distinguished name (RDN)** of *o* is CN=NTDS Site Settings. See also **site object**.

SRV record: A type of information record in DNS that maps the name of a service to the **DNS name** of a server that offers that service. **domain controllers (DCs)** advertise their capabilities by publishing **SRV records** in DNS.

stamp: Information describing an **originating update** by a **domain controller (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 conventional data values.

SubAuthority: A **security identifier (SID)** can have a variable-length array of unsigned, 32-bit integer values. Each of these values is called a **SubAuthority**. All **SubAuthority** values excluding the last one collectively identify a **domain**. The last value, termed as the **relative identifier (RID)**, identifies a particular **group** or account relative to the **domain**. For more information, see [SIDD].

syntax: See **attribute syntax**.

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

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

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

top level name (TLN): The **DNS name** of the **forest root domain NC**.

transitive membership: An indirect group membership that occurs when an object is a member of a group that is a member of a second group. The object is a member of the second group through a **transitive membership**.

Transport Layer Security (TLS): The successor to **Secure Sockets Layer (SSL)**. As with **SSL**, it provides privacy, data protection, and optionally **authentication** between a client and server. See [RFC2246].

trust: A relationship between two **domains**. If **domain** A trusts **domain** B, **domain** A accepts **domain** B's **authentication** and **authorization** statements for **principals** represented by **security principal objects** in **domain** B.

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 and only if GROUP_TYPE_UNIVERSAL_GROUP is present in g!groupType. A **security-enabled universal group** is valid for inclusion within ACLs anywhere in the **forest**. If a **domain** is in **mixed mode**, then a **universal group** cannot be created in that **domain**. See also **domain local group**, **security-enabled group**.

update: An add, modify, or delete of one or more **objects** or **attribute** values. See also **originating update**, **replicated update**.

update sequence number (USN): A monotonically increasing sequence number used in assigning a **stamp** to an **originating update**. See also **invocationId**.

user object: An **object of class** user. A **user object** is a **security principal object**; the **principal** is a person operating or service entity running on the computer. The shared secret allows the person or service entity to authenticate itself.

UTF-8: An 8-bit, variable-width encoding of **Unicode** characters.

UTF-16: A 16-bit, variable-width encoding form of **Unicode** characters.

Virtual List View (VLV) search: Refers to a **Lightweight Directory Access Protocol (LDAP)** search operation that enables the server to return a contiguous subset of a large search result set. **LDAP** controls LDAP_CONTROL_VLVREQUEST and LDAP_CONTROL_VLVRESPONSE (section 3.1.1.3.4.1.17) that are used to perform a **VLV search**.

well-known object (WKO): An **object** within an **naming context (NC)** that can be located using a fixed **globally unique identifier (GUID)**.

Windows security descriptor: See **security descriptor (SD)**.

writable NC replica: An **NC replica** that accepts **originating updates**. A **writable NC replica** is always **full**, but a **full NC replica** is not always writable. See also **read-only full NC replica**.

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

1.2 References

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

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

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. 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

[GRAY] Gray, J. and Reuter, A., "Transaction Processing: Concepts and Techniques", San Mateo, CA: Morgan Kaufmann Publishers, 1993, ISBN: 1558601902.

[IEEE1003.1] The Open Group, "IEEE Std 1003.1, 2004 Edition", 2004, http://www.unix.org/version3/ieee_std.html

Note Registration is required to view or download this specification.

[ISO-8601] International Organization for Standardization, "Data Elements and Interchange Formats - Information Interchange - Representation of Dates and Times", ISO/IEC 8601:2004, December 2004, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40874&ICS1=1&ICS2=140&ICS3=30

Note There is a charge to download the specification.

[ISO/IEC-9899] International Organization for Standardization, "Programming Languages - C", ISO/IEC 9899:TC2, May 2005, http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf

[ISO/IEC-14977] International Organization for Standardization, "Information technology -- Syntactic metalanguage -- Extended BNF", ISO/IEC 14977:1996, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=26153

[ITUX680] ITU-T, "Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", Recommendation X.680, July 2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf

[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

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

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

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

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

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

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

[MS-APDS] Microsoft Corporation, "Authentication Protocol Domain Support".

[MS-CTA] Microsoft Corporation, "Claims Transformation Algorithm".

[MS-DRSR] Microsoft Corporation, "Directory Replication Service (DRS) Remote Protocol".

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

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

[MS-FRS1] Microsoft Corporation, "File Replication Service Protocol".

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

[MS-GPSB] Microsoft Corporation, "Group Policy: Security Protocol Extension".

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

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

[MS-MAIL] Microsoft Corporation, "Remote Mailslot Protocol".

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol".

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

[MS-PAC] Microsoft Corporation, "Privilege Attribute Certificate Data Structure".

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

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

[MS-SFU] Microsoft Corporation, "Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol".

[MS-SPNG] Microsoft Corporation, "Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) Extension".

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

[MS-UCODEREF] Microsoft Corporation, "Windows Protocols Unicode Reference".

[MS-W32T] Microsoft Corporation, "W32Time Remote Protocol".

[MSASRT] Microsoft Corporation, "Active Directory Sorting Weight Table", June 2006. http://www.microsoft.com/downloads/details.aspx?FamilyId=60B16E13-D1E5-4865-AE6D-0C13A9D7036F&displaylang=en

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981, http://www.ietf.org/rfc/rfc791.txt

[RFC1001] Network Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", STD 19, RFC 1001, March 1987, http://www.ietf.org/rfc/rfc1001.txt

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

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

[RFC1035] Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1035, November 1987, http://www.ietf.org/rfc/rfc1035.txt

[RFC1088] McLaughlin III, L., "A Standard for the Transmission of IP Datagrams over NetBIOS Networks", STD 48, RFC 1088, February 1989, http://www.ietf.org/rfc/rfc1088.txt

[RFC1166] Kirkpatrick, S., Stahl, M., and Recker, M., "Internet Numbers", RFC 1166, July 1990, http://www.ietf.org/rfc/rfc1166.txt

[RFC1274] Barker, P., and Kille, S., "The COSINE and Internet X.500 Schema", RFC 1274, November 1991, http://www.ietf.org/rfc/rfc1274.txt

[RFC1278] Hardcastle-Kille, S. E., "A string encoding of Presentation Address", RFC 1278, November 1991, http://www.ietf.org/rfc/rfc1278.txt

[RFC1777] Yeong, W., Howes, T., and Kille, S., "Lightweight Directory Access Protocol", RFC 1777, March 1995, http://www.ietf.org/rfc/rfc1777.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

[RFC1798] Young, A., "Connection-less Lightweight X.500 Directory Access Protocol", RFC 1798, June 1995, http://www.ietf.org/rfc/rfc1798.txt

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

[RFC2078] Linn, J., "Generic Security Service Application Program Interface, Version 2", RFC 2078, January 1997, http://www.ietf.org/rfc/rfc2078.txt

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

[RFC2136] Thomson, S., Rekhter Y. and Bound, J., "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997, http://www.ietf.org/rfc/rfc2136.txt

[RFC2222] Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997, http://www.ietf.org/rfc/rfc2222.txt

[RFC2246] Dierks, T., and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[RFC2247] Kille, S., Wahl, M., Grimstad, A., et al., "Using Domains in LDAP/X.500 Distinguished Names", RFC 2247, January 1998, http://www.ietf.org/rfc/rfc2247.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

[RFC2255] Howes, T., and Smith, M., "The LDAP URL Format", RFC 2255, December 1997, http://www.ietf.org/rfc/rfc2255.txt

[RFC2256] Wahl, M., "A Summary of the X.500(96) User Schema for use with LDAPv3", RFC 2256, December 1997, http://www.ietf.org/rfc/rfc2256.txt

[RFC2307] Howard, L., "An Approach for Using LDAP as a Network Information Service", RFC 2307, March 1998, http://www.ietf.org/rfc/rfc2307.txt

[RFC2373] Hinden, R., and Deering, S., "IP Version 6 Addressing Architecture", RFC 2373, July 1998, http://www.ietf.org/rfc/rfc2373.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

[RFC2696] Weider, C., Herron, A., Anantha, A., and Howes, T., "LDAP Control Extension for Simple Paged Results Manipulation", RFC 2696, September 1999, http://www.ietf.org/rfc/rfc2696.txt

[RFC2782] Gulbrandsen, A., Vixie, P., and Esibov, L., "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000, http://www.ietf.org/rfc/rfc2782.txt

[RFC2798] Smith, M., "Definition of the inetOrgPerson LDAP Object Class", RFC 2798, April 2000, http://www.ietf.org/rfc/rfc2798.txt

[RFC2829] Wahl, M., Alvestrand, H., Hodges, J., and Morgan, R., "Authentication Methods for LDAP", RFC 2829, May 2000, http://www.ietf.org/rfc/rfc2829.txt

[RFC2830] Hodges, J., Morgan, R., and Wahl, M., "Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security", RFC 2830, May 2000, http://www.ietf.org/rfc/rfc2830.txt

[RFC2831] Leach, P., and Newman, C., "Using Digest Authentication as a SASL Mechanism", RFC 2831, May 2000, http://www.ietf.org/rfc/rfc2831.txt

[RFC2849] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical Specification", RFC 2849, June 2000, http://www.ietf.org/rfc/rfc2849.txt

[RFC2891] Howes, T., Wahl, M., and Anantha, A., "LDAP Control Extension for Server Side Sorting of Search Results", RFC 2891, August 2000, http://www.ietf.org/rfc/rfc2891.txt

[RFC3377] Hodges, J., and Morgan, R., "Lightweight Directory Access Protocol (v3): Technical Specification", RFC 3377, September 2002, http://www.ietf.org/rfc/rfc3377.txt

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005, http://www.ietf.org/rfc/rfc3961.txt

[RFC4120] Neuman, C., Yu, T., Hartman, S., and Raeburn, K., "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005, http://www.ietf.org/rfc/rfc4120.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

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and Ingersoll, W., "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005, http://www.ietf.org/rfc/rfc4178.txt

[RFC4370] Weltman, R., "Lightweight Directory Access Protocol (LDAP) Proxied Authorization Control", RFC 4370, February 2006, http://www.ietf.org/rfc/rfc4370.txt

[RFC4532] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP) "Who Am I?" Operation", RFC 4532, June 2006, http://www.ietf.org/rfc/rfc4532.txt

[RFC4757] Jaganathan, K., Zhu, L., and Brezak, J., "The RC4-HMAC Kerberos Encryption Types Used by Microsoft Windows", RFC 4757, December 2006, http://www.ietf.org/rfc/rfc4757.txt

[SSL3] Netscape, "SSL 3.0 Specification", http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00

If you have any trouble finding [SSL3], please check here.

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

[XMLSCHEMA2/2] Biron, P.V., and Malhotra, A., Eds., "XML Schema Part 2: Datatypes Second Edition", W3C Recommendation, October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

1.2.2 Informative References

[ADDLG] Microsoft Corporation, "Security Briefs: Credentials and Delegation", September 2005, http://msdn.microsoft.com/en-us/magazine/cc163740.aspx

[LISP15] McCarthy, J., Abrahams, P., Edwards, D., Hart, T., and Levin, M., "LISP 1.5 Programmers Manual", Cambridge, MA: The M.I.T. Press, 1965, ISBN-10: 0262130114.

[MS-ADDM] Microsoft Corporation, "Active Directory Web Services: Data Model and Common Elements".

[MS-AUTHSOD] Microsoft Corporation, "Authentication Services Protocols Overview".

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

[MS-GPOD] Microsoft Corporation, "Group Policy Protocols Overview".

[MS-SYS] Microsoft Corporation, "Windows System Overview".

[MS-WPO] Microsoft Corporation, "Windows Protocols Overview".

[MS-XCA] Microsoft Corporation, "Xpress Compression Algorithm".

[MSKB-298713] Microsoft Corporation, "How to prevent overloading on the first domain controller during domain upgrade", Version 6.8, December 2007, http://support.microsoft.com/kb/298713

[SIDD] Microsoft Corporation, "How Security Identifiers Work", March 2003, http://technet.microsoft.com/en-us/library/cc778824.aspx

[VLVDRAFT] Boreham, D., Sermersheim, J., and Kashi, A., "LDAP Extensions for Scrolling View Browsing of Search Results", draft-ietf-ldapext-ldapv3-vlv-09, November 2002, http://tools.ietf.org/html/draft-ietf-ldapext-ldapv3-vlv-09

1.3 Overview

This is the primary specification for Active Directory. The state model for this specification is prerequisite to the other specifications for Active Directory: [MS-DRSR] and [MS-SRPL].

Active Directory is either deployed as AD DS or as AD LDS. This document describes both forms. When the specification does not refer specifically to AD DS or AD LDS, it applies to both.

The remainder of this section describes the structure of this document.

The basic state model is specified in section 3.1.1.1. The basic state model is prerequisite to the remainder of the document. Section 3.1.1.1 also includes descriptive content to introduce key concepts and refer to places in the document where the full specification is given.

The **schema** completes the state model and is specified in section 3.1.1.2. The schema is prerequisite to the remainder of the document.

Active Directory is a server for LDAP. Section 3.1.1.3 specifies the extensions and variations of LDAP that are supported by Active Directory.

LDAP is an access protocol that determines very little about the behavior of the data being accessed. Section 3.1.1.4 specifies read (LDAP Search) behaviors, and section 3.1.1.5 specifies update (LDAP Add, Modify, Modify DN, Delete) behaviors. Section 3.1.1.6 specifies background tasks required due to write operations, to the extent that those tasks are exposed by protocols.

One of the update behaviors is the maintenance of the change log for use by Windows NT 4.0 operating system **backup domain controller (BDC)** replication [MS-NRPC] section 3.6. The maintenance of this change log is specified in section 3.1.1.7.

The security services that Active Directory offers clients of LDAP are specified in section 5.1.

Active Directory contains a number of objects, visible through LDAP, that have special significance to the system. Section 6.1 specifies these objects.

A server running Active Directory is part of a distributed system that performs replication. The **Knowledge Consistency Checker (KCC)** is a component that is used to create spanning trees for **DC**-to-DC replication, and is specified in section 6.2.

A server running Active Directory is responsible for publishing the services that it offers, in order to eliminate the administrative burden of configuring clients to use particular servers running Active Directory. A server running Active Directory also implements the server side of the **LDAP ping** and **mailslot ping** protocols to aid clients in selecting among all the servers offering the same service. Section 6.3 specifies how a server running Active Directory publishes its services, and how a client needing some service can use this publication plus the LDAP ping or mailslot ping to locate a suitable server.

Computers in a network with Active Directory can be put into a state called "**domain joined**"; when in this state, the computer can authenticate itself. Section 6.4 specifies both the state in Active Directory and the state on a computer required for the domain joined state.

Each type of data stored in Active Directory has an associated function that compares two values to determine if they are equal and, if not, which is greater. Section 3.1.1.2 specifies all but one of these functions; the methodology for comparing two **Unicode** strings is specified in section 6.5.

1.4 Relationship to Other Protocols

This is the primary specification for Active Directory. The state model for this specification is prerequisite to the specification for Active Directory described in [MS-DRSR]. This Active Directory technical specification depends on the following protocols:

- Lightweight Directory Access Protocol (LDAP)
- Remote Procedure Call (RPC)
- Domain Name System (DNS)

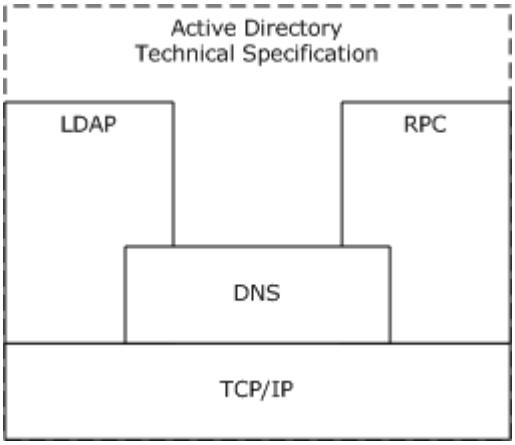


Figure 1: Protocol and technical specification relationships

Other protocols make use of implementations of the Active Directory Technical Specification as a data store.

1.5 Prerequisites/Preconditions

Active Directory requires an IP network and a DNS infrastructure.

1.6 Applicability Statement

Active Directory is not suitable for storing very large attribute values because, for instance, there is no provision for check-pointing a large data transfer to allow restart after a failure. The bandwidth and latency of typical networks makes Active Directory unsuitable for storing volatile data in **replicated attributes**. Active Directory is especially suitable for storing security account data, including passwords, and email address book data.

1.7 Versioning and Capability Negotiation

Capability negotiation is performed using the **root DSE** as described in section 3.1.1.3.2.

1.8 Vendor-Extensible Fields

LDAP is not extensible by Active Directory applications. Applications extend the **directory** by adding objects, including **schema objects** to control the application objects.

1.9 Standards Assignments

Active Directory's extensions and variations of LDAP have no standards assignments. AD DS uses private allocations for its LDAP **global catalog (GC)** port (3268) and LDAP GC port with **Secure Sockets Layer**/**Transport Layer Security** (SSL/TLS) (3269).

2 Messages

The following sections specify how LDAP is transported and denote common information such as bit flag values.

2.1 Transport

LDAP transport is specified in section 3.1.1.3, and in [RFC2251] section 5 (for LDAPv3), in [RFC1777] section 3 (for LDAPv2), and in [RFC1798] section 3.1 (for both LDAPv2 and LDAPv3).

2.2 Message Syntax

This section specifies types and data structures used in the remainder of this document. These type specifications reference the following:

- DWORD and FILETIME types: [MS-DTYP] sections 2.2.9 and 2.3.3.
- repsFrom, repsTo, replUpToDateVector abstract attributes of an **NC replica**: [MS-DRSR] sections 5.169, 5.170, and 5.165.
- ReplUpToDateVector abstract type of a NC replica: [MS-DRSR] section 5.165.
- kCCFailedConnections, kCCFailedLinks, RPCClientContexts, RPCOutgoingContexts, ldapConnections, and replicationQueue variables of a DC: [MS-DRSR] sections 5.110, 5.111, 5.174, 5.175, 5.115, and 5.163.
- **Stamp** variable of an attribute: [MS-DRSR] section 5.11.
- Stamp variable of a **link value**: [MS-DRSR] section 5.117.
- DS_REPL_ATTR_META_DATA_2, DS_REPL_CURSOR_3W, DS_REPL_KCC_DSA_FAILUREW, DS_REPL_NEIGHBORW, DS_REPL_OPW, DS_REPL_VALUE_META_DATA_2 types: [MS-DRSR] section 4.1.13.1.
- IDL_DRSGetReplInfo method: [MS-DRSR] section 4.1.13.

2.2.1 LCID-Locale Mapping Table

The following table maps Windows locales (for example, French - France, Irish - Ireland) to numeric identifiers called locale identifiers (LCIDs). These numeric identifiers are used as input to the Unicode string comparison function specified in section 6.5. They are also used to name Display Specifier **containers**, specified in section 6.1.1.2.3, "Display Specifiers Container".

LCID	Language	Location
0436	Afrikaans	South Africa
041c	Albanian	Albania
0401	Arabic	Saudi Arabia
0801	Arabic	Iraq
0c01	Arabic	Egypt
1001	Arabic	Libya

LCID	Language	Location
1401	Arabic	Algeria
1801	Arabic	Morocco
1c01	Arabic	Tunisia
2001	Arabic	Oman
2401	Arabic	Yemen
2801	Arabic	Syria
2c01	Arabic	Jordan
3001	Arabic	Lebanon
3401	Arabic	Kuwait
3801	Arabic	U.A.E.
3c01	Arabic	Bahrain
4001	Arabic	Qatar
042b	Armenian	Armenia
082c	Azeri (Cyrillic)	Azerbaijan
042c	Azeri (Latin)	Azerbaijan
042d	Basque	Basque
0423	Belarusian	Belarus
201a	Bosnian (Cyrillic)	Bosnia and Herzegovina
141a	Bosnian (Latin)	Bosnia and Herzegovina
0402	Bulgarian	Bulgaria
0403	Catalan	Catalan
0004	Chinese	Simplified
0404	Chinese	Taiwan
0804	Chinese	PRC
0c04	Chinese	Hong Kong SAR
1004	Chinese	Singapore
1404	Chinese	Macao SAR
7c04	Chinese	Traditional
041a	Croatian	Croatia
101a	Croatian (Latin)	Bosnia and Herzegovina

LCID	Language	Location
0405	Czech	Czech Republic
0406	Danish	Denmark
0465	Divehi	Maldives
0813	Dutch	Belgium
0413	Dutch	Netherlands
1009	English	Canada
2009	English	Jamaica
2409	English	Caribbean
2809	English	Belize
2c09	English	Trinidad
0809	English	United Kingdom
1809	English	Ireland
1c09	English	South Africa
3009	English	Zimbabwe
0c09	English	Australia
1409	English	New Zealand
3409	English	Philippines
0409	English	United States
0425	Estonian	Estonia
0438	Faroese	Faroe Islands
0464	Filipino	Philippines
040b	Finnish	Finland
0c0c	French	Canada
040c	French	France
180c	French	Monaco
100c	French	Switzerland
080c	French	Belgium
140c	French	Luxembourg
0462	Frisian	Netherlands
0456	Galician	Galician

LCID	Language	Location
0437	Georgian	Georgia
0407	German	Germany
0807	German	Switzerland
0c07	German	Austria
1407	German	Liechtenstein
1007	German	Luxembourg
0408	Greek	Greece
0447	Gujarati	India
040d	Hebrew	Israel
0439	Hindi	India
040e	Hungarian	Hungary
040f	Icelandic	Iceland
0421	Indonesian	Indonesia
085d	Inuktitut (Latin)	Canada
083c	Irish	Ireland
0434	isiXhosa	South Africa
0435	isiZulu	South Africa
0410	Italian	Italy
0810	Italian	Switzerland
0411	Japanese	Japan
044b	Kannada	India
043f	Kazakh	Kazakhstan
0441	Kiswahili	Kenya
0457	Konkani	India
0412	Korean	Korea
0440	Kyrgyz	Kirghizstan
0426	Latvian	Latvia
0427	Lithuanian	Lithuania
046e	Luxembourgish	Luxembourg
042f	Macedonian (FYROM)	Macedonia, Former Yugoslav Republic of

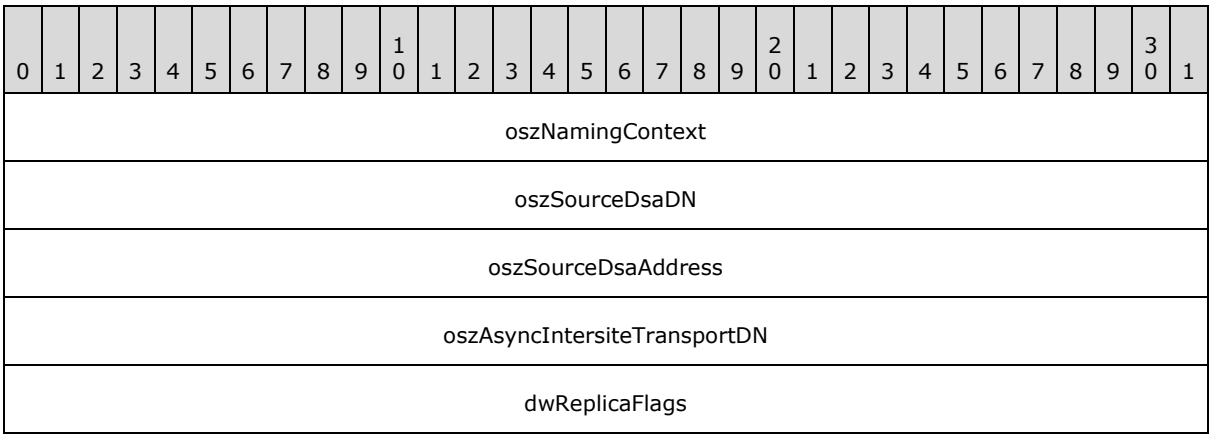
LCID	Language	Location
043e	Malay	Malaysia
083e	Malay	Brunei Darussalam
043a	Maltese	Malta
0481	Maori	New Zealand
047a	Mapudungun	Chile
044e	Marathi	India
047c	Mohawk	Mohawk
0450	Mongolian (Cyrillic)	Mongolia
0461	Nepali	Nepal
0414	Norwegian (Bokmål)	Norway
0814	Norwegian (Nynorsk)	Norway
0463	Pashto	Afghanistan
0429	Persian	Iran
0415	Polish	Poland
0416	Portuguese	Brazil
0816	Portuguese	Portugal
0446	Punjabi (Gurmukhi)	India
046b	Quechua	Bolivia
086b	Quechua	Ecuador
0c6b	Quechua	Peru
0418	Romanian	Romania
0417	Romansh	Switzerland
0419	Russian	Russia
243b	Sami, Inari	Finland
143b	Sami, Lule	Sweden
103b	Sami, Lule	Norway
043b	Sami, Northern	Norway
083b	Sami, Northern	Sweden
0c3b	Sami, Northern	Finland
203b	Sami, Skolt	Finland

LCID	Language	Location
183b	Sami, Southern	Norway
1c3b	Sami, Southern	Sweden
044f	Sanskrit	India
0c1a	Serbian (Cyrillic)	Serbia
0c1a	Serbian (Cyrillic)	Montenegro
1c1a	Serbian (Cyrillic)	Bosnia and Herzegovina
081a	Serbian (Latin)	Serbia
081a	Serbian (Latin)	Montenegro
181a	Serbian (Latin)	Bosnia and Herzegovina
046c	Sesotho sa Leboa	South Africa
0432	Setswana	South Africa
041b	Slovak	Slovakia
0424	Slovenian	Slovenia
080a	Spanish	Mexico
100a	Spanish	Guatemala
140a	Spanish	Costa Rica
180a	Spanish	Panama
1c0a	Spanish	Dominican Republic
200a	Spanish	Venezuela
240a	Spanish	Colombia
280a	Spanish	Peru
2c0a	Spanish	Argentina
300a	Spanish	Ecuador
340a	Spanish	Chile
3c0a	Spanish	Paraguay
400a	Spanish	Bolivia
440a	Spanish	El Salvador
480a	Spanish	Honduras
4c0a	Spanish	Nicaragua
500a	Spanish	Commonwealth of Puerto Rico

LCID	Language	Location
380a	Spanish	Uruguay
0c0a	Spanish (International Sort)	Spain
040a	Spanish (Traditional Sort)	Spain
041d	Swedish	Sweden
081d	Swedish	Finland
045a	Syriac	Syria
0449	Tamil	India
0444	Tatar	Russia
044a	Telugu	India
041e	Thai	Thailand
041f	Turkish	Turkey
0422	Ukrainian	Ukraine
0420	Urdu	Pakistan
0843	Uzbek (Cyrillic)	Uzbekistan
0443	Uzbek (Latin)	Uzbekistan
042a	Vietnamese	Vietnam
0452	Welsh	United Kingdom

2.2.2 DS_REPL_NEIGHBORW_BLOB

The DS_REPL_NEIGHBORW_BLOB structure is a representation of a tuple from the repsFrom or repsTo abstract attribute of an NC replica. This structure, retrieved using an LDAP search method, is an alternative representation of DS_REPL_NEIGHBORW, retrieved using the IDL_DRSGetReplInfo **RPC** method.



dwReserved
uuidNamingContextObjGuid
...
...
...
uuidSourceDsaObjGuid
...
...
...
uuidSourceDsaInvocationID
...
...
...
uuidAsyncIntersiteTransportObjGuid
...
...
...
usnLastObjChangeSynced
...
usnAttributeFilter
...
ftimeLastSyncSuccess
...

ftimeLastSyncAttempt
...
dwLastSyncResult
cNumConsecutiveSyncFailures
data (variable)
...

oszNamingContext (4 bytes): A 32-bit offset, in bytes, from the address of this structure to a null-terminated Unicode string that contains the **naming context (NC)** to which this replication state data pertains.

oszSourceDsaDN (4 bytes): A 32-bit offset, in bytes, from the address of this structure to a null-terminated Unicode string that contains the **distinguished name (DN)** of the **nTDSDSA object** of the source server to which this replication state data pertains. Each source server has different associated neighbor data.

oszSourceDsaAddress (4 bytes): A 32-bit offset, in bytes, from the address of this structure to a null-terminated Unicode string that contains the transport-specific network address of the source server—that is, a directory name service name for RPC/IP replication, or a **Simple Mail Transfer Protocol (SMTP)** address for an SMTP replication.

oszAsyncIntersiteTransportDN (4 bytes): A 32-bit offset, in bytes, from the address of this structure to a null-terminated Unicode string that contains the DN of the interSiteTransport object (as specified in [MS-ADSC] section 2.63) that corresponds to the transport over which replication is performed. This member contains NULL for RPC/IP replication.

dwReplicaFlags (4 bytes): A 32-bit bit field containing a set of flags that specify attributes and options for the replication data. This can be zero or a combination of one or more of the following flags presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	NCN	CC	X	X	X	X	X	X	NS	X	X	X	FSN	FSP	X	X	X	X	X	X	TWS	X	AIT	DSS	SOS	W	X	X	X	X

X: Unused. Must be zero and ignored.

W (DS_REPL_NBR_WRITEABLE, 0x00000010): The NC replica is writable.

SOS (DS_REPL_NBR_SYNC_ON_STARTUP, 0x00000020): Replication of this NC from this source is attempted when the destination server is booted.

DSS (DS_REPL_NBR_DO_SCHEDULED_SYNCS, 0x00000040): Perform replication on a schedule.

AIT (DS_REPL_NBR_USE_ASYNC_INTERSITE_TRANSPORT, 0x00000080): Perform replication indirectly through the Inter-Site Messaging Service. This flag is set only when replicating over SMTP. This flag is not set when replicating over inter-site RPC/IP.

TWS (DS_REPL_NBR_TWO_WAY_SYNC, 0x00000200): When inbound replication is complete, the destination server requests the source server to synchronize in the reverse direction.

FSP (DS_REPL_NBR_FULL_SYNC_IN_PROGRESS, 0x00010000): The destination server is performing a full synchronization from the source server.

FSN (DS_REPL_NBR_FULL_SYNC_NEXT_PACKET, 0x00020000): The last packet from the source indicated a modification of an object that the destination server has not yet created. The next packet to be requested instructs the source server to put all attributes of the modified object into the packet.

NS (DS_REPL_NBR_NEVER_SYNCED, 0x00200000): A synchronization has never been successfully completed from this source.

CC (DS_REPL_NBR_COMPRESS_CHANGES, 0x10000000): Changes received from this source are to be compressed.

NCN (DS_REPL_NBR_NO_CHANGE_NOTIFICATIONS, 0x20000000): Applies to repsFrom only. The domain controller (DC) storing this repsFrom is not configured to receive change notifications from this source.

dwReserved (4 bytes): Reserved for future use.

uuidNamingContextObjGuid (16 bytes): A **GUID** structure, as defined in [MS-DTYP] section 2.3.4, specifying the objectGUID of the NC that corresponds to oszNamingContext.

uuidSourceDsaObjGuid (16 bytes): A GUID structure, as defined in [MS-DTYP] section 2.3.4, specifying the objectGUID of the nTDSDSA object that corresponds to oszSourceDsaDN.

uuidSourceDsaInvocationID (16 bytes): A **GUID** structure, as defined in [MS-DTYP] section 2.3.4, specifying the **invocationId** used by the source server as of the last replication attempt.

uuidAsyncIntersiteTransportObjGuid (16 bytes): A **GUID** structure, as defined in [MS-DTYP] section 2.3.4, specifying the objectGUID of the intersite transport object that corresponds to oszAsyncIntersiteTransportDN.

usnLastObjChangeSynced (8 bytes): An **update sequence number (USN)** value, as defined in section 3.1.1.1.9, containing the USN of the last object update received.

usnAttributeFilter (8 bytes): A USN value, as defined in section 3.1.1.1.9, containing the usnLastObjChangeSynced value at the end of the last complete, successful **replication cycle**, or 0 if none.

ftimeLastSyncSuccess (8 bytes): A **FILETIME** structure that contains the date and time that the last successful replication cycle was completed from this source. All members of this structure are zero if the replication cycle has never been completed.

ftimeLastSyncAttempt (8 bytes): A **FILETIME** structure that contains the date and time of the last replication attempt from this source. All members of this structure are zero if the replication has never been attempted.

dwLastSyncResult (4 bytes): A 32-bit unsigned integer specifying a **Windows error code** associated with the last replication attempt from this source. Contains ERROR_SUCCESS if the last attempt was successful or replication was not attempted.

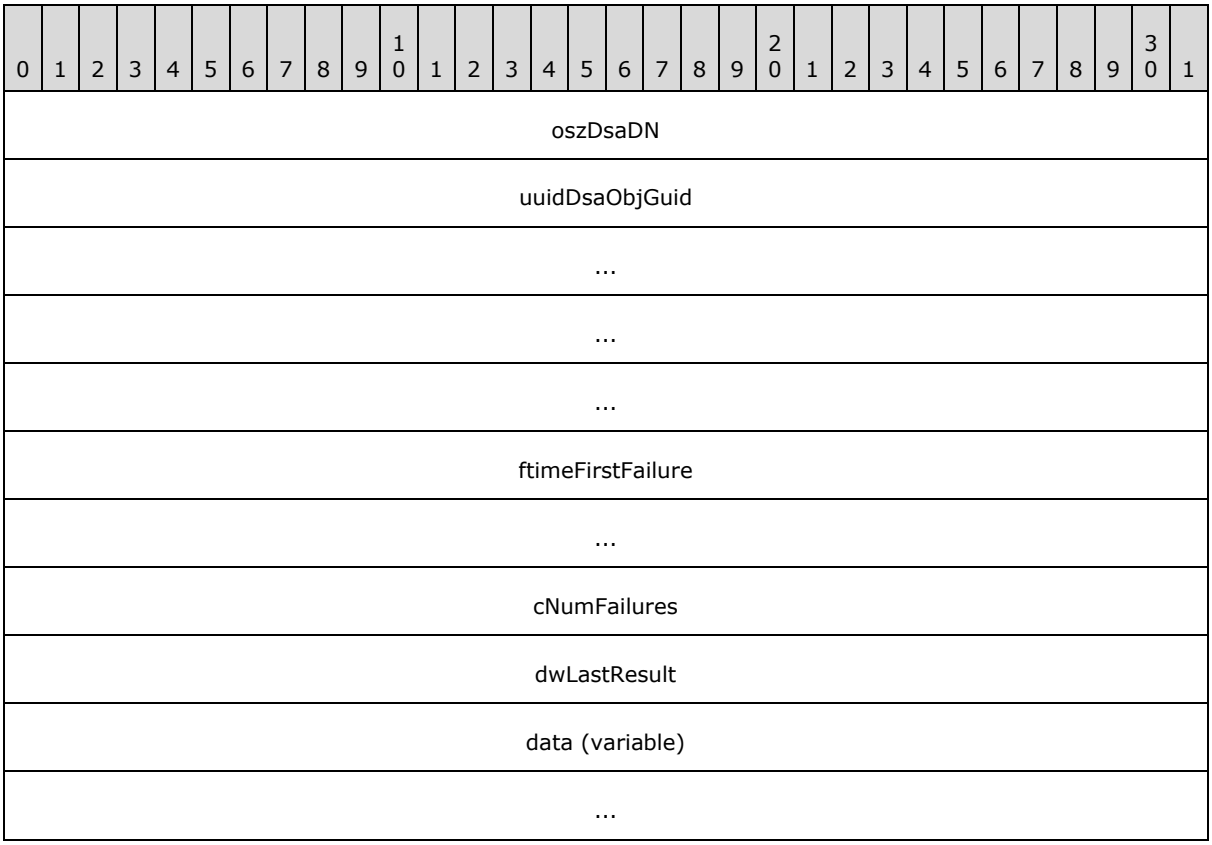
cNumConsecutiveSyncFailures (4 bytes): A 32-bit integer specifying the number of failed replication attempts that have been made from this source since the last successful replication attempt or since the source was added as a neighbor, if no previous attempt succeeded.

data (variable): This field contains all the null-terminated strings that are pointed to by the offset fields in the structure (oszNamingContext, oszSourceDsaDN, oszSourceDsaAddress, oszAsyncIntersiteTransportDN). The strings are packed into this field, and the offsets can be used to determine the start of each string.

All multibyte fields have little-endian byte ordering.

2.2.3 DS_REPL_KCC_DSA_FAILUREW_BLOB

The DS_REPL_KCC_DSA_FAILUREW_BLOB structure is a representation of a tuple from the *kCCFailedConnections* or *kCCFailedLinks* variables of a DC. This structure, retrieved using an LDAP search method, is an alternative representation of DS_REPL_KCC_DSA_FAILUREW, retrieved using the IDL_DRSGetReplInfo RPC method.



oszDsaDN (4 bytes): A 32-bit offset, in bytes, from the address of this structure to a null-terminated string that contains the DN of the nTDSDSA object of the source server.

uuidDsaObjGuid (16 bytes): A GUID structure, defined in [MS-DTYP] section 2.3.4, specifying the objectGUID of the object represented by the oszDsaDN member.

ftimeFirstFailure (8 bytes): A **FILETIME** structure, the content of which depends on the requested binary replication data.

Attribute requested	Meaning
msDS-ReplConnectionFailures	Contains the date and time that the first failure occurred when attempting to establish a replica link to the source server.
msDS-ReplLinkFailures	Contains the date and time that the first failure occurred when replicating from the source server.

cNumFailures (4 bytes): A 32-bit unsigned integer specifying the number of consecutive failures since the last successful replication.

dwLastResult (4 bytes): A 32-bit unsigned integer specifying the error code associated with the most recent failure, or ERROR_SUCCESS if no failures occurred.

data (variable): The **data** field contains the null-terminated string that contains the DN of the nTDSDSA object of the source server.

All multibyte fields have little-endian byte ordering.

2.2.4 DS_REPL_OPW_BLOB

The DS_REPL_OPW_BLOB structure is a representation of a tuple from the **replicationQueue** variable of a DC. This structure, retrieved using an LDAP search method, is an alternative representation of DS_REPL_OPW, retrieved using the IDL_DRSGetReplInfo RPC method.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ftimeEnqueued																															
...																															
ulSerialNumber																															
ulPriority																															
opType																															
ulOptions																															
oszNamingContext																															
oszDsaDN																															
oszDsaAddress																															

uuidNamingContextObjGuid
...
...
...
uuidDsaObjGuid
...
...
...
data (variable)
...

ftimeEnqueued (8 bytes): A **FILETIME** structure that contains the date and time that this operation was added to the queue.

ulSerialNumber (4 bytes): An unsigned integer specifying the identifier of the operation. The counter used to assign this identifier is volatile; it is reset during startup of a DC. Therefore, these identifiers are only unique between restarts of a DC.

ulPriority (4 bytes): An unsigned integer specifying the priority value of this operation. Tasks with a higher priority value are executed first. The priority is calculated by the server based on the type of operation and its parameters.

opType (4 bytes): Contains one of the following values that indicate the type of operation that this structure represents.

Operation	Value
DS_REPL_OP_TYPE_SYNC	0
DS_REPL_OP_TYPE_ADD	1
DS_REPL_OP_TYPE_DELETE	2
DS_REPL_OP_TYPE_MODIFY	3
DS_REPL_OP_TYPE_UPDATE_REFS	4

ulOptions (4 bytes): Zero or more bits from the Directory Replication Service (DRS) options defined in [MS-DRSR] section 5.41, the interpretation of which depends on the **OpType**.

oszNamingContext (4 bytes): Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated string that contains the DN of the NC associated with this operation (for example, the NC to be synchronized for DS_REPL_OP_TYPE_SYNC).

oszDsaDN (4 bytes): Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated string that contains the DN of the nTDSDSA object of the remote server corresponding to this operation. For example, the server from which to ask for changes for DS_REPL_OP_TYPE_SYNC. This can be NULL.

oszDsaAddress (4 bytes): Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated string that contains the transport-specific network address of the remote server associated with this operation. For example, the DNS or SMTP address of the server from which to ask for changes for DS_REPL_OP_TYPE_SYNC. This can be NULL.

uuidNamingContextObjGuid (16 bytes): A GUID structure, as defined in [MS-DTYP] section 2.3.4, specifying the objectGUID of the NC identified by oszNamingContext.

uuidDsaObjGuid (16 bytes): A GUID structure, as defined in [MS-DTYP] section 2.3.4, specifying the objectGUID of the directory system agent object identified by oszDsaDN.

data (variable): This field contains all the null-terminated strings that are pointed to by the offset fields in the structure (**oszNamingContext**, **oszDsaDN**, **oszDsaAddress**). The strings are packed into this field and the offsets can be used to determine the start of each string.

All multibyte fields have little-endian byte ordering.

2.2.5 DS_REPL_QUEUE_STATISTICSW_BLOB

The DS_REPL_QUEUE_STATISTICSW_BLOB structure contains the statistics related to the *replicationQueue* variable of a DC, returned by reading the msDS-ReplQueueStatistics (section 3.1.1.3.2.30) rootDSE attribute.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ftimeCurrentOpStarted																															
...																															
cNumPendingOps																															
ftimeOldestSync																															
...																															
ftimeOldestAdd																															
...																															
ftimeOldestMod																															
...																															

ftimeOldestDel
...
ftimeOldestUpdRefs
...

ftimeCurrentOpStarted (8 bytes): A **FILETIME** structure that contains the date and time that the currently running operation started.

cNumPendingOps (4 bytes): An unsigned integer specifying the number of currently pending operations.

ftimeOldestSync (8 bytes): A **FILETIME** structure that contains the date and time of the oldest synchronization operation.

ftimeOldestAdd (8 bytes): A **FILETIME** structure that contains the date and time of the oldest add operation.

ftimeOldestMod (8 bytes): A **FILETIME** structure that contains the date and time of the oldest modification operation.

ftimeOldestDel (8 bytes): A **FILETIME** structure that contains the date and time of the oldest delete operation.

ftimeOldestUpdRefs (8 bytes): A **FILETIME** structure that contains the date and time of the oldest reference update operation.

All multibyte fields have little-endian byte ordering.

2.2.6 DS_REPL_CURSOR_BLOB

The DS_REPL_CURSOR_BLOB is the packet representation of the ReplUpToDateVector type ([MS-DRSR] section 5.165) of an NC replica. This structure, retrieved using an LDAP search method, is an alternative representation of DS_REPL_CURSOR_3W, retrieved using the IDL_DRSGetReplInfo RPC method.

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
uuidSourceDsaInvocationID																															
...																															
...																															
...																															
usnAttributeFilter																															

...
fTimeLastSyncSuccess
...
oszSourceDsaDN
data (variable)
...

uuidSourceDsaInvocationID (16 bytes): A GUID structure, defined in [MS-DTYP] section 2.3.4, specifying the invocationId of the originating server to which the usnAttributeFilter corresponds.

usnAttributeFilter (8 bytes): A USN value, as defined in section 3.1.1.1.9, containing the maximum USN to which the destination server can indicate that it has recorded all changes originated by the given server at USNs less than or equal to this USN. This is used to **filter** changes at replication source servers that the destination server has already applied.

fTimeLastSyncSuccess (8 bytes): A **FILETIME** structure that contains the date and time of the last successful synchronization operation.

oszSourceDsaDN (4 bytes): Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated Unicode string. The string contains the distinguished name of the **directory service agent (DSA)** that corresponds to the source server to which this replication state data applies.

data (variable): This field contains the null-terminated string pointed to by the offset field in the structure (oszSourceDsaDN). The offset can be used to determine the start of the string.

All multibyte fields have little-endian byte ordering.

2.2.7 DS_REPL_ATTR_META_DATA_BLOB

The DS_REPL_ATTR_META_DATA_BLOB packet is a representation of a *stamp* variable (of type **AttributeStamp**) of an attribute. This structure, retrieved using an LDAP search method, is an alternative representation of DS_REPL_ATTR_META_DATA_2, retrieved using the IDL_DRSGetReplInfo RPC method.

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

...
uuidLastOriginatingDsaInvocationID
...
...
...
usnOriginatingChange
...
usnLocalChange
...
oszLastOriginatingDsaDN
data (variable)
...

oszAttributeName (4 bytes): Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated Unicode string that contains the LDAP display name of the attribute corresponding to this metadata.

dwVersion (4 bytes): Contains the *dwVersion* of this attribute's AttributeStamp, as specified in section 3.1.1.1.9.

ftimeLastOriginatingChange (8 bytes): Contains the *timeChanged* of this attribute's AttributeStamp, as specified in section 3.1.1.1.9.

uuidLastOriginatingDsaInvocationID (16 bytes): Contains the *uuidOriginating* of this attribute's AttributeStamp, as specified in section 3.1.1.1.9.

usnOriginatingChange (8 bytes): Contains the *usnOriginating* of this attribute's AttributeStamp, as specified in section 3.1.1.1.9.

usnLocalChange (8 bytes): A USN value, defined in section 3.1.1.1.9, specifying the USN on the destination server (the server from which the metadata information is retrieved) at which the last change to this attribute was applied. This value typically is different on all servers.

oszLastOriginatingDsaDN (4 bytes): Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated Unicode string that contains the DN of the nTDSDSA object of the server that originated the last replication.

data (variable): This field contains all the null-terminated strings that are pointed to by the offset fields in the structure (oszAttributeName, oszLastOriginatingDsaDN). The strings are packed into this field, and the offsets can be used to determine the start of each string.

All multibyte fields have little-endian byte ordering.

2.2.8 DS_REPL_VALUE_META_DATA_BLOB

The DS_REPL_VALUE_META_DATA_BLOB packet is a representation of a *stamp* variable (of type **LinkValueStamp**) of a link value. This structure, retrieved using an LDAP search method, is an alternative representation of DS_REPL_VALUE_META_DATA_2, retrieved using the IDL_DRSGetReplInfo RPC method.

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
oszAttributeName																															
oszObjectDn																															
cbData																															
pbData																															
ftimeDeleted																															
...																															
ftimeCreated																															
...																															
dwVersion																															
ftimeLastOriginatingChange																															
...																															
uuidLastOriginatingDsaInvocationID																															
...																															
...																															
...																															
usnOriginatingChange																															
...																															
usnLocalChange																															

...
oszLastOriginatingDsaDN
data (variable)
...

oszAttributeName (4 bytes): Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated Unicode string that contains the LDAP display name of the attribute corresponding to this metadata.

oszObjectDn (4 bytes): Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated Unicode string that contains the DN of the object that this attribute belongs to.

cbData (4 bytes): Contains the number of bytes in the pbData array.

pbData (4 bytes): Contains a 32-bit offset, in bytes, from the address of this structure to a buffer that contains the attribute replication metadata. The **cbData** member contains the length, in bytes, of this buffer.

ftimeDeleted (8 bytes): Contains the *timeDeleted* of this link value's LinkValueStamp, as specified in section 3.1.1.1.9.

ftimeCreated (8 bytes): Contains the *timeCreated* of this link value's LinkValueStamp, as specified in section 3.1.1.1.9.

dwVersion (4 bytes): Contains the *dwVersion* of this link value's LinkValueStamp, as specified in section 3.1.1.1.9.

ftimeLastOriginatingChange (8 bytes): Contains the *timeChanged* of this link value's LinkValueStamp, as specified in section 3.1.1.1.9.

uuidLastOriginatingDsaInvocationID (16 bytes): Contains the *uuidOriginating* of this link value's LinkValueStamp, as specified in section 3.1.1.1.9.

usnOriginatingChange (8 bytes): Contains the *usnOriginating* of this link value's LinkValueStamp, as specified in section 3.1.1.1.9.

usnLocalChange (8 bytes): Specifies the USN, as found on the server from which the metadata information is being retrieved, at which the last change to this attribute was applied. This value is typically different on all servers.

oszLastOriginatingDsaDN (4 bytes): Contains a 32-bit offset, in bytes, from the address of this structure to a null-terminated Unicode string that contains the DN of the nTDSDSA object of the server that originated the last replication.

data (variable): This field contains all the null-terminated strings that are pointed to by the offset fields in the structure (oszAttributeName, oszObjectDn, oszLastOriginatingDsaDN) and the buffer pointed to by **pbData**. The strings and buffers are packed into this field (aligned at 32-bit boundaries), and the offsets can be used to determine the start of each string.

All multibyte fields have little-endian byte ordering.

2.2.9 Search Flags

The following table defines the valid search flags used on attributes, as specified in section 3.1.1.2.3. The flags are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	SE	BO	XL	RO	NV	CF	ST	TP	CP	PR	AR	PI	IX

X: Unused. Must be zero and ignored.

IX (fATTINDEX, 0x00000001): Specifies a hint to the DC to create an index for the attribute.

PI (fPDNTATTINDEX, 0x00000002): Specifies a hint to the DC to create an index for the container and the attribute.

AR(fANR, 0x00000004): Specifies that the attribute is a member of the **ambiguous name resolution (ANR)** set.

PR (fPRESERVEONDELETE, 0x00000008): Specifies that the attribute MUST be preserved on objects after deletion of the object (that is, when the object is transformed to a **tombstone**, **deleted-object**, or **recycled-object**). This flag is ignored on **link attributes**, objectCategory, and sAMAccountType.

CP (fCOPY, 0x00000010): Specifies a hint to LDAP clients that the attribute is intended to be copied when copying the object. This flag is not interpreted by the server.

TP (fTUPLEINDEX, 0x00000020): Specifies a hint for the DC to create a tuple index for the attribute. This will affect the performance of searches where the wildcard appears at the front of the search string.

ST (fSUBTREEATTINDEX, 0x00000040): Specifies a hint for the DC to create subtree index for a **Virtual List View (VLV) search**.

CF (fCONFIDENTIAL, 0x00000080): Specifies that the attribute is confidential. An extended access check (section 3.1.1.4.4) is required.

NV (fNEVERVALUEAUDIT, 0x00000100): Specifies that auditing of changes to individual values contained in this attribute MUST NOT be performed. Auditing is outside of the state model.

RO (fRODCFilteredAttribute, 0x00000200): Specifies that the attribute is a member of the **filtered attribute set**.

XL (fEXTENDEDLINKTRACKING, 0x00000400): Specifies a hint to the DC to perform additional implementation-specific, nonvisible tracking of link values. The behavior of this hint is outside the state model.

BO (fBASEONLY, 0x00000800): Specifies that the attribute is not to be returned by search operations that are not scoped to a single object. Read operations that would otherwise return an attribute that has this search flag set instead fail with *operationsError* / *ERROR_DS_NON_BASE_SEARCH*.

SE (fPARTITIONSECRET, 0x00001000): Specifies that the attribute is a partition secret. An extended **access check** is required.

Flags that specify "hints" only direct the server to create certain indices that affect the system performance. The effects of these flags are outside the state model. Implementations are permitted to ignore these flags.

2.2.10 System Flags

The following table defines the valid system flags used on **directory objects**. The flags are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
DD	AR	AM	AL	DR	DM	DE	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	RD	BS	OP	CS	PS	NR

X: Unused. Must be zero and ignored.

NR (FLAG_ATTR_NOT_REPLICATED or FLAG_CR_NTDS_NC, 0x00000001): When used on an attributeSchema object, it specifies that this attribute is not replicated. If it is used on a **crossRef object**, it specifies that the NC that the crossRef is for is an Active Directory NC.

PS (FLAG_ATTR_REQ_PARTIAL_SET_MEMBER or FLAG_CR_NTDS_DOMAIN, 0x00000002): When used on an attributeSchema object, it specifies that the attribute is a member of a **partial attribute set (PAS)**. If used on a crossRef object, it specifies that the NC is a **domain NC**.

CS (FLAG_ATTR_IS_CONSTRUCTED or FLAG_CR_NTDS_NOT_GC_REPLICATED, 0x00000004): When used on an attributeSchema object, this flag specifies that the attribute is a **constructed attribute**. If used on a crossRef object, it specifies that the NC is not to be replicated to **GCs**.

OP (FLAG_ATTR_IS_OPERATIONAL, 0x00000008): Only used on an attributeSchema object. It specifies that the attribute is an **operational attribute**.

BS (FLAG_SCHEMA_BASE_OBJECT, 0x00000010): Only used on attributeSchema and classSchema objects. It specifies that this attribute or class is part of the base schema. Modifications to base schema objects are specially restricted.

RD (FLAG_ATTR_IS_RDN, 0x00000020): Only used on an attributeSchema object. It specifies that this attribute can be used as an **RDN attribute**.

DE (FLAG_DISALLOW_MOVE_ON_DELETE, 0x02000000): Specifies that the object does not move to the Deleted Objects container when the object is deleted.

DM (FLAG_DOMAIN_DISALLOW_MOVE, 0x04000000): Specifies that if the object is in a domain NC, the object cannot be moved.

DR (FLAG_DOMAIN_DISALLOW_RENAME, 0x08000000): Specifies that if the object is in a domain NC, the object cannot be renamed.

AL (FLAG_CONFIG_ALLOW_LIMITED_MOVE, 0x10000000): Specifies that if the object is in the **config NC**, the object can be moved, with restrictions.

AM (FLAG_CONFIG_ALLOW_MOVE, 0x20000000): Specifies that if the object is in the config NC, the object can be moved.

AR (FLAG_CONFIG_ALLOW_RENAME, 0x40000000): Specifies that if the object is in the config NC, the object can be renamed.

DD (FLAG_DISALLOW_DELETE, 0x80000000): Specifies that the object cannot be deleted.

2.2.11 schemaFlagsEx Flags

The following table defines the valid schemaFlagsEx flags that are used on attributes, as specified in section 3.1.1.2.3. The flags are presented in big-endian byte order.

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

X: Unused. MUST be zero and ignored.

CR (FLAG_ATTR_IS_CRITICAL, 0x00000001): Specifies that the attribute is not a member of the filtered attribute set even if the fRODCFilteredAttribute flag is set. For more information, see sections 3.1.1.2.3 and 3.1.1.2.3.5.

2.2.12 Group Type Flags

Constants for defining **group** type. These constants define the values that are used in the groupType attribute.

Symbolic name	Value
GROUP_TYPE_BUILTIN_LOCAL_GROUP	0x00000001
GROUP_TYPE_ACCOUNT_GROUP	0x00000002
GROUP_TYPE_RESOURCE_GROUP	0x00000004
GROUP_TYPE_UNIVERSAL_GROUP	0x00000008
GROUP_TYPE_APP_BASIC_GROUP	0x00000010
GROUP_TYPE_APP_QUERY_GROUP	0x00000020
GROUP_TYPE_SECURITY_ENABLED	0x80000000

GROUP_TYPE_BUILTIN_LOCAL_GROUP: Specifies a group that is created by the system.

GROUP_TYPE_ACCOUNT_GROUP: Specifies a **global group**.

GROUP_TYPE_RESOURCE_GROUP: Specifies a **domain local group**.

GROUP_TYPE_UNIVERSAL_GROUP: Specifies a **universal group**.

GROUP_TYPE_APP_BASIC_GROUP: Groups of this type are not used by Active Directory. This constant is included in this document because the value of this constant is used by Active Directory in processing the groupType attribute (see section 3.1.1.5.4.2.2).

GROUP_TYPE_APP_QUERY_GROUP: Groups of this type are not used by Active Directory. This constant is included in this document because the value of this constant is used by Active Directory in processing the groupType attribute.

GROUP_TYPE_SECURITY_ENABLED: Specifies a **security-enabled group**.

The flags GROUP_TYPE_BUILTIN_LOCAL_GROUP, GROUP_TYPE_ACCOUNT_GROUP, GROUP_TYPE_RESOURCE_GROUP, GROUP_TYPE_UNIVERSAL_GROUP, GROUP_TYPE_APP_BASIC_GROUP, and GROUP_TYPE_APP_QUERY_GROUP are mutually exclusive, and only one value must be set. The flag GROUP_TYPE_SECURITY_ENABLED can be combined using a bitwise OR with flags GROUP_TYPE_BUILTIN_LOCAL_GROUP, GROUP_TYPE_ACCOUNT_GROUP, GROUP_TYPE_RESOURCE_GROUP, and GROUP_TYPE_UNIVERSAL_GROUP.

2.2.13 Security Privilege Flags

Constants for defining security **privilege**.

Symbolic name	Value
SE_SECURITY_PRIVILEGE	0x00000008
SE_TAKE_OWNERSHIP_PRIVILEGE	0x00000009
SE_RESTORE_PRIVILEGE	0x00000012
SE_DEBUG_PRIVILEGE	0x00000014
SE_ENABLE_DELEGATION_PRIVILEGE	0x0000001B

SE_SECURITY_PRIVILEGE: Specifies the privilege to manage auditing and the security log.

SE_TAKE_OWNERSHIP_PRIVILEGE: Specifies the privilege to take ownership of objects. Possession of this privilege overrides the **DACL** on an object and gives the possessor implicit RIGHT_WRITE_OWNER access.

SE_RESTORE_PRIVILEGE: Specifies the privilege to restore objects.

SE_DEBUG_PRIVILEGE: Specifies the privilege to debug the system.

SE_ENABLE_DELEGATION_PRIVILEGE: Specifies the privilege to enable accounts to be trusted for delegation.

2.2.14 Domain RID Values

Constants for defining **domain relative identifiers (RIDs)**.

Symbolic name	Value
DOMAIN_USER_RID_ADMIN	0x000001F4
DOMAIN_USER_RID_KRBTGT	0x000001F6

Symbolic name	Value
DOMAIN_GROUP_RID_ADMINS	0x00000200
DOMAIN_GROUP_RID_CONTROLLERS	0x00000204
DOMAIN_GROUP_RID_SCHEMA_ADMINS	0x00000206
DOMAIN_GROUP_RID_ENTERPRISE_ADMINS	0x00000207
DOMAIN_GROUP_RID_READONLY_CONTROLLERS	0x00000209
DOMAIN_ALIAS_RID_ADMINS	0x00000220
DOMAIN_ALIAS_RID_ACCOUNT_OPS	0x00000224
DOMAIN_ALIAS_RID_SYSTEM_OPS	0x00000225
DOMAIN_ALIAS_RID_PRINT_OPS	0x00000226
DOMAIN_ALIAS_RID_BACKUP_OPS	0x00000227
DOMAIN_ALIAS_RID_REPLICATOR	0x00000228

DOMAIN_USER_RID_ADMIN: The administrative user account in a domain.

DOMAIN_USER_RID_KRBTGT: The Kerberos **ticket-granting ticket (TGT)** account in a domain.

DOMAIN_GROUP_RID_ADMINS: The domain administrators' group.

DOMAIN_GROUP_RID_CONTROLLERS: The DCs' group. All DCs in the domain are members of the group.

DOMAIN_GROUP_RID_SCHEMA_ADMINS: The schema administrators' group. Members of this group can modify the Active Directory schema.

DOMAIN_GROUP_RID_ENTERPRISE_ADMINS: The enterprise administrators' group. Members of this group have full access to all domains in the Active Directory **forest**. Enterprise administrators are responsible for forest-level operations, such as adding or removing new domains.

DOMAIN_GROUP_RID_READONLY_CONTROLLERS: The read-only domain controllers' group. All read-only DCs in the domain are members of this group.

DOMAIN_ALIAS_RID_ADMINS: The administrators' group in the **built-in domain**.

DOMAIN_ALIAS_RID_ACCOUNT_OPS: A group that permits control over nonadministrator accounts.

DOMAIN_ALIAS_RID_SYSTEM_OPS: A group that performs system administrative functions, not including security functions. It establishes network shares, controls printers, unlocks workstations, and performs other operations.

DOMAIN_ALIAS_RID_PRINT_OPS: A group that controls printers and print queues.

DOMAIN_ALIAS_RID_BACKUP_OPS: A group that is used for controlling assignment of file backup and restoring user rights.

DOMAIN_ALIAS_RID_REPLICATOR: A group responsible for copying security databases to the Windows NT operating system backup controllers.

2.2.15 userAccountControl Bits

Bit flags describing various qualities of a security account. The flags are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	PS	NA	TA	PE	DR	DK	ND	TD	SR	X	DP	X	X	ST	WT	ID	X	N	X	ET	CC	NR	L	HR	X	D	X

X: Unused. Must be zero and ignored.

D (ADS_UF_ACCOUNT_DISABLE, 0x00000002): Specifies that the account is not enabled for **authentication**.

HR (ADS_UF_HOMEDIR_REQUIRED, 0x00000008): Specifies that the homeDirectory attribute is required.

L (ADS_UF_LOCKOUT, 0x00000010): Specifies that the account is temporarily locked out.

NR (ADS_UF_PASSWD_NOTREQD, 0x00000020): Specifies that the password-length policy, as specified in [MS-SAMR] section 3.1.1.8.1, does not apply to this user.

CC (ADS_UF_PASSWD_CANT_CHANGE, 0x00000040): Specifies that the user cannot change his or her password.

ET (ADS_UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED, 0x00000080): Specifies that the cleartext password is to be persisted.

N (ADS_UF_NORMAL_ACCOUNT, 0x00000200): Specifies that the account is the default account type that represents a typical user.

ID (ADS_UF_INTERDOMAIN_TRUST_ACCOUNT, 0x00000800): Specifies that the account is for a domain-to-domain **trust**.

WT (ADS_UF_WORKSTATION_TRUST_ACCOUNT, 0x00001000): Specifies that the account is a computer account for a computer that is a member of this domain.

ST (ADS_UF_SERVER_TRUST_ACCOUNT, 0x00002000): Specifies that the account is a computer account for a DC.

DP (ADS_UF_DONT_EXPIRE_PASSWD, 0x00010000): Specifies that the password does not expire for the account.

SR (ADS_UF_SMARTCARD_REQUIRED, 0x00040000): Specifies that a smart card is required to log in to the account.

TD (ADS_UF_TRUSTED_FOR_DELEGATION, 0x00080000): Used by the Kerberos protocol. This bit indicates that the "OK as Delegate" ticket flag, as described in [RFC4120] section 2.8, MUST be set.

ND (ADS_UF_NOT_DELEGATED, 0x00100000): Used by the Kerberos protocol. This bit indicates that the ticket-granting tickets (TGTs) of this account and the service tickets obtained by this account are not marked as forwardable or proxiable when the forwardable or proxiable ticket flags are requested. For more information, see [RFC4120].

DK (ADS_UF_USE_DES_KEY_ONLY, 0x00200000): Used by the Kerberos protocol. This bit indicates that only des-cbc-md5 or des-cbc-crc keys, as defined in [RFC3961], are used in the Kerberos protocols for this account.

DR (ADS_UF_DONT_REQUIRE_PREAUTH, 0x00400000): Used by the Kerberos protocol. This bit indicates that the account is not required to present valid preauthentication data, as described in [RFC4120] section 7.5.2.

PE (ADS_UF_PASSWORD_EXPIRED, 0x00800000): Specifies that the password age on the user has exceeded the maximum password age policy.

TA (ADS_UF_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION, 0x01000000): Used by the Kerberos protocol. When set, this bit indicates that the account (when running as a service) obtains an S4U2self service ticket (as specified in [MS-SFU]) with the forwardable flag set. If this bit is cleared, the forwardable flag is not set in the S4U2self service ticket.

NA (ADS_UF_NO_AUTH_DATA_REQUIRED, 0x02000000): Used by the Kerberos protocol. This bit indicates that when the Key Distribution Center (KDC) is issuing a service ticket for this account, the Privilege Attribute Certificate (PAC) MUST NOT be included. For more information, see [RFC4120].

PS (ADS_UF_PARTIAL_SECRETS_ACCOUNT, 0x04000000): Specifies that the account is a computer account for a **read-only domain controller (RODC)**. If this bit is set, the ADS_UF_WORKSTATION_TRUST_ACCOUNT must also be set. This flag is only interpreted by a DC whose DC functional level is DS_BEHAVIOR_WIN2008 or greater.

2.2.16 Optional Feature Values

Constants for defining behaviors of **optional features**.

Symbolic name	Value
FOREST_OPTIONAL_FEATURE	0x00000001
DOMAIN_OPTIONAL_FEATURE	0x00000002
DISABLABLE_OPTIONAL_FEATURE	0x00000004
SERVER_OPTIONAL_FEATURE	0x00000008

FOREST_OPTIONAL_FEATURE: Specifies that the scope of the optional feature is forest-wide.

DOMAIN_OPTIONAL_FEATURE: Specifies that the scope of the optional feature is domain-wide.

DISABLABLE_OPTIONAL_FEATURE: Specifies that the optional feature can be disabled.

SERVER_OPTIONAL_FEATURE: Specifies that the scope of the optional feature is server-wide.

For more information, see section 3.1.1.9.

2.2.17 Claims Wire Structures

This section defines the structures related to **claims** using Interface Definition Language (IDL) format. Refer to the term **marshal** in [MS-GLOS] for information on converting these structures into the appropriate wire format.

The following figure illustrates the nesting of various larger claims structures for descriptive reference purposes.

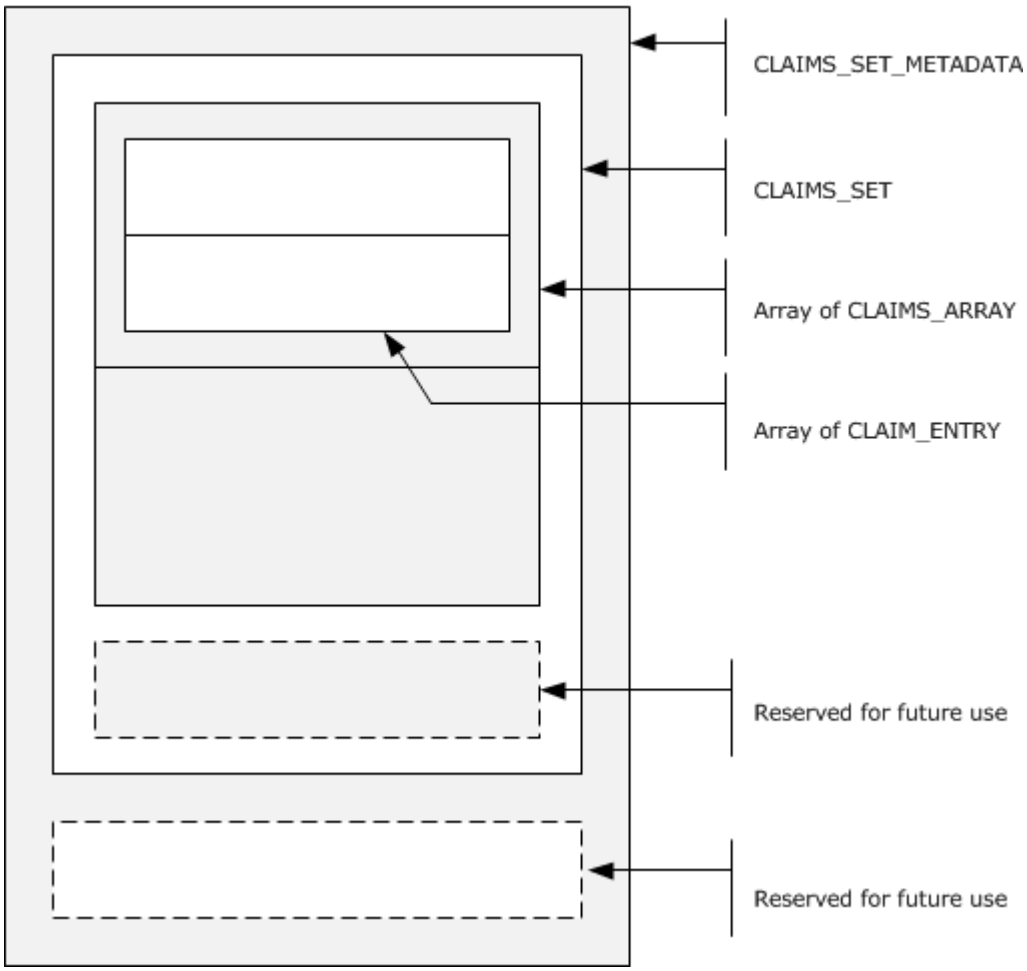


Figure 2: Nesting of claims structures

2.2.17.1 CLAIM_ID

The CLAIM_ID type is a null-terminated UTF-16 string used for typing claim IDs.

```
typedef [string] wchar_t* CLAIM_ID;

typedef [string] wchar_t** PCLAIM_ID;
```

2.2.17.2 CLAIM_TYPE

The CLAIM_TYPE enumeration enumerates various value types of a claim.

```
typedef  enum _CLAIM_TYPE
{
  CLAIM_TYPE_INT64 = 1,
  CLAIM_TYPE_UINT64 = 2,
  CLAIM_TYPE_STRING = 3,
  CLAIM_TYPE_BOOLEAN = 6
} CLAIM_TYPE,
 *PCLAIM_TYPE;
```

CLAIM_TYPE_INT64: The value type of the claim is **LONG64**.

CLAIM_TYPE_UINT64: The value type of the claim is **ULONG64**.

CLAIM_TYPE_STRING: The value type of the claim is a null-terminated string of Unicode characters.

CLAIM_TYPE_BOOLEAN: The value type of the claim is **ULONG64**; a value is set to 1 to specify TRUE, or 0 to specify FALSE.

2.2.17.3 CLAIMS_SOURCE_TYPE

The CLAIMS_SOURCE_TYPE enumeration specifies the source of the claims.

```
typedef  enum _CLAIMS_SOURCE_TYPE
{
  CLAIMS_SOURCE_TYPE_AD = 1,
  CLAIMS_SOURCE_TYPE_CERTIFICATE
} CLAIMS_SOURCE_TYPE;
```

Note No semantics should be attached to these values other than those specified in section 3.

2.2.17.4 CLAIMS_COMPRESSION_FORMAT

The CLAIMS_COMPRESSION_FORMAT enumeration specifies the source of the compression algorithm that is used for encoding claims in a **CLAIMS_SET_METADATA** structure.

```
typedef  enum _CLAIMS_COMPRESSION_FORMAT
{
  COMPRESSION_FORMAT_NONE = 0,
  COMPRESSION_FORMAT_LZNT1 = 2,
  COMPRESSION_FORMAT_XPRESS = 3,
  COMPRESSION_FORMAT_XPRESS_HUFF = 4
} CLAIMS_COMPRESSION_FORMAT;
```

COMPRESSION_FORMAT_NONE: No compression.

COMPRESSION_FORMAT_LZNT1: The LZNT1 compression algorithm is used. For more information, see [MS-XCA] section 2.5.

COMPRESSION_FORMAT_XPRESS: The Xpress LZ77 compression algorithm is used. For more information, see [MS-XCA] sections 2.3 and 2.4.

COMPRESSION_FORMAT_XPRESS_HUFF: The Xpress LZ77+Huffman compression algorithm is used. For more information, see [MS-XCA] sections 2.1 and 2.2.

2.2.17.5 CLAIM_ENTRY

The CLAIM_ENTRY structure specifies a single claim.

```
typedef struct _CLAIM_ENTRY {
  CLAIM_ID Id;
  CLAIM_TYPE Type;
  [switch_is(Type), switch_type(CLAIM_TYPE)]
    union {
    [case(CLAIM_TYPE_INT64)]
      struct {
      [range(1, 10*1024*1024)] ULONG ValueCount;
      [size_is(ValueCount)] LONG64* Int64Values;
    };
    [case(CLAIM_TYPE_UINT64)]
      struct {
      [range(1, 10*1024*1024)] ULONG ValueCount;
      [size_is(ValueCount)] ULONG64* Uint64Values;
    };
    [case(CLAIM_TYPE_STRING)]
      struct {
      [range(1, 10*1024*1024)] ULONG ValueCount;
      [size_is(ValueCount), string] LPWSTR* StringValues;
    };
    [case(CLAIM_TYPE_BOOLEAN)]
      struct {
      [range(1, 10*1024*1024)] ULONG ValueCount;
      [size_is(ValueCount)] ULONG64* BooleanValues;
    };
    [default]      ;
  } Values;
} CLAIM_ENTRY,
 *PCLAIM_ENTRY;
```

Id: Specifies the claim identifier.

Type: Specifies the type of the data in the **Values** union. Refer to section 2.2.17.2 for allowed values and their interpretation.

Values: A union of arrays of the various types of claim values that a CLAIM_ENTRY can contain. The actual type of the elements is specified by the **Type** member.

ValueCount: Specifies the number of array elements in the **Int64Values** member.

Int64Values: An array of **LONG64** values of the claim. The array has **ValueCount** elements.

ValueCount: Specifies the number of array elements in the **Uint64Values** member.

Uint64Values: An array of **ULONG64** values of the claim. The array has **ValueCount** elements.

ValueCount: Specifies the number of array elements in the **StringValues** member.

StringValues: An array of null-terminated, Unicode string values of the claim. The array has **ValueCount** elements.

ValueCount: Specifies the number of array elements in the **BooleanValues** member.

BooleanValues: An array of **ULONG64** values of the claim. The array has **ValueCount** elements.

2.2.17.6 CLAIMS_ARRAY

The **CLAIMS_ARRAY** structure specifies an array of CLAIM_ENTRY structures and the associated claims source type.

```
typedef struct _CLAIMS_ARRAY {
  CLAIMS_SOURCE_TYPE usClaimsSourceType;
  ULONG ulClaimsCount;
  [size_is(ulClaimsCount)] PCLAIM_ENTRY ClaimEntries;
} CLAIMS_ARRAY,
 *PCLAIMS_ARRAY;
```

usClaimsSourceType: Specifies the source of the claims.

ulClaimsCount: Specifies the number of **CLAIM_ENTRY** elements in the **ClaimEntries** member of this structure.

ClaimEntries: An array that contains **ulClaimsCount** number of **CLAIM_ENTRY** elements.

2.2.17.7 CLAIMS_SET

The **CLAIMS_SET** structure specifies **CLAIMS_ARRAY** structures, each from a different claims source.

```
typedef struct _CLAIMS_SET {
  ULONG ulClaimsArrayCount;
  [size_is(ulClaimsArrayCount)] PCLAIMS_ARRAY ClaimsArrays;
  USHORT usReservedType;
  ULONG ulReservedFieldSize;
  [size_is(ulReservedFieldSize)] BYTE* ReservedField;
} CLAIMS_SET,
 *PCLAIMS_SET;
```

ulClaimsArrayCount: Specifies the number of **CLAIMS_ARRAY** elements that are in the **ClaimsArrays** member. This field MUST be greater than or equal to 1.

ClaimsArrays: An array containing **ulClaimsArrayCount** number of **CLAIMS_ARRAY** structures.

usReservedType: This field is not used.

ulReservedFieldSize: Specifies the length, in bytes, of the **ReservedField** member.

ReservedField: A byte array containing **ulReservedFieldSize** bytes.

2.2.17.8 CLAIMS_SET_METADATA

The **CLAIMS_SET_METADATA** structure specifies an encoded CLAIMS_SET structure with information about the encoding.

```
typedef struct _CLAIMS_SET_METADATA {
  ULONG ulClaimsSetSize;
  [size_is(ulClaimsSetSize)] BYTE* ClaimsSet;
  CLAIMS_COMPRESSION_FORMAT usCompressionFormat;
  ULONG ulUncompressedClaimsSetSize;
  USHORT usReservedType;
  ULONG ulReservedFieldSize;
  [size_is(ulReservedFieldSize)] BYTE* ReservedField;
} CLAIMS_SET_METADATA,
 *PCLAIMS_SET_METADATA;
```

ulClaimsSetSize: Contains the size, in bytes, of the **ClaimsSet** member.

ClaimsSet: A byte array of length **ulClaimsSetSize** bytes. This field contains a **CLAIMS_SET** structure that is encoded as described in section 3.1.1.11.2.5.

usCompressionFormat: Specifies the compression algorithm used for encoding a **CLAIMS_SET** structure, as specified in section 3.1.1.11.2.5.

ulUncompressedClaimsSetSize: Specifies the size of the partially encoded **CLAIMS_SET** structure before compression, the fully encoded version of which is stored in the **ClaimsSet** member.

usReservedType: The server MUST set this member to 0. The client MUST ignore this member.

ulReservedFieldSize: Specifies the size, in bytes, of the **ReservedField** member.

ReservedField: A byte array containing **ulReservedFieldSize** elements.

2.2.17.9 CLAIMS_BLOB

The **CLAIMS_BLOB** structure is generated from a **CLAIMS_SET** structure, as specified in section 3.1.1.11.2.5.

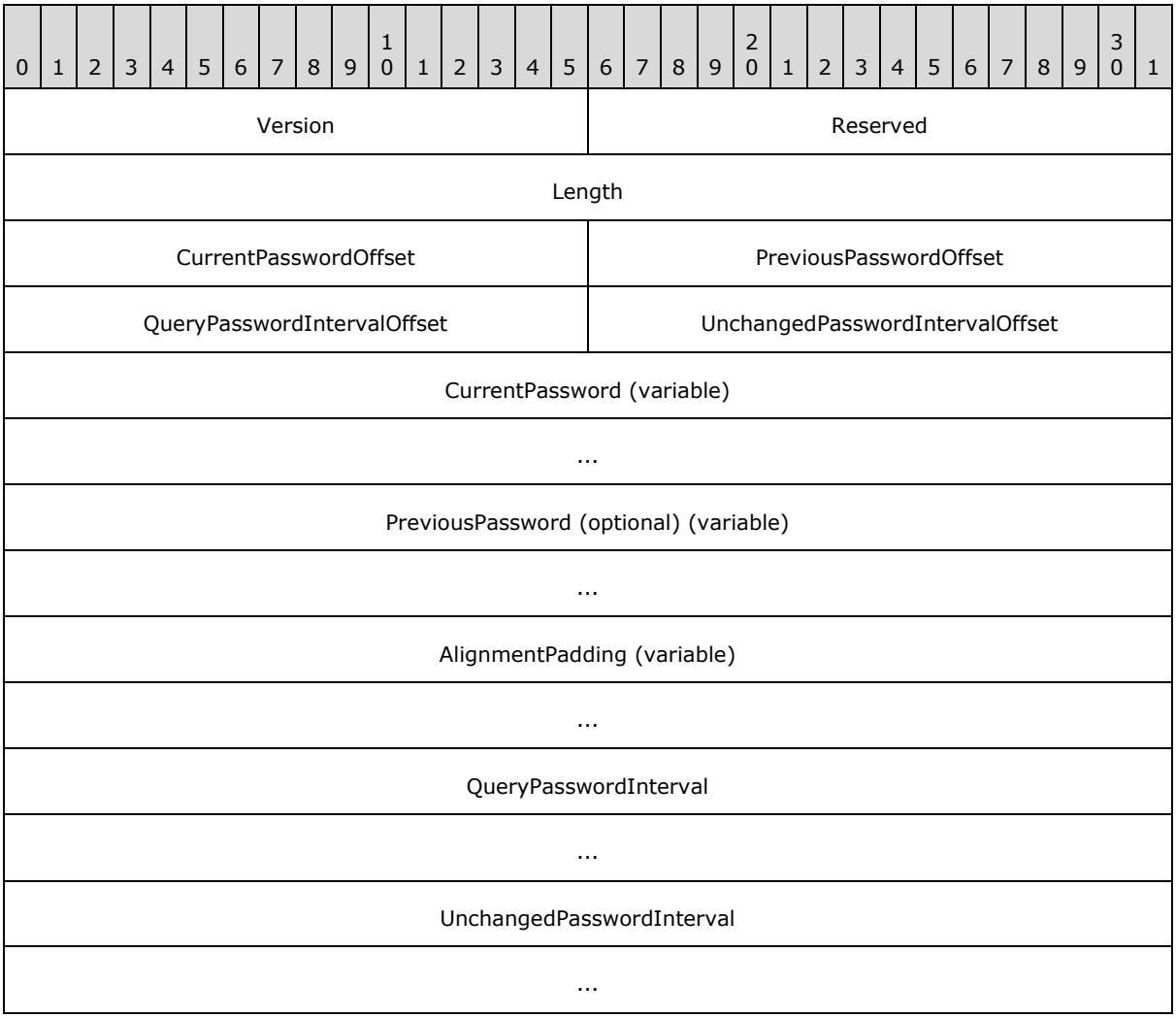
```
typedef struct CLAIMS_BLOB {
  ULONG ulBlobSizeinBytes;
  [size_is(dwBlobSizeinBytes)] PVOID EncodedBlob;
} CLAIMS_BLOB,
 *PCLAIMS_BLOB;
```

ulBlobSizeinBytes: The size of the **EncodedBlob** member, in bytes.

EncodedBlob: A byte array of length **ulBlobSizeinBytes** bytes that contains an encoded **CLAIMS_SET_METADATA** structure.

2.2.18 MSDS-MANAGEDPASSWORD_BLOB

The **MSDS-MANAGEDPASSWORD_BLOB** structure is a representation of a group-managed service account's password information. This structure is returned as the msDS-ManagedPassword (section 3.1.1.4.5.39) constructed attribute.



Version (2 bytes): A 16-bit unsigned integer that defines the version of the msDS-ManagedPassword **binary large object (BLOB)**. The **Version** field MUST be set to 0x0001.

Reserved (2 bytes): A 16-bit unsigned integer that MUST be set to 0x0000.

Length (4 bytes): A 32-bit unsigned integer that specifies the length, in bytes, of the msDS-ManagedPassword BLOB.

CurrentPasswordOffset (2 bytes): A 16-bit offset, in bytes, from the beginning of this structure to the **CurrentPassword** field. The **CurrentPasswordOffset** field MUST NOT be set to 0x0000.

PreviousPasswordOffset (2 bytes): A 16-bit offset, in bytes, from the beginning of this structure to the **PreviousPassword** field. If this field is set to 0x0000, then the account has no previous password.

QueryPasswordIntervalOffset (2 bytes): A 16-bit offset, in bytes, from the beginning of this structure to the **QueryPasswordInterval** field.

UnchangedPasswordIntervalOffset (2 bytes): A 16-bit offset, in bytes, from the beginning of this structure to the **UnchangedPasswordInterval** field.

CurrentPassword (variable): A null-terminated **WCHAR** string containing the cleartext current password for the account.

PreviousPassword (optional) (variable): A null-terminated **WCHAR** string containing the cleartext previous password for the account. If **PreviousPasswordOffset** is 0x0000, then this field MUST be absent.

AlignmentPadding (variable): A padding field used to align the **QueryPasswordInterval** field to a 64-bit boundary. This field is ignored by the receiver. This field SHOULD set to zero and MUST be ignored on receipt.

QueryPasswordInterval (8 bytes): A 64-bit unsigned integer containing the length of time, in units of 10^(-7) seconds, after which the receiver should requery the password. The **QueryPasswordInterval** field MUST be placed on a 64-bit boundary.

UnchangedPasswordInterval (8 bytes): A 64-bit unsigned integer containing the length of time, in units of 10^(-7) seconds, before which password queries will always return this password value. The **UnchangedPasswordInterval** field MUST be placed on a 64-bit boundary.

3 Details

The following sections specify details of the abstract data model and directory operations for Active Directory.

When an **LDAP** operation results in an error, the error is expressed in this document in the form:

- *LDAP error* / *Extended error code*

Where the *Extended error code* is either a *Windows error code* or the literal string "<unrestricted>".

The *LDAP error* is specified in the **resultCode** field of an LDAP response. See [RFC2251] section 4.1.10 for the specification of **resultCode** in an LDAP response. See section 3.1.1.3.1.9 for the specification of *Extended error codes* in an LDAP response.

3.1 Common Details

3.1.1 Abstract Data Model

Sections 3.1.1.1 and 3.1.1.2 describe a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

3.1.1.1 State Model

3.1.1.1.1 Scope

The specification of all Active Directory protocols is based on a definition, shared by all Active Directory protocols, of the state of a server running Active Directory that is implied by the protocols. Call this the "state model" of Active Directory.

The Active Directory state model is divided into two categories:

1. Certain state that is represented as objects and attributes within Active Directory is *promoted directly* into the state model. State within Active Directory becomes part of the state model if it satisfies one of the following conditions:

    1. It is replicated.

    2. It is nonreplicated, but a protocol exists in the Windows Server operating system protocol documentation set whose behavior is dependent upon the state.

    The representation of nonreplicated state that is only accessed by a process running on the same server, that is itself implementing Active Directory, is private to the implementation. Therefore, such attributes are *not* promoted directly into the state model. It might still be required for this state to be modeled as described in category 2 later in this section.

    Excluded from the second condition above is all generic access by browsing tools such as ldp.exe that can access any attribute of any object in the directory. If ldp.exe or a similar tool covered by a Windows license can display or even modify a **nonreplicated attribute** of an object using only the attribute's **syntax** as defined by the schema, that does not make the attribute part of the state model. If ldp.exe or a similar tool covered by a Windows license accesses a nonreplicated attribute and decodes or encodes its value using information outside the attribute's syntax as

defined by the schema, that nonreplicated attribute is included in the state model under condition 1 (2) above. For example, by using LDP, it is possible to look at a nonreplicated attribute using an attribute's syntax of type String(Unicode). However, if the string stored in that attribute would be an XML content defined by an external XSD, then if LDP had special knowledge of how to interpret that XML, that nonreplicated attribute would be included in the state model under condition 1 (2) above.

2. Other state, however represented within Active Directory, is "abstracted" in the state model. Such state is included only as necessitated by the requirement that a licensee implementation of Windows Server protocols be able to receive messages and respond in the same manner as a Windows Server.

   For example, certain values sent by the Active Directory replication protocol [MS-DRSR] are accompanied by metadata. If the replicated values are stored by the receiving system, it must also store the metadata associated with the values. Otherwise, the receiving system will make incorrect responses to subsequent replication requests. These incorrect responses will, in general, prevent replication from converging. So this metadata must be included within the state model. The specific way that this metadata is stored by Active Directory, and the algorithms that optimize access to this metadata, are excluded from the state model.

   The various indexes used by the Active Directory implementation to improve the performance of directory search are another example of state within Active Directory. These indexes have no effect, other than performance, on the protocol responses that Active Directory makes. Therefore, these indexes are not included in the state model.

In this specification, the first category of state is modeled in a variant of LDAP information structures: naming contexts, objects, attributes, and values. These structures are defined precisely in the following sections. The set of replicated attributes is defined in [MS-ADA1], [MS-ADA2], and [MS-ADA3]. The set of nonreplicated attributes covered under condition 1 (2) (described earlier in this section) consists of the repsFrom and repsTo attributes documented in [MS-DRSR] sections 5.169 and 5.170.

Note Only the schema elements and instances of objects that are fundamental to Active Directory are described in this specification. If a protocol defines its own schema objects or otherwise creates its own objects in the directory, those objects are described in that protocol's specification. A summary of schema elements defined by such other protocols is included in [MS-ADA1], [MS-ADA2], [MS-ADA3], [MS-ADSC], and [MS-ADLS] as a convenience for the reader, but the documentation for the protocols using those schema elements should be consulted for a complete description.

In this specification, the second category of state is modeled using standard mathematical concepts. The concepts used and their associated notational conventions are described in the next section.

LDAP mandates very little about the behavior of a directory. Active Directory has many specific behaviors that are observable through LDAP. The remainder of this section describes the most pervasive of these behaviors. The remainder of the specification completes the discussion.

3.1.1.1.2 State Modeling Primitives and Notational Conventions

Attribute names are underlined in this document, as specified in section 1. If a variable *o* refers to an object, and a is an attribute name, then *o*!a denotes the value or values of attribute a on object *o*. If attribute a is not present on *o*, the value of *o*!a is null.

The specification uses the LDAP display names of attributes and object classes when referring to specific attributes and object classes. So if *o* refers to an object,

o!name

denotes the name attribute of object *o*.

Some attributes in this specification are abstract in the sense of [MS-DRSR] section 3.3.3. Abstract attribute names are also underlined, for example, repsFrom. RootDSE attribute names are also underlined, for example, dumpDatabase, even though rootDSE attributes are not declared as attributes in the schema.

This specification models state in category 2 from the previous section using the standard mathematical concepts of set, sequence, directed graph, and tuple.

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, which are 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* corresponding 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 fixed-length sequence can be constructed using the notation:

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

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 this example:

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

This defines **DSName** as a type of tuple with a DN–valued field *dn*, a **GUID**–valued field *guid*, and a **SID**–valued field *sid*.

3.1.1.1.3 Basics, objectGUID, and Special Attribute Behavior

The LDAP data model is defined by [RFC3377]. Because the LDAP RFCs and their underlying ITU specifications have been interpreted in a variety of ways, this section defines a more specific model that correctly represents the behavior of Active Directory objects and attributes and describes the correspondence between this model and the LDAP model.

The model is based on the general definitions of Replica, Object, and Attribute given in section 1, and repeated here for convenience:

A *replica* is a variable containing a set of objects.

An *attribute* is an identifier for a set of values.

An *object* is 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 either contains the

value of the parent's identifying attribute, or contains a reserved value (**NULL GUID**, as described later in this section) 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 edges, with the child as an edge's tail and the parent as an edge's head.

Note that 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 word replica suggests, it is often the case that two replicas 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 (replication) to converge both the set of objects in existence and the values of the non-identifying attributes as **originating updates** take place in replicas. When the members of a set of replicas are considered to be the same, it is common to say "an object" as a shorthand referring to the set of corresponding objects in the replicas.

A child object is an object that is not the root of its oriented tree. The children of an object *o* is the set of all objects whose parent is *o*.

The directory model used in this specification instantiates the preceding definitions as follows. The identifying attribute is objectGUID and the parent-identifying attribute is parent, an abstract attribute. Both attributes have **GUID** values. No actual object has objectGUID equal to the NULL GUID (all zeros), and the root object has parent equal to the NULL GUID.

This specification uses the following s-expression representation ([LISP15]) of directory values, attributes, objects, and replicas to provide a notation for examples:

- Represent an attribute and its values as a list (Attr Val_1 Val_2 ... Val_n) where *Attr* is an atom whose name is the attribute's name (its lDAPDisplayName, defined in section 3.1.1.2) and each Val_k is a value. The attribute comes first, but the ordering of values in the list is not significant, with the exception of the values of the objectClass attribute explained later in this section. If a value is a GUID, represent it as a 128-bit unsigned integer instead of using a representation that reflects the internal structure of a GUID. To aid the readability of examples, the GUIDs used in examples are unrealistically small integers.
- Represent an object as a list ($Attrval_1$ $Attrval_2$ ...$Attrval_n$) where each $Attrval_k$ is the representation of an attribute and its values; the ordering of this list is not significant.
- Represent a replica as a list (Obj_1 Obj_2 ... Obj_n) where each Obj_k is the representation of an object; the ordering of this list is not significant.

The following list

```
(
  ( (objectGUID 5) (parent 0) (dc "microsoft") )
  ( (objectGUID 2) (parent 5) (ou "NTDEV") )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston") )
)
```

is one representation of the value of some replica containing three objects. The object with objectGUID = 5 is the root, the object with objectGUID = 2 is the only child of the root, and the object with objectGUID = 9 is the only grandchild of the root. Each object in this example has one additional attribute whose meaning has not yet been described.

Representing an attribute as its lDAPDisplayName makes examples readable. In the actual state model, an attribute is identified by an **ATTRTYP**. An ATTRTYP is a 32-bit unsigned integer that can be mapped to and from an object representing an attribute. This mapping is specified in section 3.1.1.2.6.

Active Directory's objectGUID attribute has special behavior. A fresh GUID is assigned to the objectGUID attribute of an object during its creation (LDAP Add), and this attribute is read-only thereafter. This is the first of many examples of an attribute with special behavior. Section 3.1.1.5 specifies the behavior of every attribute that has special behavior.

Active Directory includes the objectSid attribute on certain objects, called **security principal objects**. The objectSid attribute has special behavior. A fresh **SID** is assigned to the objectSid attribute of an object during its creation (LDAP Add), and this attribute is read-only thereafter, unless the object moves to another NC (LDAP Modify DN; see section 3.1.1.5 for the specification of such moves). More on objectSid generation can be found in section 3.1.1.1.5.

3.1.1.1.4 objectClass, RDN, DN, Constructed Attributes, Secret Attributes

A directory object is constrained by the directory's schema, which is a set of predicates. A few schema concepts are mentioned here. A full understanding of these concepts is not required to understand this section; additional information is available in the Glossary or in section 3.1.1.2.

When an object is created, it is assigned a most specific **structural object class** or an **88 object class**, plus the sequence of object classes that this class inherits from. The set of inherited classes always includes the class top. The value of an object's objectClass attribute is the full set of object classes (each identified by lDAPDisplayName) assigned to the object. The example in the previous section is elaborated in the following list.

```
(
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS) )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit) )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user) )
)
```

This list represents three objects, including their first and last objectClass values. The intermediate objectClass values are elided. Unlike all other multivalued attributes, the ordering of objectClass values is significant—top is always listed first; the most specific structural object class (or the 88 object class used in place of the structural class) is always listed last. So, for instance, the most specific structural object class of the root is domainDNS.

Representing a class as its lDAPDisplayName makes examples readable. In the actual state model, a class is identified by an ATTRTYP. An ATTRTYP is a 32-bit unsigned integer that can be mapped to and from the schema object representing a class. This mapping is specified in section 3.1.1.2.6.

In Active Directory, each object has an RDN attribute, which is determined by the most specific structural object class of the object when the object is created. The RDN attribute is the attribute that defines an object's name relative to its parent. In Active Directory, the RDN attribute of an object class has String(Unicode) syntax; that is, its value is a Unicode string, and the RDN attribute of an object always has exactly one value. (See section 3.1.1.2 for more on the topic of **attribute syntax**.)

Confusingly, the Active Directory schema includes an attribute whose attributeSchema object's cn is "RDN"; this is the name attribute, described later in this section. The term "RDN attribute" never refers to the name attribute in this document.

The **RDN** of an object is a string of the form "att=val" where *att* is the lDAPDisplayName of the RDN attribute of the object and *val* is the value of the RDN attribute on this object. In the preceding example, the object class user has RDN attribute cn, as can be confirmed by consulting [MS-ADSC]. Therefore the RDN of the object with objectGUID = 9 is "cn=Peter Houston". An RDN can also be written using the attributeID of the RDN attribute in place of its lDAPDisplayName; the example just given becomes "2.5.4.3=Peter Houston". The RDN form based on lDAPDisplayName is used throughout this document.

Active Directory requires that the value parts of the RDNs of all children of an object be distinct. This guarantees that the RDNs of all children of an object are distinct.

The DN of an object is defined recursively as follows. The DN of the root has an assigned value; the way Active Directory assigns this value is described later in section 3.1.1.1.5. The DN of a child object is the RDN of the child, followed by "," and the DN of the parent. In the preceding example, suppose the assigned DN of the root object is "dc=microsoft,dc=com". Then the DN of the object with objectGUID = 9 is "cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com".

The correspondence between this model and the LDAP data model is as follows. An object with its attributes and values corresponds to an LDAP **entry** with its attributes and values. This model and LDAP agree on the definition of the objectClass attribute. The definition of RDN in this model is a subset of LDAP's definition; all RDNs in this model are valid LDAP RDNs, but not vice versa. For example, the following multivalued RDN is a valid LDAP RDN, but it is not valid in this model: "cn=Peter Houston+employeeID=ABC123". Given the RDN definition, the definition of DN in this model is the same as LDAP's definition. In the LDAP data model, the child-parent relationship is represented in the DNs of the child and parent, whereas in the Active Directory data model, the child-parent relationship is represented in the parent attribute and the DN is derived. Active Directory does not expose the model's parent attribute through LDAP.

Active Directory includes the distinguishedName attribute on every object; the value is the object's DN. The following example elaborates the previous example to include a value of distinguishedName on each object.

```
(
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (distinguishedName "dc=microsoft,dc=com") )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit)
    (distinguishedName "ou=NTDEV,dc=microsoft,dc=com" ) )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user)
    (distinguishedName
      "cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com") )
)
```

But including distinguishedName on each object this way is misleading, because the distinguishedName attribute is not stored as a string on each object. If it were stored as a string on each object, renaming an object would require updating every object in the subtree rooted at the renamed object. For a large subtree, this would take a long time and would either interfere with other directory activity (if performed as a single transaction) or would expose observable

inconsistency to clients (if performed as multiple transactions). Active Directory does neither of these, so its state model can't imply that it does.

The distinguishedName attribute is not declared in the schema as a constructed attribute, but it behaves like one. Normal attributes, including attributes with special behavior such as objectGUID, have their values stored as part of an object's representation. Constructed attributes have the property that they have values computed from normal attributes (for read) and/or have effects on the values of normal attributes (for write). Constructed attributes are not included in the state model. Because the distinguishedName attribute behaves like a constructed attribute in that it also contributes no state to an instance of an object, it is not considered to be part of the state model.

Active Directory includes the name attribute on every object. An object's value of name equals the value of the object's RDN attribute. The following example removes the incorrect modeling of distinguishedName from the previous example, then elaborates that example to include name.

```
(
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (name "microsoft") )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit)
    (name "NTDEV") )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user)
    (name "Peter Houston") )
)
```

The name attribute has special behavior. Even if an object is renamed (LDAP Modify DN), the object's name attribute remains equal to the object's RDN attribute. As with the distinguishedName attribute, the name attribute is not declared in the schema as a constructed attribute, but it behaves like one.

Because Active Directory requires that the value parts of the RDNs of all children of an object be distinct, it follows that the name attribute of all children of an object are distinct.

Active Directory includes the rdnType attribute on every object. An object's value of rdnType is the object's RDN attribute at object creation time—the identifier, not its associated value. The following example elaborates the previous example to include rdnType.

```
(
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (name "microsoft") (rdnType dc))
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit)
    (name "NTDEV") (rdnType ou))
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user)
    (name "Peter Houston") (rdnType cn))
)
```

The rdnType attribute, like the parent attribute, is not declared in the Active Directory schema. [MS-DRSR] section 5.158 specifies the special behavior of the rdnType attribute.

A **secret attribute** is any attribute from the following set: currentValue, dBCSPwd, initialAuthIncoming, initialAuthOutgoing, lmPwdHistory, ntPwdHistory, priorValue, supplementalCredentials, trustAuthIncoming, trustAuthOutgoing, and unicodePwd.

3.1.1.1.5 NC, NC Replica

The type **DSNAME** is defined as a C structure in [MS-DRSR] section 5.50; this state model uses the simpler **DSName**, which contains the same information in a tuple of the form:

DSName: [*dn*: DN; *guid*: GUID; *sid*: SID]

An NC is a set of objects organized as a tree. It is referenced by a **DSName** containing a non-NULL dn and a non-NULL GUID. This **DSName** also references the NC root, which is the root object of the tree of objects in the NC. The NC root has the IT_NC_HEAD bit set in the instanceType attribute. Any instance of the NC on any DC is called an NC replica. It is convenient to say "the NC x" where x is the **DSName** referencing the NC.

A replica of NC *x* is a replica as already defined, with its root object *r* constrained as follows:

- r!objectGUID = x.guid
- r!distinguishedName = x.dn
- If x.sid ≠ NULL then r!objectSid = x.sid, otherwise r!objectSid = NULL

Mutation of a replica in the general sense is unconstrained. In the case of a replica of a specific NC, the root object cannot be replaced, because doing so would change the objectGUID (and objectSid if any), and this must equal the NC's *guid*. In a replica of a given NC the root object's DN cannot be changed, because the root object's DN must equal the NC's *dn*.

All replicas in Active Directory are NC replicas.

NC replicas are mutable. The term originating update means any mutation to an NC replica performed via any protocol except replication.

Active Directory performs replication between replicas of the same NC to converge their states, so an update originated on one replica is reflected in all the others. The replication algorithm has the property that if originating updates to all replicas ceases and communication between replicas is maintained, the application-visible states of the replicas will eventually converge to a common value. Applications of Active Directory can read from several replicas of a given NC and observe the differences, but applications typically bind to a single replica.

Active Directory supports four NC types:

Domain NC: A domain naming context. The *sid* field of a domain NC is not NULL.

Config NC: An NC that stores Active Directory configuration information. The *sid* field of a config NC is NULL.

Schema NC: An NC that stores Active Directory schema information. The *sid* field of a **schema NC** is NULL.

Application NC: An **application NC**. The *sid* field of an application NC is NULL.

The *dn* of a domain NC or an AD DS application NC takes the form:

dc=n_1,dc=n_2, ... dc=n_k

where each n_i satisfies the syntactic requirements of a **DNS name** component [RFC1034]. Such a DN corresponds to the DNS name:

n_1. n_2. ... .n_k

This is the *DNS name of the NC*. The mapping just specified follows [RFC2247].

In AD LDS, an application NC can have any valid DN; therefore an AD LDS application NC does not necessarily have a DNS name.

Replicas of a domain NC have one of these two subtypes:

- *Full*. A replica whose objects contain their full state as defined by all originating updates.
- *Partial*. A replica whose objects contain a filtered view of the full state as defined by all originating updates. There are three types of the partial replica:
  - **GC partial NC replica**: The filter removes all attributes (and their values) that are not in the partial replica's **GC partial attribute set**.
  - **Filtered partial NC replica**: The filter removes all the attributes (and their values) that are in the filtered attribute set. The **default NC**, config NC, and application NC on a RODC are filtered partial NC replicas.
  - **Filtered GC partial NC replica**: The filter removes all the attributes (and their values) that are not in the partial replica's GC partial attribute set, as well as all the attributes (and their values) in the filtered attribute set. Domain NCs, excluding the **default domain NC**, that are hosted on an RODC are filtered GC partial NC replicas. Such domain NCs will exist on the RODC when the RODC is a GC.

Replicas of other NC types are always full. A full replica is either *writable*, that is, it accepts originating updates, or is read-only. A partial replica is read-only.

This section has introduced many concepts without describing how they are reflected in the state model. To a great extent this obligation will be discharged in other sections of this document. The schema NC is described in section 3.1.1.2, while the other NC types are described in section 6.1. Here are three elaborations of the state model that can be explained without making a forward reference:

1. NC replicas are modeled by making a **DSName**, converted into a string formatted as specified in [MS-DRSR] section 5.16.2.1, the first element of a replica.
2. The root object of a domain NC or an AD DS application NC has class domainDNS. The RDN attribute of domainDNS is dc. Therefore both the dc and name attributes of the root object of a domain NC or an AD DS application NC equal the first component (for example, n_1 for DNS name n_1. n_2. ... . n_k) of the NC's DNS name. The root object of an AD LDS application NC can have any object class except dMD or configuration.
3. In AD DS, the generation of objectSid values is constrained by the *sid* of a domain NC as follows. The *sid* of a domain NC, the domain SID, is a SID with four **SubAuthority** values. The root object of a domain NC has objectSid equal to the domain SID, as required by the definition of NC replica. Every security principal object *o* in a domain NC has *o*!objectSid equal to the domain SID plus the RID portion (that is, it has five SubAuthority values). The RID portion of *o*!objectSid is a number not assigned as the RID portion of the objectSid to any other object of the domain, including objects that existed earlier but have been deleted.

Section 3.1.1.5.2.4 specifies how AD DS assigns RIDs. The same section specifies how AD LDS generates objectSid values for new AD LDS **security principals**.

Continuing the example, let the example NC be a domain NC, and let the object with name "Peter Houston" be assigned the RID value 2055 (decimal). Then the state of the example NC is as follows.

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;
   dc=microsoft,dc=com"
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (name "microsoft") (rdnType dc)
    (objectSid 0x0105...94E1F2E6) )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit)
    (name "NTDEV") (rdnType ou) )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user)
    (name "Peter Houston") (rdnType cn)
    (objectSid 0x0105...94E1F2E607080000) )
)
```

The DNS name of this domain NC is *microsoft.com*. Note that the domain SID is a prefix of the "Peter Houston" object's objectSid. Portions of the (long) SID values have been elided for clarity; consider the elided portions to be the following hex digits

```
0000000000051500000089598D33D3C56B68
```

and the example SID will be a valid SID.

3.1.1.1.5.1 Tombstone Lifetime and Deleted-Object Lifetime

The **tombstone lifetime** is controlled by the tombstoneLifetime attribute of the Directory Services object specified in section 6.1.1.2.4.1.1, interpreted as a number of days. If no value is specified for the tombstoneLifetime attribute of the Directory Services object, the tombstone lifetime defaults to 60 days. The minimum value that can be specified is 2 days. If a value of less than 2 days is specified, tombstone lifetime defaults to 60 days, except for Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, where the tombstone lifetime defaults to 2 days.

The deleted-object lifetime is controlled by the msDS-DeletedObjectLifetime attribute of the Directory Services object specified in section 6.1.1.2.4.1.1, interpreted as a number of days. If no value is specified for the msDS-DeletedObjectLifetime attribute of the Directory Services object, deleted-object lifetime defaults to the tombstone lifetime as calculated above. The minimum value that can be specified is 2 days. If a value less than 2 days is specified, deleted-object lifetime defaults to 2 days.

3.1.1.1.6 Attribute Syntaxes, Object References, Referential Integrity, and Well-Known Objects

The complete set of attribute syntaxes supported by Active Directory are specified in section 3.1.1.2. The representation used by the abstract data model for values of each attribute syntax is specified in [MS-DRSR] section 5.16.2. These representations of each syntax can be returned in an

LDAP response without conversion, that is, the values are represented in the abstract data model in the same format as used by the LDAP protocol.

The following five attribute syntaxes are called **object reference** syntaxes:

- Object(DS-DN)
- Object(DN-String)
- Object(DN-Binary)
- Object(Access-Point)
- Object(OR-Name)

The values of an attribute with Object(DS-DN) syntax are DNs, which represent references to objects. The values of attributes with the other object reference syntaxes have two portions; one portion is a DN, which represents a reference to an object, and the other has information specific to each syntax. The five object reference syntaxes have a special behavior called "referential integrity"; no other attribute syntax have special behavior intrinsic to the syntax. The referential integrity behavior applies only to the DN portion of the syntax (the portion that represents a reference to an object), leaving the remaining portion unchanged. For this reason, and because the referential integrity is the same for the DN portion of all five object reference syntaxes, it suffices to specify the referential integrity behavior of syntax (the portion that represents a reference to an object), leaving the remaining portion unchanged. For this reason, and because the referential integrity is the same for the DN portion of all five object reference syntaxes, it suffices to specify the referential integrity behavior only for the Object(DS-DN) syntax (the simplest of the object reference syntaxes).

To specify referential integrity, some background on object deletion is required; object deletion is specified fully in section 3.1.1.5.

When the **Recycle Bin** optional feature is not enabled, object deletion is performed in two stages.

1. In the first stage, the object to be deleted is transformed into a tombstone. A tombstone is a special object, part of a replica's state. The state of a deleted object's tombstone resembles the state of the object before deletion; it has the same objectGUID but a different DN. Specifically, its RDN is changed to a "delete-mangled RDN" and, in most cases, it is moved into the Deleted Objects container of its NC, as described in section 3.1.1.5.5. A tombstone is generally *not* an object from the LDAP perspective: a tombstone is not returned by a normal LDAP Search request, only by a Search request with extended control LDAP_SERVER_SHOW_DELETED_OID or LDAP_SERVER_SHOW_RECYCLED_OID, as described in section 3.1.1.3.

2. In the second stage, after a significant delay (the tombstone lifetime), a tombstone is **garbage collected**, which removes it from the replica's state.

When the Recycle Bin optional feature is enabled, object deletion is performed in three stages.

1. In the first stage, the object being deleted is transformed into a deleted-object. A deleted-object is a special object, part of a replica's state. The deleted-object also resembles the state of the object before deletion; it has the same objectGUID but a different DN. Specifically, its RDN is changed to a "delete-mangled RDN" and, in most cases, it is moved into the Deleted Objects container of its NC, as described in section 3.1.1.5.5. A deleted-object is generally *not* an object from the LDAP perspective: a deleted-object is not returned by a normal LDAP Search request, only by a Search request with extended control LDAP_SERVER_SHOW_DELETED_OID OID or LDAP_SERVER_SHOW_RECYCLED_OID, as described in section 3.1.1.3.

2. In the second stage, after a significant delay (the **deleted-object lifetime**), a deleted-object is transformed into a recycled-object. A recycled-object is a special object, part of a replica's state. The recycled-object also resembles the state of the object before deletion; it has the same objectGUID but a different DN. Specifically, its RDN has been changed and, in most cases, the object moved, as described in the first stage. A recycled-object is also generally *not* an object from the LDAP perspective: a recycled-object is not returned by a normal LDAP Search request, only by a Search request with extended control LDAP_SERVER_SHOW_RECYCLED_OID, as described in section 3.1.1.3.

   Note that this transformation from deleted-object to recycled-object is only initiated on DCs where the deleted-object is in a **writable NC replica**. On DCs where the deleted-object is not in a writable NC replica, the transformation from deleted-object to recycled-object occurs as the result of replication in this state change from a DC that holds a writable copy of the object.

3. In the third and final stage, after a significant delay (the tombstone lifetime), a recycled-object is garbage collected, which removes it from the replica's state.

In situations where a deletion does not need to be replicated, an object is **expunged** (that is, removed in a single step from the replica's state) instead. A deletion does not need to be replicated in the following cases: removal of a **lingering object** (section 3.1.1.3.3.15), removal of an object being moved during a cross-domain move (section 3.1.1.5.4.2), and removal of a **dynamic object** (section 6.1.7).

An application is not limited to specifying a DN when creating an object reference; using the syntax specified in section 3.1.1.2, it can specify any combination of DN, SID, or GUID as the reference as long as it specifies at least one. A **DSName** is created using the specified references and is resolved to an object using DSName equality as defined in [MS-DRSR] section 5.50, DSNAME.

The state kept with an attribute to represent an object reference is a **DSName**.

When reading an object reference, an application can request the full **DSName** in the representation specified in [MS-DRSR] section 5.16.2.1 instead of a DN by passing the LDAP_SERVER_EXTENDED_DN_OID extended control as described in section 3.1.1.3.

A single-valued Object(DS-DN) attribute *a* on object *src* behaves as follows:

- When an LDAP Add or Modify creates an object reference within attribute *src.a*, the server uses the DN (or SID or GUID) specified in the Add or Modify to locate an existing object *dst*. If no such object exists, including the case where the object has been deleted and exists as a tombstone, deleted-object, or recycled-object, the request fails with error *noSuchObject* / *ERROR_DS_OBJ_NOT_FOUND*. The values *dst*!distinguishedName, *dst*!objectGUID and *dst*!objectSid are used to populate the **DSName** representing the object reference within attribute *src.a*. If the object *dst* has no objectSid attribute, the "SID=" portion of the **DSName** representation is omitted.

- If object *dst* has not been deleted, reading attribute *a* gives the DN (or extended format as described in section 3.1.1.3) of object *dst*, even if *dst* has been renamed since *a* was written.

- If the object *dst* has been deleted or expunged, reading *src.a* gives a **DN** field that corresponds to no object. Either this DN is impossible to create via LDAP Add and LDAP Modify DN, or this DN changes (that is, the value of *src.a* changes) when an LDAP Add or Modify DN would give some other object this DN.

The multivalued case is similar; a multivalued attribute is capable of containing multiple object references that behave as described.

Each object reference syntax exists in two versions. The description just given is for the "nonlink" version. The other version is the "forward link". The Object(DS-DN) syntax also exists in a "back link" version.

A forward link Object(DS-DN) attribute supports the definition of a corresponding back link Object(DS-DN) attribute. The **back link attribute** is a read-only constructed attribute; clients MUST NOT write to the back link attribute, and servers MUST reject any such writes. If an object *o* contains a reference to object *r* in **forward link attribute** *f*, and there exists a back link attribute *b* corresponding to *f*, then a **back link value** referencing *o* exists in attribute *b* on object *r*. The correspondence between the forward and back link attributes is expressed in the schema; see section 3.1.1.2 for details. A forward link attribute can exist with no corresponding back link attribute, but not vice versa.

If the syntax of a forward link attribute is not Object(DS-DN), a corresponding back link attribute has syntax Object(DS-DN), *not* the syntax of the forward link. The non-reference portion of the forward link, if any, is ignored in computing the back link. If ignoring the non-reference portion of the forward link results in duplicate back references, the duplicates are present in the values of the back link attribute.

The referential integrity behavior of a forward link attribute differs from that of a nonlink attribute as follows:

- When an object *o* is expunged or transformed into a tombstone or recycled-object, any forward link reference to *o* is removed from the attribute that contains it.
- When an object *o* is transformed into a deleted-object, any forward link reference to *o* is maintained, but is made invisible to LDAP operations that do not specify the LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID control.
- When a deleted-object *o* is transformed into an object that is not a deleted-object, a tombstone, or a recycled-object, any forward link reference to *o* from object *p* where *p* is not a deleted-object is made visible to LDAP operations. Similarly, any forward link reference from *o* to *p* is made visible to LDAP operations.

Since a back link attribute is constructed, its referential integrity behavior follows from that of the corresponding forward link attribute.

The distinction between nonlink and forward link references is not visible in the part of the state model described in this section; it is a schema difference only. There is no difference in the state kept with an attribute to represent the object reference. There is a difference in the replication metadata accompanying the object reference, as will be described in section 3.1.1.1.9.

The behavior described in this section is for object references within a single NC replica. Additional behaviors, specified in section 3.1.1.1.12, are possible when an object reference crosses an NC replica boundary.

Extend the running example by adding a **group object** named "DSYS" as a child of "ou=NTDEV,dc=microsoft,dc=com". The object class group includes the attribute member with Object(DS-DN) syntax. In this example, the "DSYS" group has the **user object** "Peter Houston" as its only member.

```
(
  "<GUID=5>;<SID=0x0105...00000000>;dc=microsoft,dc=com"
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (name "microsoft") (rdnType dc)
```

```
    (objectSid 0x0105...94E1F2E6) )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit)
    (name "NTDEV") (rdnType ou) )
  ( (objectGUID 9) (parent 2) (cn "Peter Houston")
    (objectClass top ... user)
    (name "Peter Houston") (rdnType cn)
    (objectSid 0x0105...94E1F2E607080000) )
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (member
      "<GUID=9>;<SID=0x0105...07080000>;
      cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com" ) )
)
```

Note that the group "DSYS" is a security principal object within the domain NC, with the distinct RID value 2059 (decimal).

The root object of each NC contains the attribute wellKnownObjects. The purpose of this attribute is to provide a location-independent way to access certain objects within the NC. For instance, the Deleted Objects container where most tombstones live can be found using wellKnownObjects.

The wellKnownObjects attribute has syntax Object(DN-Binary). Each value consists of an object reference *ref* and a byte string *binary* that is 16 bytes long. The byte string *binary* contains a GUID identifying a **well-known object (WKO)** within an NC; the object reference *ref* is a reference to the corresponding object. A table of the GUIDs that identify well-known objects is given in section 6.1.1.4.

The following procedure implements well-known object location using the wellKnownObjects attribute. This procedure will be used throughout the rest of this specification:

- procedure GetWellknownObject(*nc*: NC, *guid*: GUID): DSName
  - If there is no replica of NC *nc* on the server executing this procedure, return null.
  - Let *v* be the value of *nc*!wellKnownObjects on the server's replica satisfying *v.binary* = *guid*; if no such *v* exists, return null.
  - Return *v.ref*.

Assignments to the wellKnownObjects attribute are specially checked as described in section 3.1.1.5.

LDAP supports access to well-known objects using an extended **DSName** syntax as described in section 3.1.1.3.

3.1.1.1.7 Forest, Canonical Name

An Active Directory forest is a set of NCs. Every forest contains one schema NC and one config NC. The other types of NCs present in a forest depends on whether it is an AD DS forest or an AD LDS forest:

- AD DS: Every forest also contains one or more domain NCs, and zero or more application NCs.
- AD LDS: Every forest also contains zero or more application NCs.

The NCs within a forest are related by their assigned DNs as follows:

- In AD DS there must exist a domain NC *root* such that the config NC's *dn* equals *Cat*("cn=Configuration", *root.dn*) (where *Cat* is the string concatenation function). This unique domain NC is called the **root domain** NC of the forest.

  Describe this DN relationship as "The config NC is a child of the root domain NC". Technically these NCs are not related in the same way that a child object and its parent object are related within an NC; the parent relationship stops at the root of an NC. But their DNs are related in the same way as the DNs of a child object and its parent object within an NC. Given NCs with their corresponding DNs forming a child and parent relationship, it is convenient to refer to the NCs as the **child NC** and the **parent NC**.

  In AD LDS, the config NC does not have a parent NC. An AD LDS forest contains no domain NCs, so there is no **forest root domain NC**, either. The DN of an AD LDS config NC takes the form "CN=Configuration, CN={*G*}" where *G* is a GUID in dashed-string form ([RFC4122] section 3). For example,

  CN=Configuration, CN={FD783EE9-0216-4B83-8A2A-60E45AECCB81}

  is a possible DN of the config NC in an AD LDS forest.

- The schema NC is a child of the config NC, with RDN "cn=Schema".

- If *short* and *long* are NCs with **DNS** names (domain NCs or application NCs), and *short* is a suffix of *long*, then each DNS name obtained by removing DNS name components successively from the front of *long* until the result is *short* must also name NCs with DNS names. For instance, if a forest contains both NCs *microsoft.com* and *nttest.ntdev.microsoft.com*, it must also contain NC *ntdev.microsoft.com*.

- If *app* is an application NC and *dom* is a domain NC, then *dom* must not be a child of *app*.

- If *root* is the root domain NC and *dom* is another domain NC in the forest, then *root* must not be a child of *dom*.

Extend the running example by adding the config NC and schema NC as follows.

```
(
  "<GUID=4>;cn=Configuration,dc=microsoft,dc=com"
  ( (objectGUID 4) (parent 0) (cn "Configuration")
    (objectClass top ... configuration)
    (name "Configuration") (rdnType cn) )
)
(
  "<GUID=8>;cn=Schema,cn=Configuration,dc=microsoft,dc=com"
  ( (objectGUID 8) (parent 0) (cn "Schema")
    (objectClass top ... dMD)
    (name "Schema") (rdnType cn) )
)
(
  "<GUID=5>;<SID=0x0105...00000000>;dc=microsoft,dc=com"
  ( (objectGUID 5) (parent 0) (dc "microsoft")
    (objectClass top ... domainDNS)
    (name "microsoft") (rdnType dc)
    (objectSid 0x0105...94E1F2E6) )
  ( (objectGUID 2) (parent 5) (ou "NTDEV")
    (objectClass top ... organizationalUnit)
    (name "NTDEV") (rdnType ou))
```

```
    ( (objectGUID 9) (parent 2) (cn "Peter Houston")
      (objectClass top ... user)
      (name "Peter Houston") (rdnType cn)
      (objectSid 0x0105...94E1F2E607080000) )
    ( (objectGUID 6) (parent 2) (cn "DSYS")
      (objectClass top ... group)
      (name "DSYS") (rdnType cn)
      (objectSid 0x0105...94E1F2E60B080000)
      (members
        "<GUID=9>;<SID=0x0105...07080000>;
        cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com" ) )
  )
```

This example illustrates the *dn* relationships between the root domain NC, config NC, and schema NC. It shows that in a forest, the parent relationship does not cross NC boundaries. It also illustrates the object classes of the config NC and schema NC root objects and the lack of a *sid* in these NCs. It does not show the contents of these NCs, which are specified in sections 6.1 and 3.1.1.2.

Every object in a forest has a **canonical name**. The canonical name of an object is a syntactic transformation of its DN into something resembling a pathname that still identifies the object. A canonical name is a DNS name, followed by a "/", followed by a sequence of zero or more names separated by "/". The DNS name is the translation of the final sequence of "dc=" DN components into an equivalent DNS name (following [RFC2247]). The sequence of names is the sequence of names in the non-"dc=" DN components, appearing in the reverse order to the order they appeared in the DN. Here are several examples of this translation drawn from the preceding example.

```
DN:              cn=Peter Houston, ou=NTDEV, dc=microsoft,
                 dc=com
canonical name: microsoft.com/NTDEV/Peter Houston

DN:              cn=Configuration, dc=microsoft, dc=com
canonical name: microsoft.com/Configuration

DN:              dc=microsoft, dc=com
canonical name: microsoft.com/
```

Active Directory supports a constructed attribute canonicalName on every object. Its value is the object's canonical name.

3.1.1.1.8 GC

In AD DS, the global catalog (GC) is a partial view of a forest's NCs, with these properties:

- The GC view includes all domain NCs, the config NC, and the schema NC.
- The GC view is partial. It includes all objects in the included NCs, but only those attributes defined as members of the partial attribute set in the schema NC (as specified in section 3.1.1.2). If the GC is an RODC, the attribute list is further restricted to those attributes not present in the filtered attribute set in the schema NC (as specified in section 3.1.1.2).
- The GC view is read-only.

The GC has no state model impact outside the schema NC, which defines the forest's partial attribute set. The implementation of the GC (that is, actually providing the specified view to LDAP clients) does have impact, explained in section 3.1.1.1.9.

In AD LDS there is no support for the GC.

3.1.1.1.9 DCs, usn Counters, and the Originating Update Stamp

The model defines the state of a DC as a tuple of type DC.

```
type DC = [
    serverGuid: GUID,
    invocationId: GUID,
    usn: 64-bit integer,
    prefixTable: PrefixTable,
    defaultNC: domain NC replica,
    configNC: config NC replica,
    schemaNC: schema NC replica,
    partialDomainNCs: set of partial domain NC replica,
    appNCs: set of application NC replica,
    pdcChangeLog: PDCChangeLog
    nt4ReplicationState: NT4ReplicationState
    ldapConnections: LDAPConnections,
    replicationQueue: ReplicationQueue,
    kccFailedConnections: KCCFailedConnections,
    kccFailedLinks: KCCFailedLinks,
    rpcClientContexts: RPCClientContexts,
    rpcOutgoingContexts: RPCOutgoingContexts,
    fLinkValueStampEnabled: boolean,
    nt4EmulatorEnabled: boolean,
    fEnableUpdates: boolean
    dnsRegistrationSettings: DNSRegistrationSettings
    minimumGetChangesRequestVersion: integer
    minimumGetChangesReplyVersion: integer
]
```

The variable **dc** is the only global variable in this specification. It contains the state of the DC.

```
dc: DC
```

serverGuid is initialized to a fresh GUID when the *dc* is created and does not change thereafter. Section 6.1.1.2.2.1.2.1.1 describes the nTDSDSA object; *serverGuid* equals the objectGUID of the DC's nTDSDSA object. *serverGuid* is independent of the objectGUID of the **computer object** for the computer playing the role of this DC.

invocationId is initialized to a fresh GUID when the *dc* is created. The circumstances under which a DC changes its *invocationId* are outside the effects of the state model. A DC changes its *invocationId* when Active Directory is restored from a backup. Section 6.1.1.2.2.1.2.1.1 describes the nTDSDSA object; *invocationId* equals the invocationId of the DC's nTDSDSA object.

usn is a counter used in assigning replication metadata to every originating update to an NC replica in the DC, as detailed later in this section. The invocationId of *dc's* nTDSDSA object is an "epoch number" for *usn*; if an observer reads a *dc* at times t_1 and t_2 with $t_1 < t_2$, and invocationId is the same, then *usn* at time t_1 is less than or equal to *usn* at time t_2. If the invocationId has been

changed between t_1 and t_2, the DC at t_2 is treated as a different DC then at t_1 for the purposes of replication, and the *usn* of the DC is not compared.

prefixTable is the PrefixTable used to translate all ATTRTYP values stored in this DC's NC replicas; section 3.1.1.2.6 specifies the translation process.

The default NC replica of an AD DS DC, modeled as *dc.defaultNC*, is a domain NC replica of some domain NC in the forest. In an AD LDS DC, *dc.defaultNC* is null.

The fields *dc.configNC* and *dc.schemaNC* contain replicas of the forest's config NC and schema NC.

If *dc* is not an AD DS **GC server** (as determined by the state of the GC bit of the options **attribute** of the nTDSDSA object as specified in section 6.1.1.2.2.1.2.1.1), then *dc.partialDomainNCs* is null. Otherwise it contains a partial domain NC replica for each domain NC in the forest, excluding the default domain NC of dc.

The field *dc.appNCs* contains replicas of the application NCs hosted by the DC. An AD DS DC can be an RODC; [MS-DRSR] section 5.7, AmIRODC, specifies how this is determined by state in the config NC.

All NC replicas of an RODC are read-only; that is, they do not accept originating updates. In other DCs, all NC replicas are writable except for *dc.partialDomainNCs*, but writes to these NC replicas are controlled by the constraints and processing specifics described in section 3.1.1.5. Also, on an RODC the *dc.defaultNC* is a filtered partial domain NC replica. On other DCs, the *dc.defaultNC* is a **full domain NC replica**, and is the only full domain NC replica in the state of a DC.

The *nt4ReplicationState* and *pdcChangeLog* variables contain state used by the IDL_DRSGetNT4ChangeLog method ([MS-DRSR] section 4.1.11.3). Section 3.1.1.7 specifies the format of these variables and how they are maintained during state changes in AD DS.

The *ldapConnections*, *replicationQueue*, *kccFailedConnections*, *kccFailedLinks*, *rpcClientContexts*, and *rpcOutgoingContexts* fields of a DC are volatile state. Each volatile field is set to the empty sequence on server startup. The other fields are persistent state, updated using transactions.

The construction of the *kccFailedConnections* and *kccFailedLinks* fields of a DC are discussed in section 6.2. The construction of the *replicationQueue*, *kccFailedConnections*, and *rpcOutgoingContexts* fields are discussed in [MS-DRSR]. The construction of the *fLinkValueStampEnabled* field is described later in this section.

The *nt4EmulatorEnabled* field determines how the DC responds to a Mailslot Ping request, as described in section 6.3.5. The *nt4EmulatorEnabled* field is not configurable through the Active Directory. The *nt4EmulatorEnabled* field can be configured by an implementation-dependent mechanism. On Windows Server operating system, the field *nt4EmulatorEnabled* can be configured at the following registry key path:

```
HKEY_LOCAL_MACHINE\system\currentcontrolset\services\netlogon\parameters\NT4Emulator
```

This registry value is of type REG_DWORD. If the value is 0 or not present, the field *nt4EmulatorEnabled* is set to FALSE; otherwise, the field is set to TRUE. By default, this registry value is not set.

The *fEnableUpdates* field determines whether or not a DC allows updates, as described in section 3.1.1.5.1.9. The field is initialized to TRUE.

The *dnsRegistrationSettings* field contains the settings that determine whether the DC registers DNS records (for the purpose of DC location), and which DNS records it registers. The field is of type DNSRegistrationSettings (section 6.3.1.10) and is initialized as described in section 6.3.1.10.

The *minimumGetChangesRequestVersion* field contains a value limiting the acceptable versions of the input message for a replication request. See [MS-DRSR] section 4.1.10.5.1. The value is set by DSA Heuristics (section 6.1.1.2.4.1.2).

The *minimumGetChangesReplyVersion* field contains a value limiting the acceptable versions of the output message for a replication request. See [MS-DRSR] section 4.1.10.5.18. The value is set by DSA Heuristics (section 6.1.1.2.4.1.2).

Each originating update on a DC creates replication metadata values (*AttributeStamp* and *LinkValueStamp* values), as will now be described.

AttributeStamp and *LinkValueStamp* values contain times read from the system clock of the server creating the value. If clocks on different DCs disagree by a significant fraction of the tombstone lifetime, then it is probable that different DCs will eventually disagree about whether some objects have been deleted or not; see section 3.1.1.1.15. DCs use Kerberos for mutual authentication, and Kerberos does not mutually authenticate two DCs whose clocks are more than 5 minutes out of sync. The tombstone lifetime is generally several months, so synchronization within 5 minutes is much better than required to avoid object lifetime issues.

The type *AttributeStamp* is defined authoritatively in [MS-DRSR] section 5.11. In summary, it is the following tuple.

```
AttributeStamp: [
    dwVersion: 32-bit Integer;
    timeChanged: 64-bit number of seconds
                 since January 1, 1601, 12:00:00am;
    uuidOriginating: GUID;
    usnOriginating: 64-bit Integer]
```

Similarly, the type *LinkValueStamp* is defined authoritatively in [MS-DRSR] section 5.117. In summary, it is an *AttributeStamp* tuple extended on the bottom with the following fields:

- *timeCreated*: 64-bit number of seconds since January 1, 1601, 12:00:00 A.M.
- *timeDeleted*: 64-bit number of seconds since January 1, 1601, 12:00:00 A.M.

An *AttributeStamp* stamp is associated with all replicated attributes, except forward link attributes updated when the **forest functional level** is greater than DS_BEHAVIOR_WIN2000 or *dc.fLinkValueStampEnabled* is TRUE, that have ever had values on an object. For forward link attributes updated when the forest functional level is greater than DS_BEHAVIOR_WIN2000 or *dc.fLinkValueStampEnabled* is TRUE, a *LinkValueStamp* stamp is associated with each value of the attribute, both current link values and *tombstoned* link values. More details on *tombstoned* link values are given later in this section.

Together with forest functional level, *dc.fLinkValueStampEnabled* regulates whether a DC creates replication metadata for forward link attributes. *dc.fLinkValueStampEnabled* is initialized to TRUE when the forest functional level is greater than DS_BEHAVIOR_WIN2000. When the forest functional level is DS_BEHAVIOR_WIN2000, *dc.fLinkValueStampEnabled* is initialized to FALSE. When a DC receives an update containing *LinkValueStamp* values, it sets *dc.fLinkValueStampEnabled* to TRUE. (For more information, see [MS-DRSR] sections 4.1.10.5.5 and 4.1.10.6.1.)

When an originating write occurs, either the *AttributeStamp* or the *LinkValueStamp* of the attribute's value is updated, but not both. This chart specifies the conditions under which each is updated.

Attribute type	Forest functional level	AttributeStamp associated with the attribute	LinkValueStamp associated with the attribute's values
Any type of attribute other than a forward link attribute	Any	Updated	Not updated
Forward link attribute	DS_BEHAVIOR_WIN2000	Updated	Not updated
Forward link attribute	Greater than DS_BEHAVIOR_WIN2000	Not updated	Updated

Whether an attribute value has an *AttributeStamp* or *LinkValueStamp* depends on the state at the time of the originating update. The data model does not require an attribute to have an *AttributeStamp* or *LinkValueStamp*. If an attribute has never had a value, it will not have an *AttributeStamp*.

A forward link attribute will have an *AttributeStamp* if it is updated when the forest functional level is DS_BEHAVIOR_WIN2000. However, if the forest functional level is changed to be greater than DS_BEHAVIOR_WIN2000, then any further updates will cause the attribute's value to have a *LinkValueStamp*. The previously associated *AttributeStamp* of the attribute will be left unchanged.

On the other hand, if the attribute is a forward link attribute that was never updated when the forest functional level was DS_BEHAVIOR_WIN2000, it will not have an associated *AttributeStamp*. If a value of the attribute is updated when the forest functional level is greater than DS_BEHAVIOR_WIN2000, the attribute value will have a *LinkValueStamp* and the attribute will still not have an *AttributeStamp*.

Let *o*!*a.stamp* denote the *AttributeStamp* associated with replicated attribute *a* on object *o*. When an originating update creates or modifies replicated attribute *a* on object *o*, the value of *o*!*a.stamp* is determined as follows:

- *dwVersion*: If the attribute did not exist on this object before the originating update (that is, an LDAP Add operation of this object, or an LDAP Modify operation creating the initial value of this attribute on this object), *dwVersion* equals one. Otherwise *dwVersion* equals *o*!*a.stamp.dwVersion* before the update, plus one.
- *timeChanged*: The time of the originating update, according to the system clock on this DC.
- *uuidOriginating*: the invocationId of the *dc's* nTDSDSA object.
- *usnOriginating*: *dc.usn*.

Once a replicated attribute exists on an object, it will continue to exist for the lifetime of the object, in order to carry the stamp. If all values have been removed from the attribute, the attribute will be absent from the LDAP perspective, but it remains present in the state model in order to preserve the stamp. If a value is added to *o*!*a* and *o*!*a.stamp* exists, even if *o*!*a* had no values before the addition, the value of *o*!*a.stamp*.*dwVersion* is used as described previously in creating the new stamp's *dwVersion*.

Let *o*!*a.r* denote a single link value *r* that is part of a replicated forward link attribute *a*, and let *o*!*a.r.stamp* denote the *LinkValueStamp* associated with this value. An originating update cannot modify a single link value *r* that is part of a forward link attribute, except to delete it or to re-create it. A link value *r* is deleted, but exists as a *tombstone*, if *r.stamp.timeDeleted* ≠ 0. When the current time minus *r.stamp.timeDeleted* exceeds the tombstone lifetime, the link value *r* is garbage-collected; that is, removed from its containing forward link attribute.

When an originating update creates a link value *r* of a forward link attribute *a* of object *o*, the *LinkValueStamp* *o*!*a.r.stamp* is computed as follows:

- *dwVersion*: 1.
- *timeChanged*: The time of the originating update, according to the system clock on this DC.
- *uuidOriginating*: the invocationId of *dc's* nTDSDSA object.
- *usnOriginating*: *dc.usn*.
- *timeCreated*: The time of the originating update, according to the system clock on this DC.
- *timeDeleted*: Zeros.

When an originating update re-creates a link value *r* of a forward link attribute *a* of object *o*, that is, a create occurs when the same link value exists as a tombstone, the *LinkValueStamp* *o*!*a.r.stamp* is computed as follows:

- *dwVersion*: *o*!*a.r.stamp.dwVersion* before the originating update, plus one.
- *timeChanged*: The time of the originating update, according to the system clock on this DC.
- *uuidOriginating*: the invocationId of *dc's* nTDSDSA object.
- *usnOriginating*: *dc.usn*.
- *timeCreated*: *o*!*a.r.stamp.timeCreated* before the originating update.
- *timeDeleted*: Zeros.

When an originating update deletes a link value *r* of a forward link attribute *a* of object *o*, the *LinkValueStamp* *o*!*a.r.stamp* is computed as follows:

- *dwVersion*: *o*!*a.r.stamp.dwVersion* before the originating update, plus one.
- *timeChanged*: The time of the originating update, according to the system clock on this DC.
- *uuidOriginating*: the invocationId of *dc's* nTDSDSA object.
- *usnOriginating*: *dc.usn*.
- *timeCreated*: *o*!*a.r.stamp.timeCreated* before the originating update.
- *timeDeleted*: The time of the originating update, according to the system clock on this DC.

The stamp values created by originating updates are used by protocols described in [MS-DRSR]. Some stamp values maintained in this state model are not used by those protocols; see [MS-DRSR] section 4.1.10.5.6 (FilterAttribute) for specifics on the stamps that are filtered out.

When all updates associated with an originating update request are complete, the variable *dc.usn* is increased by at least one. Between originating updates, the variable *dc.usn* does not decrease.

The effects of an originating update are captured in the state model by committing a transaction. When the originating update is initiated by a protocol request, such as an LDAP Modify, the transaction is committed before sending the appropriate protocol response. The transaction has the **ACID** properties [GRAY] and provides at least degree 2 isolation of concurrent read and update requests [GRAY].

Each read request is performed as a transaction. When multiple read requests are used to retrieve a large set of results, each request is its own transaction. Section 3.1.1.5 specifies the transaction boundaries that are used for all originating updates. To preview: An originating update is almost always performed as a single transaction; a few are processed as multiple transactions. In some cases, an originating update request will cause transactions to occur after the response has been sent; section 3.1.1.5 specifies all cases where processing of an update continues after the response.

The following example illustrates the effects of originating updates on stamp values. In this example, the forest functional level is assumed to be greater than DS_BEHAVIOR_WIN2000, so LinkValueStamps are used for updates to forward link attributes. In the example, stamp values are represented as lists whose elements are the elements of the stamp, in the order listed in the type definition. Thus *dwVersion* is always first, and *timeDeleted* is last in a *LinkValueStamp*. An *AttributeStamp* is placed between the attribute's lDAPDisplayName and the first value, if any. A *LinkValueStamp* is placed immediately following the link value.

This example shows the stamp values on two attributes of a single group object: the description attribute and the member attribute (a forward link attribute). In the initial state neither attribute is present.

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  . . .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
  )
)
```

An LDAP Modify adds a value for description. This DC's invocationId is 103, and its *usn* is 501 at the time of the originating update.

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  . . .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (description (1 0x2FA9A74EA 103 501) "QWERTY")
  )
)
```

An LDAP Modify adds a value for member. This originating update occurred one second after the previous one, with no updates in between. This pattern continues for the rest of this example.

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  . . .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (description (1 0x2FA9A74EA 103 501) "QWERTY")
    (member
      "<GUID=9>;<SID=0x0105...07080000>;
       cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com"
      (1 0x2FA9A74EB 103 502 0x2FA9A74EB 0) )
  )
)
```

An LDAP Modify removes the values of both description and member.

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  . . .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (description (2 0x2FA9A74EC 103 503) )
    (member
      "<GUID=9>;<SID=0x0105...07080000>;
       cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com"
      (2 0x2FA9A74EC 103 503 0x2FA9A74EB 0x2FA9A74EC) )
  )
)
```

An LDAP Modify sets member back to the value it had before the previous update. The stamp it receives is *not* what it had before.

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  . . .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (description (2 0x2FA9A74EC 103 503) )
    (member
      "<GUID=9>;<SID=0x0105...07080000>;
       cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com"
      (3 0x2FA9A74ED 103 504 0x2FA9A74EB 0) )
  )
)
```

Finally, an LDAP Modify sets description to a new value.

```
(
  "<GUID=5>;<SID=0x0105...94E1F2E6>;dc=microsoft,dc=com"
  . . .
  ( (objectGUID 6) (parent 2) (cn "DSYS")
    (objectClass top ... group)
    (name "DSYS") (rdnType cn)
    (objectSid 0x0105...94E1F2E60B080000)
    (description (3 0x2fa9a74ee 103 505) "SHRDLU")
    (member
      "<GUID=9>;<SID=0x0105...07080000>;
       cn=Peter Houston,ou=NTDEV,dc=microsoft,dc=com"
      (3 0x2FA9A74ED 103 504 0x2FA9A74EB 0) )
  )
)
```

3.1.1.1.10 GC Server

An AD DS DC can be a **GC server** as determined by state in the config NC, as specified in section 6.1.1.2.2.1.2.1.1. A GC server provides LDAP access to the GC view of the forest via a special LDAP port, as specified in section 3.1.1.3.

3.1.1.1.11 FSMO Roles

Each DC accepts originating updates for most attributes of most objects within its writable NC replicas. But certain updates are only accepted if the DC is the single designated "master" DC for the update, as specified in this section. The mechanism is called **FSMO roles**, which stands for **flexible single master operation (FSMO)** roles.

If some or all of the updates to an object are single-mastered, that object belongs to a defined set of objects. [MS-DRSR] section 4.1.10.5.3 (GetReplScope) specifies these sets, which are called FSMO roles. Each FSMO role is contained within a single NC. Each domain NC contains three FSMO roles called *InfrastructureMasterRole*, *RidAllocationMasterRole*, and *PdcEmulationMasterRole*. A config NC contains one FSMO role called *DomainNamingMasterRole*. A schema NC contains one FSMO role called *SchemaMasterRole*. An application NC has no FSMO roles.

Since a DC operating as AD LDS does not host domain NCs, it cannot own any of the three roles contained by domain NCs. It can own the Schema Master and Domain Naming FSMO roles.

In a given NC, each FSMO role is represented by an object. [MS-DRSR] section 4.1.10.5.3 (GetReplScope) specifies these objects, which are called **FSMO role objects**.

The fSMORoleOwner attribute of each FSMO role object is an object reference to the nTDSDSA object of the DC that owns the role; that is, the DC that performs updates to objects in the role. nTDSDSA objects and how they represent DCs are specified in section 6.1.

An originating update to an object within a FSMO role generates an LDAP referral if the DC that receives the request cannot perform the update; the referral is to the DC represented by the nTDSDSA object referenced by the FSMO role object's fSMORoleOwner attribute on the DC that received the request.

The processing of updates affected by FSMO roles is fully specified in section 3.1.1.5.

The IDL_DRSGetNCChanges method ([MS-DRSR] section 4.1.10) makes an originating update to the fSMORoleOwner attribute of a FSMO role object while preserving single-mastering of updates to the FSMO role. The ability to update the fSMORoleOwner attribute in this way is exposed through LDAP as the root DSE updates becomeDomainMaster, becomeInfrastructureMaster, becomePdc,

becomePdcWithCheckPoint, becomeRidMaster, and becomeSchemaMaster specified in section 3.1.1.3.

Reading the rootDSE attribute validFSMOs on a DC returns the set of all FSMO roles (represented as FSMO role objects) that the DC will update; this is specified in section 3.1.1.3.

3.1.1.1.12 Cross-NC Object References

Section 3.1.1.1.6 specifies the referential integrity behavior of attributes with object reference syntaxes. That section only specifies the case of references within a single NC. This section specifies the differences for the case of object references that cross an NC boundary.

Suppose *src* and *dst* are objects in different NCs, *src* has an attribute *a* with an object reference syntax, and *dc* is a DC hosting a writable replica of *src*'s NC.

- When an LDAP Add or Modify creates an object reference within attribute *src.a*, the server uses the DN (or SID or GUID) specified in the Add or Modify to locate an existing object *dst*. The behavior is identical to the single NC case, with two exceptions:

  1. Locating the object *dst* can fail if *dc* does not host a replica of *dst* and if *dc* fails to communicate with a server that hosts a *dst* replica; the response is error *unavailable* / *<unrestricted>*.

  2. Certain cross-NC references are not allowed; the specific references that are not allowed are specified in section 3.1.1.2.2.3. If the reference is not allowed, the response is error *constraintViolation* / *ERROR_DS_NAME_REFERENCE_INVALID*.

- After the assignment, the referential integrity behavior is the same as if the reference did not cross an NC boundary, except that reference *src.a* reflects the state of object *dst* at some time *t* in the past, not at the current time. If the distributed system of DCs in the forest is functioning normally, the difference between the current time and the time *t* of the previous sentence is bounded by an administrator-configurable amount of time. (During this period of time, between *t* and the current time, the cross-NC reference can refer to the object by its previous name or at its previous location, or it can refer to the object after the object has been deleted.) The phrase "functioning normally" shown previously means that the DCs are running and communicating as needed, with only transient failures.

The mechanism the system uses for restoring the integrity of object references is specified in section 3.1.1.6.

3.1.1.1.13 NC Replica Graph

This section uses directed graphs to model replication topology. Use [KNUTH1] section 2.3.4.2 as a reference for the terms *directed graph*, *vertex*, *arc*, *initial vertex*, *final vertex*, *path*, and *strongly-connected*.

This section introduces concepts that are used in specifying the KCC in section 6.2. The concepts are simplified here because this section ignores the SMTP replication transport [MS-SRPL] and RODCs. Section 6.2 specifies the concepts in full generality.

Associated with each NC replica is a repsFrom abstract attribute as specified in [MS-DRSR] section 5.169. The value of this attribute is a set of tuples. Each tuple contains a field *uuidDsa* that contains the objectGUID of an nTDSDSA object. The nTDSDSA object represents a DC as specified in section 6.1.

Given a forest and an NC within the forest, define the **NC replica graph** as follows:

- Each DC of the given forest is a vertex of the directed graph.
- For each DC *d* containing a replica of the given NC:
  - Set *r* to the given NC's repsFrom on the DC *d*, as a sequence in any order.
  - For *i* in [0 .. *r.length*-1]:
    - *r[i].uuidDsa* is a directed arc to *d* (the final vertex of the arc) from the DC represented by the nTDSDSA object with objectGUID = *r[i].uuidDsa* (the initial vertex of the arc).

Each arc in the directed graph represents a replication relationship. The DC at the final vertex of an arc performs **cycles** of IDL_DRSGetNCChanges requests ([MS-DRSR] section 4.1.10.1) to the DC at the initial vertex of that arc, applying the results of these requests to update the replica of the given NC at the final vertex. The events that trigger a cycle of IDL_DRSGetNCChanges request over a given arc of the NC replica graph are specified in the next section.

The KCC is an automated management component of Active Directory that controls the repsFrom values on each DC and thereby controls the NC replica graph for each NC. One of the KCC's goals is to keep each NC replica graph of the forest in a good state, defined as follows:

1. Each DC in the NC replica graph contains a replica of the given NC.
2. If the DC at the initial vertex of an arc contains a partial replica of the given NC, so does the DC at the final vertex of that arc.
3. If *d* is any DC that contains a partial replica of the given NC, there is a path to *d* from some DC that contains a full replica of the given NC.
4. Define *F* as the set of all DCs that contain full replicas of the given NC. The subgraph of the NC replica graph whose vertex set is *F* is strongly-connected.

For example, the following NC replica graph contains five DCs. DC 1, DC 2, and DC 3 contain full replicas of the given NC and DC 4 and DC 5 contain partial replicas of the given NC.

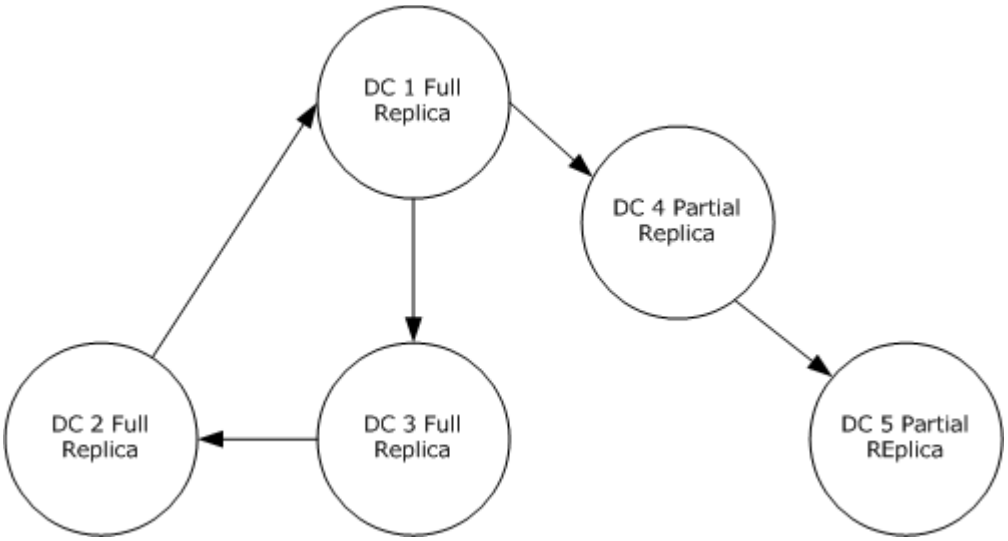


Figure 3: A sample NC replica graph

Per item 1 in the numbered list above, every DC present in the graph contains a replica of the given NC.

There is an arc from DC 4 to DC 5. DC 4 is the initial vertex of this arc and DC 5 is the final vertex. Per item 2 in the list above, because DC 4 contains a partial replica of the NC, DC 5 also contains a partial replica of the NC.

Per item 3 in the list above, there is a path from DC 1, which contains a full replica of the NC, to both DC 4 and DC 5 that contains a partial replica of the NC.

Per item 4 in the list above, the subgraph of the NC replica graph made by DC 1, DC 2, and DC 3 that contains a full replica of the NC is strongly connected because there is a path from each vertex in the subgraph to every other vertex in the subgraph.

The KCC performs this management by first creating *connection* objects (specified in section 6.1.1.2.2.1.2.1.2), then creating repsFrom values from those connection objects (specified in section 6.2). An administrator can create specially marked connection objects, with the NTDSCONN_OPT_IS_GENERATED bit not set in the options attribute, that the KCC will not modify but will use in creating repsFrom values.

3.1.1.1.14 Scheduled and Event-Driven Replication

If *client* and *server* are two DCs in the NC replica graph of a given NC and forest, where *server* is the initial vertex of an arc and *client* is the final vertex of the same arc, *client* will perform a replication cycle from *server* by calling IDL_DRSGetNCChanges ([MS-DRSR] section 4.1.10) until the cycle is complete in either of these two cases:

1. The DC *client's* repsFrom tuple for *server* contains a *schedule* field that calls for replication at the current time. The *schedule* contains a REPLTIMES structure as specified in [MS-DRSR] section 5.164. This is *scheduled replication*.

2. The DC *server* calls the IDL_DRSReplicaSync method ([MS-DRSR] section 4.1.23.2) on the *client*. This is *event-driven replication*. The events that cause this form of replication are specified later in this section.

A precondition for event-driven replication involves *server's* repsTo abstract attribute, specified in [MS-DRSR] section 5.170. The repsTo abstract attribute is a sequence tuples, like repsFrom. Like repsFrom, each repsTo tuple contains a field *uuidDsa* that contains the objectGUID of an nTDSDSA object. The nTDSDSA object represents a DC as specified in section 6.1. If *server's* repsTo abstract attribute contains a tuple whose *uuidDsa* field contains the objectGUID of client's nTDSDSA object, *server* performs event-driven replication to *client*.

It remains to specify how a DC's repsTo abstract attribute is populated, and to specify the events that trigger event-driven replication.

A DC's repsTo abstract attribute is populated as follows:

1. A DC *server's* repsTo abstract attribute is populated for event-driven replication to *client* if *client's* repsFrom tuple for *server* has the DRS_ADD_REF bit set in its *replicaFlags* field, and *client* calls the IDL_DRSGetNCChanges method on *server* during scheduled replication. The DC *client* sets the DRS_ADD_REF bit in *Request.ulFlags* on the scheduled call to IDL_DRSGetNCChanges on *server* ([MS-DRSR] section 4.1.10.4.1) and *server* updates repsTo for event-driven replication to *client* as a result ([MS-DRSR] section 4.1.10.5.2).

   Since the KCC running on *client* writes *client's* repsFrom, this behavior is controlled by the state of KCC objects as specified in section 6.2.

2. A DC *server's* repsTo abstract attribute is populated for event-driven replication to DC *client* if the IDL_DRSReplicaAdd method ([MS-DRSR] section 4.1.19.2) is called on *client*, specifying *server* as the replication source (either *pmsgIn.V1.pszSourceDsaAddress* or *pmsgIn.V2.pszDsaSrc*, depending upon the request version used). If the IDL_DRSReplicaAdd adds a new tuple to *client's* repsFrom, it proceeds to call IDL_DRSUpdateRefs ([MS-DRSR] section 4.1.26.2) on *server* to update *server's* repsTo abstract attribute.

   Since IDL_DRSReplicaAdd is an RPC method, this behavior is controlled by any authorized requester of this method. Within Active Directory itself, IDL_DRSReplicaAdd is called by the KCC to maintain repsFrom.

The events that trigger event-driven replication from a DC *server* are as follows:

1. The DC *server* receives an update, either originating or replicated, as specified in section 3.1.1.5.1.7 (Urgent replication).

2. A configurable time expires after DC server receives any update, as specified in section 3.1.1.5.1.6 (Replication notification).

3.1.1.1.15 Replication Latency and Tombstone Lifetime

Replication latency is the delay between the time of an originating update to an NC and the time when this update is reflected in all replicas of that NC. Some updates are superseded before reaching all replicas, but for the purposes of this simplified definition, consider an attribute update that is not followed by other updates to that attribute for a long time.

Administrators of Active Directory control replication latency by setting several variables, specified in section 6.1 and section 6.2. These variables ultimately control the schedules used for scheduled replication, and they control the use of event-driven replication. Replication latency is not fully predictable in a real system, because it depends upon the volume of read requests to DCs, the volume of originating update requests to DCs, and the availability of DCs and communications links.

If the typical replication latency is larger than the tombstone lifetime (the value of the tombstoneLifetime attribute of the Directory Services object specified in section 6.1.1.2.4.1.1, interpreted as a number of days), some tombstones or recycled-objects will be garbage collected before they have replicated to every NC replica. As a result, some objects will never be deleted in some replicas. To restore consistency of object existence, an administrator cleans up such *lingering objects* with utility programs.

3.1.1.1.16 Delayed Link Processing

When an update to an object would result in removal of more than 10,000 **forward link values**, or the update would result in more than 10,000 forward link values to be made either visible or invisible to LDAP operations that do not specify the LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID control, then at least 10,000 of the value changes so directed are completed within the transaction encompassing the modification (that is, the "originating transaction").

Note In Windows Server 2003 operating system, Windows Server 2003 R2 operating system, and Windows Server 2008 operating system, the number is 1,000 instead of 10,000.

Any values not so changed within the originating transaction are changed by continuing processing after and outside of that originating transaction. These changes that occur outside the originating transactions are called "delayed link processing". Delayed link processing occurs within one or more transactions subsequent to the originating transaction.

Although delayed link processing always uses at least one subsequent transaction, there is no constraint on the upper bound of the number of transactions that Active Directory uses during delayed link processing. Therefore, there is no requirement that at any given time all such values have been removed, made visible, or made invisible. It is possible that there is a period of time during which an object that should not have a specific value for a link valued attribute will continue to have that value. Likewise, it is possible that there is a period of time during which an object that should have a specific value for a link valued attribute be either visible or invisible might not have that value in the correct state. Although the protocol places no boundary or requirements on the length of this period of time, it is recommended that implementations minimize the length of this period of time to improve usability of the directory for clients.

The server MUST guarantee that all such changes to values of link valued attributes are eventually made to all affected link valued attributes.

Note In Windows 2000 Server operating system, delayed link processing is not supported.

3.1.1.2 Active Directory Schema

In Active Directory, the schema contains definitions for the objects that can be stored in the directory, and it enforces the rules that govern both the structure and the content of the directory. The schema consists of a set of classes, attributes, and syntaxes. A *class* is a category of objects that share a set of common characteristics. It is a formal description of a discrete, identifiable type of object that can be stored in the directory. Each object in the directory is an instance of one or more classes in the schema. Attributes define the types of information that an object can hold. For each class, the schema specifies the mandatory attributes and optional attributes that constitute the set of shared characteristics of the class. A syntax is the data type of a particular attribute. Syntaxes determine what data type an attribute can have. Active Directory uses a set of predefined syntaxes. The predefined syntaxes do not actually appear in the directory, and new syntaxes cannot be added.

The schema itself is represented in Active Directory by a set of objects known as *schema objects*. For each class in the schema, there is a schema object that defines the class. This object is a classSchema object. For each attribute in the schema, there is a schema object that defines the attribute. This object is an attributeSchema object. Therefore, every class is actually an instance of the classSchema class, and every attribute is an instance of the attributeSchema class. Administrators and applications can extend the schema by adding new attributes and classes and by modifying existing ones.

A schema object cannot be deleted, but it can be made **defunct** by setting the isDefunct attribute to true. A schema object that is not defunct is **active**. The primary effect of the defunct state is to prevent the schema object from being used in the creation or modification of new objects. For instance, attempts to perform an LDAP Add of an object with a defunct class fails, just as an attempt to perform an LDAP Add of a nonexistent class fails. The full effects of the defunct state are specified later in this section.

3.1.1.2.1 Schema NC

The schema NC contains all of the objects that define object classes and attributes used in a forest.

The root object of the schema NC, called the **schema container**, is an instance of class dMD.

The contents of the schema NC is established when a forest is created. To enable a DC of a forest to be upgraded to a newer version of Windows Server operating system, a *schema upgrade* process is first performed. This process updates the portion of the schema that Windows Server depends upon.

The attribute objectVersion on the schema container object stores the schema version of the forest. This attribute is set during the creation of the first domain in a forest and is changed during schema

upgrade after the schema is successfully upgraded to a newer version. In AD DS, to add a DC running a particular Windows Server version to an existing forest, the objectVersion of the forest's schema container must be greater than or equal to the value for that Windows Server version. In AD LDS, this is not a requirement. In AD LDS, to add a DC running a particular Windows Server version to an existing forest, the objectVersion of the forest's schema container may be less than the value for that Windows Server version. The correspondence between Windows Server versions and values of the schema container objectVersion is:

- Windows 2000 Server operating system: 13
- Windows Server 2003 operating system: 30
- Windows Server 2003 R2 operating system: 31
- Windows Server 2008 operating system (AD DS): 44
- Windows Server 2008 R2 operating system (AD DS): 47
- Windows Server 2012 operating system (AD DS): 56
- Windows Server 2012 R2 operating system (AD DS): 69
- Active Directory Application Mode (ADAM): 30
- Windows Server 2008 (AD LDS): 30
- Windows Server 2008 R2 (AD LDS): 31
- Windows Server 2012 (AD LDS): 31
- Windows Server 2012 R2 (AD LDS): 31

Attribute schemaInfo on the schema container stores a String(Octet) value of length 21 bytes. This attribute is updated on every original schema Add or Modify in the same transaction, and it is replicated to all the domain controllers in the forest upon completion of schema NC replication. The first byte of schemaInfo is 0xFF. The next 4 bytes are a 32-bit integer in big-endian byte order, used as the version of the update. The last 16 bytes are the invocationId of the DC where the schema change is made. The version starts from 1 for a new forest. Once a schema change is done, the version is incremented by one, and the invocationId of the DC where the schema change is done is written into the GUID part of the string. The invocationId attribute is specified in section 3.1.1.1.9.

For example, here is a value of schemaInfo:

0xFF 0x00 0x00 0x07 0xC7 0x20 0x79 0x92 0xE6 0x84 0xB6 0xF6 0x40 0x99 0x47 0x21 0x8B 0xC9 0xE0 0xF1 0xF3

After a schema change is done on the schema master, the following is the new value:

0xFF 0x00 0x00 0x07 0xC8 0x20 0x79 0x92 0xE6 0x84 0xB6 0xF6 0x40 0x99 0x47 0x21 0x8B 0xC9 0xE0 0xF1 0xF3

There is a child of the schema container with RDN cn=Aggregate and class subSchema. This object has several constructed attributes that are compliant with [RFC2251] section 4.5.2, through which the client can retrieve the forest's current schema. See constructed attributes in section 3.1.1.4.5. This object cannot be modified.

3.1.1.2.2 Syntaxes

3.1.1.2.2.1 Introduction

This section describes the LDAP syntaxes used in attributes in Active Directory DCs.

3.1.1.2.2.2 LDAP Representations

The LDAP syntaxes supported by DCs are as shown in the following table. The set of syntaxes supported is not extensible by schema modifications. Each syntax is identified by the combination of the attributeSyntax, oMSyntax and, in select cases, oMObjectClass attributes of an attributeSchema object. The cases for which oMObjectClass is not used are indicated by the presence of a hyphen in the oMObjectClass column in the table. The combinations shown in the following table are exhaustive; this table is consistent and identical for Windows 2000 Server operating system, Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system.

While oMObjectClass conceptually contains an **object identifier (OID)**, it is declared in the schema as String(Octet) syntax, requiring that values read from and written to it be expressed as the **Basic Encoding Rules (BER)** encoding of the OID (BER encoding is defined in [ITUX690]). In the table, both the BER-encoded form and the dotted string form of the OID are given.

LDAP syntax name	attributeSyntax	oMSyntax	oMObjectClass
Boolean	2.5.5.8	1	-
Enumeration	2.5.5.9	10	-
Integer	2.5.5.9	2	-
LargeInteger	2.5.5.16	65	-
Object(Access-Point)	2.5.5.14	127	0x2B 0x0C 0x02 0x87 0x73 0x1C 0x00 0x85 0x3E (1.3.12.2.1011.28.0.702)
Object(DN-String)	2.5.5.14	127	0x2A 0x86 0x48 0x86 0xF7 0x14 0x01 0x01 0x01 0x0C (1.2.840.113556.1.1.1.12)
Object(OR-Name)	2.5.5.7	127	0x56 0x06 0x01 0x02 0x05 0x0B 0x1D (2.6.6.1.2.5.11.29)
Object(DN-Binary)	2.5.5.7	127	0x2A 0x86 0x48 0x86 0xF7 0x14 0x01 0x01 0x01 0x0B (1.2.840.113556.1.1.1.11)
Object(DS-DN)	2.5.5.1	127	0x2B 0x0C 0x02 0x87 0x73 0x1C 0x00 0x85 0x4A (1.3.12.2.1011.28.0.714)
Object(Presentation-Address)	2.5.5.13	127	0x2B 0x0C 0x02 0x87 0x73 0x1C 0x00 0x85 0x5C (1.3.12.2.1011.28.0.732)
Object(Replica-Link)	2.5.5.10	127	0x2A 0x86 0x48 0x86 0xF7 0x14 0x01 0x01 0x01 0x06 (1.2.840.113556.1.1.1.6)
String(Case)	2.5.5.3	27	-
String(IA5)	2.5.5.5	22	-

LDAP syntax name	attributeSyntax	oMSyntax	oMObjectClass
String(NT-Sec-Desc)	2.5.5.15	66	-
String(Numeric)	2.5.5.6	18	-
String(Object-Identifier)	2.5.5.2	6	-
String(Octet)	2.5.5.10	4	-
String(Printable)	2.5.5.5	19	-
String(Sid)	2.5.5.17	4	-
String(Teletex)	2.5.5.4	20	-
String(Unicode)	2.5.5.12	64	-
String(UTC-Time)	2.5.5.11	23	-
String(Generalized-Time)	2.5.5.11	24	-

The representation for many of the preceding syntaxes is adopted from [RFC2252]. The following table lists the syntaxes whose representation is adopted from that RFC, the [RFC2252] name of that syntax, and the associated section of [RFC2252] that specifies the representation.

LDAP syntax name	RFC 2252 name	Section of RFC 2252
Boolean	Boolean	6.4
Enumeration	INTEGER	6.16
Integer	INTEGER	6.16*
LargeInteger	INTEGER	6.16*
Object(DS-DN)	DN	6.9 (see also [RFC2253])**
Object(Presentation-Address)	Presentation Address	6.28***
Object(Replica-Link)	Binary	6.2
String(IA5)	IA5 String	6.15†
String(Numeric)	Numeric String	6.23††
String(Object-Identifier)	OID	6.25†††
String(Octet)	Binary	6.2
String(Printable)	Printable String	6.29††††
String(Unicode)	Directory String	6.10
String(UTC-Time)	UTC Time	6.31†††††
String(Generalized-Time)	Generalized Time	6.14†††††

* The Integer syntax in Active Directory is restricted to 32-bit integers. The LargeInteger syntax is restricted to 64-bit integers.

** While Active Directory uses the [RFC2252] and [RFC2253] representation of DNs, it can also use alternative forms of the DN representation when it accepts requests and sends responses, if requested by the client. This is documented in LDAP_SERVER_EXTENDED_DN_OID (section 3.1.1.3.4.1.5).

*** No validation is done by the DC to confirm that the value conforms to the representation specified in [RFC1278].

† Values restricted to ASN.1 IA5 strings (as specified in [ITUX680]).

†† Values restricted to ASN.1 Numeric strings (as specified in [ITUX680]).

††† Values of attributes of syntax String(OID) are accepted in either the numericoid (numeric OID) or descr (the LDAP display name of the attribute or class identified by that OID) format, as defined in [RFC2252] section 4.1. The server determines the format of returning OID values using the first matching rule in the following set of processing rules:

1. If a "Binary Option" is present on the AttributeDescription (as described in [RFC2251] section 4.1.5.1) of the request, the server MUST return the OID converted to binary format as described in [RFC2252] section 4.3.1. The result is a binary encoded value using Basic Encoding Rules defined in [ITUX690].

2. If a value of either attributeID of an AttributeSchema object or governsID of a ClassSchema object is requested, the server MUST return the OID in numericoid (Numeric OID) format.

3. If the attribute requested is not attributeID or governsID, but the value of the attribute identifies an attribute or class, the server MUST return the value in Descr format.

4. If none of the above applies, the server MUST return the OID in numericoid (Numeric OID) format.

†††† Active Directory has two differences from the character set specified in [RFC2252]:

1. The quote character ("), or ASCII 0x22, is part of the character set in the RFC but not in Active Directory.

2. The "@" symbol, or ASCII 0x40, is not part of the character set in the RFC, but it is part of the character set in Active Directory.

††††† Times are measured in granularity of 1 second.

The remaining syntaxes are represented as shown in the following sections.

3.1.1.2.2.2.1 Object(DN-String)

A value with this syntax is a UTF-8 string in the following format:

S:**byte_count**:**string_value**:**object_DN**

where **byte_count** is the number (in decimal) of bytes in the **string_value** string, **object_DN** is a DN in Object(DS-DN) form, and all remaining characters are string literals. Since **string_value** is a UTF-8 string, one character can require more than one byte to represent it.

3.1.1.2.2.2.2 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, **object_DN** is a DN in Object(DS-DN) form, and all remaining characters are string literals.

3.1.1.2.2.2.3 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 (in decimal) of hexadecimal digits in **binary_value**, **binary_value** is the hexadecimal representation of a binary value, **object_DN** is a DN in Object(DS-DN) form, and all remaining characters are string literals. Each byte is represented by a pair of hexadecimal characters in **binary_value**, with the first character of each pair corresponding to the most-significant nibble of the byte. The first pair in **binary_value** corresponds to the first byte of the binary value, with subsequent pairs corresponding to the remaining bytes in sequential order. Note that **char_count** is always even in a syntactically-valid Object(DN-Binary) value.

3.1.1.2.2.2.4 Object(OR-Name)

A value with this syntax is a UTF-8 string in the following format:

object_DN

where **object_DN** is a DN in Object(DS-DN) form.

3.1.1.2.2.2.5 String(Case)

A value with this syntax is a case-sensitive UTF-8 string, but the server does not enforce that a value of this syntax must be a valid UTF-8 string.

3.1.1.2.2.2.6 String(NT-Sec-Desc)

A value with this syntax contains a **Windows security descriptor** in binary form. The binary form is that of a SECURITY_DESCRIPTOR structure and is specified in [MS-DTYP] section 2.4.6. It is otherwise encoded the same as the String(Octet) syntax.

3.1.1.2.2.2.7 String(Sid)

A value with this syntax contains a SID in binary form. The binary form is that of a SID structure (the SID structure is specified in [MS-DTYP] section 2.4.2.2; all multibyte fields have little-endian byte ordering). It is otherwise encoded the same as the String(Octet) syntax.

3.1.1.2.2.2.8 String(Teletex)

A value with this syntax is a UTF-8 string restricted to characters with values between 0x20 and 0x7E, inclusive.

3.1.1.2.2.3 Referential Integrity

Attributes with object reference syntaxes have special behavior, called *referential integrity*, as specified in section 3.1.1.1.6. The following are object reference syntaxes:

- Object(Access-Point)
- Object(DN-String)
- Object(OR-Name)
- Object(DN-Binary)
- Object(DS-DN)

For the four syntaxes other than Object(DS-DN), referential integrity only applies to the **object_DN** portion of the value.

Active Directory imposes restrictions on which objects can be referenced by an attribute that has referential integrity. An attribute can reference any object in the same NC as the object on which that attribute is located. Additionally, attributes on an object in the domain NC, schema NC, or config NC can reference any object in any domain NC in the forest, any object in the schema NC or the config NC, or the root object of any application NC. For objects in application NCs, such attributes can reference any object in the config NC or the schema NC, or the root object of any application NC, in addition to any object in the same application NC as the object doing the referencing. All other references are disallowed by the server.

These restrictions are identical for AD DS and for AD LDS. Because AD LDS does not support domain NCs, the only cross-NC references in an AD LDS forest are from any NC to any object in the config and schema NCs or to the root of an application NC.

3.1.1.2.2.4 Supported Comparison Operations

In addition to determining what can be stored in an attribute, the syntaxes determine what comparison operations the server permits on an attribute in an LDAP search filter, as well as how the server performs those comparisons. The following table maps each of the LDAP syntaxes to a comparison rule. All syntaxes of the same comparison rule support the same comparison operations and are compared using the same comparison rules.

LDAP syntax	Comparison rule
Boolean	*Bool*
Enumeration	*Integer*
Integer	*Integer*
LargeInteger	*Integer*
Object(Access-Point)	*DN-String*
Object(DN-String)	*DN-String*
Object(OR-Name)	*DN-Binary*
Object(DN-Binary)	*DN-Binary*

LDAP syntax	Comparison rule
Object(DS-DN)	*DN*
Object(Presentation-Address)	*PresentationAddress*
Object(Replica-Link)	*Octet*
String(Case)	*CaseString*
String(IA5)	*CaseString*
String(NT-Sec-Desc)	*SecDesc*
String(Numeric)	*CaseString*
String(Object-Identifier)	*OID*
String(Octet)	*Octet*
String(Printable)	*CaseString*
String(Sid)	*Sid*
String(Teletex)	*NoCaseString*
String(Unicode)	*UnicodeString*
String(UTC-Time)	*Time*
String(Generalized-Time)	*Time*

The following table (split into three parts for readability) shows which of the choices in an LDAP filter (that is, which comparison operations) are supported for each comparison rule. The LDAP filter structure is defined in [RFC2251] section 4.5.1. Each comparison rule (for example, the rule for comparing two Bool values) is discussed following the table. The "and", "or", and "not" choices in an LDAP filter are not included in this table because they are not comparisons performed against an attribute value. Active Directory treats approxMatch as equivalent to equalityMatch. For details on the three extensible matching rules, see section 3.1.1.3.4.4.

Comparison rule	present	equalityMatch	approxMatch
Bool	*X*	*X*	*X*
Integer	*X*	*X*	*X*
DN-String	*X*	*X*	*X*
DN-Binary	*X*	*X*	*X*
DN	*X*	*X*	*X*
PresentationAddress	*X*	*X*	*X*
Octet	*X*	*X*	*X*
CaseString	*X*	*X*	*X*
SecDesc	*X*		

Comparison rule	present	equalityMatch	approxMatch
OID	*X*	*X*	*X*
Sid	*X*	*X*	*X*
NoCaseString	*X*	*X*	*X*
UnicodeString	*X*	*X*	*X*
Time	*X*	*X*	*X*

Comparison rule	lessOrEqual	greaterOrEqual	substrings
Bool	*X*	*X*	
Integer	*X*	*X*	
DN-String			
DN-Binary			
DN			
PresentationAddress			
Octet	*X*	*X*	*X*
CaseString	*X*	*X*	*X*
SecDesc			
OID			
Sid	*X*	*X*	*X*
NoCaseString	*X*	*X*	*X*
UnicodeString	*X*	*X*	*X*
Time	*X*	*X*	

Note In the following table, the constant names in the headers for the extensibleMatch columns are prefixed with "LDAP_MATCHING_RULE_". For example, "...BIT_AND" is actually "LDAP_MATCHING_RULE_BIT_AND".

Comparison rule	extensibleMatch: ...BIT_AND	extensibleMatch: ...BIT_OR	extensibleMatch: ...TRANSITIVE_EVAL
Bool			
Integer	*X*	*X*	
DN-String			*X**
DN-Binary			*X**

Comparison rule	extensibleMatch: ...BIT_AND	extensibleMatch: ...BIT_OR	extensibleMatch: ...TRANSITIVE_EVAL
DN			*X**
PresentationAddress			
Octet			
CaseString			
SecDesc			
OID			
Sid			
NoCaseString			
UnicodeString			
Time			

* Supported only if the attribute is a link attribute. Evaluates to Undefined otherwise.

3.1.1.2.2.4.1 Bool Comparison Rule

A value of true is considered to be greater than a value of false.

3.1.1.2.2.4.2 Integer Comparison Rule

A signed comparison of integer values is performed.

3.1.1.2.2.4.3 DN-String Comparison Rule

Values of String(DN-String) or String(Access-Point) are equal if the **object_DN** components name the same object and the **string_value** or **presentation_address** components are equal according to the UnicodeString comparison rule.

Evaluation of an LDAP_MATCHING_RULE_TRANSITIVE_EVAL matching rule is performed as documented in section 3.1.1.3.4.4. Only the **object_DN** component is considered when evaluating a filter clause that uses this rule; **string_value** or **presentation_address** is ignored.

3.1.1.2.2.4.4 DN-Binary Comparison Rule

Values of String(DN-Binary) or String(OR-Name) are equal if the **object_DN** components name the same object and the **binary_value** or **OR_address** components are identical in length and in content.

Evaluation of an LDAP_MATCHING_RULE_TRANSITIVE_EVAL matching rule is performed as documented in section 3.1.1.3.4.4. Only the **object_DN** component is considered when evaluating a filter clause that uses this rule; **binary_value** or **OR_address** is ignored.

3.1.1.2.2.4.5 DN Comparison Rule

DN values are equal when they name the same object.

Evaluation of an LDAP_MATCHING_RULE_TRANSITIVE_EVAL matching rule is performed as documented in section 3.1.1.3.4.4.

3.1.1.2.2.4.6 PresentationAddress Comparison Rule

Two Object(Presentation-Address) values are equal when they have the same length and content.

3.1.1.2.2.4.7 Octet Comparison Rule

Two values are equal when they are the same length and have identical contents. A value S1 is less than a value S2, where L is the smaller of the length of S1 and the length of S2, if either the first L bytes of S1 are less than the first L bytes of S2, or if the first L bytes of S1 and S2 are identical but the length of S1 is less than the length of S2. Given L = 1, S1 is less than S2 if the value of the first byte of S1 is less than the value of the first byte of S2. Given L > 1, for the first L bytes of S1 to be less than the first L bytes of S2 means that there exists an N (where N<L) such that bytes 0...N-1 of S1 and S2 are identical, and byte N of S1 is less than byte N of S2.

For substring purposes, each byte in the value is treated as if it was a character. Values are compared using the ordinary rules for a SubstringFilter, as defined in [RFC2251] section 4.5.1. The "characters" are treated as if they were case-sensitive; that is, two characters are considered identical if and only if the bytes that represent them are identical.

3.1.1.2.2.4.8 CaseString Comparison Rule

When compared using this comparison rule, two values are equal if they have identical length and contents. A value S1 is less than a value S2, where L is the smaller of the length of S1 and the length of S2, if either the first L bytes of S1 are less than the first L bytes of S2, or if the first L bytes of S1 and S2 are identical but the length of S1 is less than the length of S2. Given L = 1, S1 is less than S2 if the value of the first byte of S1 is less than the value of the first byte of S2. Given L > 1, for the first L bytes of S1 to be less than the first L bytes of S2 means that there exists an N (where N<L) such that bytes 0...N-1 of S1 and S2 are identical, and byte N of S1 is less than byte N of S2.

For substring purposes, this comparison rule treats values as if they were case-sensitive strings of characters and obey the ordinary rules for a SubstringFilter, as defined in [RFC2251] section 4.5.1. In this comparison, two characters are considered identical if and only if the bytes that represent them are identical.

3.1.1.2.2.4.9 SecDesc Comparison Rule

SecDescs are compared as octet strings as in section 3.1.1.2.2.4.7.

3.1.1.2.2.4.10 OID Comparison Rule

Two String(Object-Identifier) values are equal when they are the same OID.

3.1.1.2.2.4.11 Sid Comparison Rule

String(SID) values are treated as the binary representation of the SID (see section 3.1.1.2.2.2.7). The binary representations of the SID are compared using the Octet comparison rule.

3.1.1.2.2.4.12 NoCaseString Comparison Rule

This comparison rule is identical to the CaseString comparison rule, except that for each comparison, characters are treated in a case-insensitive fashion. For equality, ordering (greater-

than-or-equals and less-than-or-equals), and substrings, two characters are identical if the bytes that represent them are identical or if the characters differ from each other only by their case. The "C" locale, as defined in [ISO/IEC-9899], is used for determining whether two characters differ by case.

3.1.1.2.2.4.13 UnicodeString Comparison Rule

Comparison of values using this comparison rule is performed via Unicode comparison, which is specified in section 6.5. If an LDAP_SERVER_SORT_OID extended control (see section 3.1.1.3.4) is attached to the search request and specifies a locale in its orderingRule field, the locale specified is used for the Unicode comparison. Otherwise, the Unicode comparison is performed using United States English (LCID 0409). The comparison function is independent of the server locale and therefore gives the same result on all DCs. The comparison function operates on Unicode strings containing characters from all alphabets and does not, for instance, involve reducing the string to the alphabet used by United States English before performing the comparison. This comparison function is used to determine both equality and ordering (greater-than-or-equals and less-than-or-equals), as well as to determine equality of substrings when performing a substring comparison.

This comparison rule is used in processing search filters, *not* in sorting search results. See section 3.1.1.3.4.1.13 for per-locale sorting of search results.

3.1.1.2.2.4.14 Time Comparison Rule

Time T1 is greater than time T2 if T1 denotes a time subsequent to T2.

3.1.1.2.3 Attributes

The attributes of class attributeSchema are specified in the following table.

The term "Unique" (in quotation marks) in the following table, and in the similar table for classSchema in section 3.1.1.2.4.8, means that the value satisfies the following constraint:

- If the forest functional level is less than DS_BEHAVIOR_WIN2003, the value is unique among all values of this attribute in the set containing every attributeSchema and classSchema object in the schema NC.
- If the forest functional level is DS_BEHAVIOR_WIN2003 or greater, the value is unique among all values of this attribute in the set containing every attributeSchema and classSchema object S in the schema NC that satisfies at least one of the following three conditions:
  - S!isDefunct ≠ true, that is, S is active.
  - FLAG_ATTR_IS_RDN is present in S!systemFlags (defined in the following table).
  - S = C!rDNAttID (section 3.1.1.2.4.8) for some classSchema object C.

The term system-only in the following table means that the attribute is defined with systemOnly true. The value of the system-only attributes in the table can be specified on Add (except where noted) but cannot be modified on existing objects by LDAP Modify requests (except as specified in section 3.1.1.5.3.2), only by the system. The table is ordered with the system-only attributes before the other attributes.

Attribute	Description
objectClass	Equals the sequence [top, classSchema]. System-only.

Attribute	Description
attributeID	"Unique" OID that identifies this attribute. System-only.
schemaIDGUID	"Unique" GUID that identifies this attribute, used in **security descriptors (SDs)**. If not specified on Add, the DC generates a fresh GUID. System-only.
msDS-IntId	Not specified on Add (if specified in the Add request, the DC returns error *unwillingToPerform* / *<unrestricted>*); the value (a 32-bit unsigned integer in the subrange [0x80000000..0xBFFFFFFF]) is generated by the DC. Present on attributeSchema objects added when forest functional level is DS_BEHAVIOR_WIN2003 or greater with FLAG_SCHEMA_BASE_OBJECT not present in systemFlags (below). The value of msDS-IntId is the ATTRTYP of this attributeSchema object. Unique among all values of this attribute on objects in the schema NC, regardless of forest functional level. System-only.
linkID	Optional. If present, and not zero, this is a link attribute, and the linkID value is unique among all values of this attribute on objects in the schema NC, regardless of forest functional level. If linkID is even, the attribute is a forward link attribute; otherwise it is a back link attribute. The linkID for back link attribute equals to the linkID of the corresponding forward link attribute plus one. Special auto-generation behavior for the linkID attribute is specified in section 3.1.1.2.3.1. System-only.
mAPIID	Optional. "Unique" integer that identifies this attribute, used by **Messaging Application Programming Interface (MAPI)** clients. Not present on attributeSchema objects in AD LDS. Special auto-generation behavior for the mAPIID attribute is specified in section 3.1.1.2.3.2. System-only. If the DC functional level is DS_BEHAVIOR_WIN2008 or greater, the mAPIID attribute can be modified on attributeSchema objects that do not include FLAG_SCHEMA_BASE_OBJECT as the systemFlags attribute. Otherwise, the mAPIID attribute cannot be modified.
attributeSyntax	One of the three attributes that identify the syntax of the attribute. See section 3.1.1.2.2. System-only.
oMSyntax	One of the three attributes that identify the syntax of the attribute. See section 3.1.1.2.2. System-only.
oMObjectClass	Optional. One of the three attributes that identify the syntax of the attribute. See section 3.1.1.2.2. System-only.
isSingleValued	True if this attribute is single-valued; false, if it is multivalued. If an attribute is multivalued, all values have the syntax specified for the attribute. System-only.
systemFlags	Optional. Flags that determine specific system operations; see section 2.2.10 for values. The systemFlags values specific to an attributeSchema object are: FLAG_ATTR_NOT_REPLICATED: This attribute is nonreplicated. FLAG_ATTR_REQ_PARTIAL_SET_MEMBER: This attribute is a member of PAS regardless the value of attribute isMemberOfPartialAttributeSet. FLAG_ATTR_IS_CONSTRUCTED: This attribute is a constructed attribute. FLAG_ATTR_IS_OPERATIONAL: This attribute is an operational attribute,

Attribute	Description
	as defined in [RFC2251] section 3.2.1. FLAG_SCHEMA_BASE_OBJECT: This class is part of the base schema. Modifications to a base schema object are restricted as described in section 3.1.1.2.5. FLAG_ATTR_IS_RDN: This attribute can be used as an RDN attribute of a class. System-only.
systemOnly	Optional. The value of a system-only attribute cannot be modified on existing objects by LDAP Modify requests (except as specified in section 3.1.1.5.3.2), only by the system. System-only.
cn	RDN for the schema object.
lDAPDisplayName	"Unique" name that identifies this attribute, used by LDAP clients. If not specified on Add, the DC generates a value as specified in section 3.1.1.2.3.4. The syntax of lDAPDisplayName is described in [RFC2251] section 4.1.4.
attributeSecurityGUID	Optional. GUID by which the security system identifies the **property set** of this attribute. See the specification of property sets in section 3.1.1.2.3.3.
extendedCharsAllowed	Optional. If true, character set constraint is not enforced on values of this attribute. Applies to attributes of syntax String(IA5), String(Numeric), String(Teletex), String(Printable).
rangeLower	Optional. Lower range of values that are allowed for this attribute. For syntax Integer, LargeInteger, Enumeration, String(UTC-time), and String(Generalized-time), rangeLower equals the minimum allowed value. For syntax Object(DN-binary), Object(DN-String), rangeLower equals the minimum length of the binary_value or string_value portion of the given value. For String(Unicode), rangeLower is the minimum length in characters. rangeLower does not affect the allowed values for syntax Boolean and Object(DS-DN). For all other syntaxes, rangeLower equals the minimum length in bytes. Note that rangeLower is a 32-bit integer and cannot express the full range of LargeInteger, String(UTC-time), and String(Generalized-time).
rangeUpper	Optional. Upper range of values that are allowed for this attribute. For syntax Integer, LargeInteger, Enumeration, String(UTC-time), and String(Generalized-time), rangeUpper equals the maximum allowed value. For syntax Object(DN-binary), Object(DN-String), rangeUpper equals the maximum length of the binary_value or string_value portion of the given value. For String(Unicode), rangeUpper is the maximum length in character. rangeUpper does not affect the allowed values for syntax Boolean and Object(DS-DN). For all other syntaxes, rangeUpper equals the maximum length in bytes. Note that rangeUpper is a 32-bit integer and cannot express the full range of LargeInteger, String(UTC-time), and String(Generalized-time).
searchFlags	Optional. The searchFlags attribute specifies whether an attribute is indexed, among other things; see section 2.2.9 for values. It contains bitwise flags as follows: fATTINDEX: * fPDNTATTINDEX: *

Attribute	Description
	fANR: Add this attribute to the ambiguous name resolution (ANR) set. If this flag is set, then fATTINDEX must also be set. See 3.1.1.3.1.3.4 for ANR search. fPRESERVEONDELETE: Specifies that the attribute values MUST be preserved on objects after deletion of the object (that is, when the object is transformed to a tombstone or recycled-object). This flag is ignored for the attributes objectCategory and sAMAccountType, plus all linked attributes. fCOPY: Specifies a hint to LDAP clients that the attribute is intended to be copied when copying the object. This flag is not interpreted by the server. fTUPLEINDEX: * fSUBTREEATTINDEX: * fCONFIDENTIAL: This attribute is confidential, special access check is needed; see section Reads:Access Checks in section 3.1.1.4. fNEVERVALUEAUDIT: Auditing of changes to values contained in this attribute MUST NOT be performed. Auditing is outside the state model. fRODCFilteredAttribute: This attribute is part of the filtered attribute set. This flag is only effective on a DC whose DC functionality level is DS_BEHAVIOR_WIN2008 or greater. See section 3.1.1.2.3.5 for additional restrictions. fEXTENDEDLINKTRACKING: The effects of this search flag are outside the state model. Indicates that a DC should do additional internal tracking for link changes. This flag may be ignored by other implementations but must not be used in a conflicting way that would affect the performance of Windows DCs. fBASEONLY: This attribute is returned only on searches scoped to one object. fPARTITIONSECRET: This attribute requires extended access checks to add, read, and update. The effects of searchFlags marked * are outside the state model. They direct the server to construct certain indexes that affect system performance. These flags may be ignored by other implementations but must not be used in a conflicting way that would affect the performance of Windows DCs.
schemaFlagsEx	Optional. The schemaFlagsEx attribute specifies whether an attribute can be part of the filtered attribute set; see section 2.2.11 for values. It contains bitwise flags as follows: FLAG_ATTR_IS_CRITICAL: If this flag is set and the fRODCFilteredAttribute flag in searchFlags is also set, the fRODCFilteredAttribute flag is ignored. If fRODCFilteredAttribute is not set, then setting this flag has no effect. This flag is effective only on a DC whose DC functionality level is DS_BEHAVIOR_WIN2008 or greater; it is ignored by a DC that is not at that level or greater.
isMemberOfPartialAttributeSet	Optional. If true, the attribute is a member of the forest's partial attribute set. An attribute is a member of the forest's partial attribute set if and only if either (1) this attribute is true or (2) the FLAG_ATTR_REQ_PARTIAL_SET_MEMBER bit is set in the systemFlags attribute. If this attribute is true and the FLAG_ATTR_NOT_REPLICATED bit is set in the systemFlags attribute, and if the attribute is modified on a DC that is

Attribute	Description
	also a GC server, then the value of the attribute is accessible through that GC server, but the value of the attribute does not replicate. If the FLAG_ATTR_NOT_REPLICATED bit is set in the systemFlags attribute, the attribute value does not replicate to other GC servers.

3.1.1.2.3.1 Auto-Generated linkID

If the DC functional level is DS_BEHAVIOR_WIN2003 or greater, and an attributeSchema object is created with LDAP Add, and the Add request assigns the OID 1.2.840.113556.1.2.50 as the value of the linkID attribute, the DC sets the linkID attribute to an even integer that does not already appear as the linkID on a schema object. The attribute created by the Add is a forward link attribute.

If the DC functional level is DS_BEHAVIOR_WIN2003 or greater, and an attributeSchema object is created with LDAP Add, and the Add request assigns either the attributeID or the lDAPDisplayName of an existing forward link attribute as the value of the linkID attribute, the DC sets the linkID attribute to the linkID of the given forward link attribute plus one. The attribute created by the Add is a back link attribute corresponding to the given forward link attribute.

The aforementioned values that trigger auto-generation behavior for the linkID are of syntax String(Object-Identifier) or String(Unicode), and therefore do not conform to the declared syntax of the linkID attribute. The DC accepts these values without the error that would normally occur in such a case.

3.1.1.2.3.2 Auto-Generated mAPIID

If the DC functional level is DS_BEHAVIOR_WIN2008 or greater, and an attributeSchema object is created with LDAP Add, and the Add request assigns the OID 1.2.840.113556.1.2.49 as the value of the mAPIID attribute, the DC sets the mAPIID attribute to an integer that does not already appear as the mAPIID on a schema object. An implementation can use any algorithm to choose the next integer as long as that algorithm satisfies this uniqueness constraint. This mAPIID uniqueness spans all the mAPIID attributes on schema objects that are currently persisted in the directory.

The aforementioned value that triggers auto-generation behavior for mAPIID is of syntax String (Object-Identifier), which does not conform to the declared syntax of the mAPIID attribute. The DC accepts these values without the error that would normally occur in such a case.

3.1.1.2.3.3 Property Set

A property set consists of a set of related attributes. An attribute whose attributeSchema object has a value for the attributeSecurityGUID attribute belongs to that property set; the property set is identified by the property set GUID, which is the attributeSecurityGUID value.

A property set GUID can be used instead of the schemaIDGUID of an attribute when defining a security descriptor, as specified in section 5.1.3.2, to grant or deny access to all attributes in one **ACE**.

The following table lists the property sets present in the default AD DS schema.

Name	Property set GUID
Domain Password & Lockout Policies	C7407360-20BF-11D0-A768-00AA006E0529

Name	Property set GUID
General Information	59BA2F42-79A2-11D0-9020-00C04FC2D3CF
Account Restrictions	4C164200-20C0-11D0-A768-00AA006E0529
Logon Information	5F202010-79A5-11D0-9020-00C04FC2D4CF
Group Membership	BC0AC240-79A9-11D0-9020-00C04FC2D4CF
Phone and Mail Options	E45795B2-9455-11D1-AEBD-0000F80367C1
Personal Information	77B5B886-944A-11D1-AEBD-0000F80367C1
Web Information	E45795B3-9455-11D1-AEBD-0000F80367C1
Public Information	E48D0154-BCF8-11D1-8702-00C04FB96050
Remote Access Information	037088F8-0AE1-11D2-B422-00A0C968F939
Other Domain Parameters (for use by SAM)	B8119FD0-04F6-4762-AB7A-4986C76B3F9A
DNS Host Name Attributes	72E39547-7B18-11D1-ADEF-00C04FD8D5CD
MS-TS-GatewayAccess (*)	FFA6F046-CA4B-4FEB-B40D-04DFEE722543
Private Information (*)	91E647DE-D96F-4B70-9557-D63FF4F3CCD8
Terminal Server License Server (*)	5805BC62-BDC9-4428-A5E2-856A0F4C185E

(*) The last three property sets are present only in Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system AD DS forests.

To determine the set of attributes that belong to a property set, search for the corresponding property-set GUID in [MS-ADA1], [MS-ADA2], and [MS-ADA3] for AD DS, or in [MS-ADLS] for AD LDS. All attributeSchema classes that have their attributeSecurityGUID set as the property-set GUID belong to that property set.

New property sets can be created by adding controlAccessRight objects to the **Extended-Rights container** as described in section 5.1.3.2.1. The rightsGuid attribute of the controlAccessRight object is the property set GUID.

AD LDS installs a reduced schema by default. The default AD LDS schema only includes the following property sets:

- General Information
- Account Restrictions
- Logon Information
- Group Membership
- Phone and Mail Options
- Personal Information

- Web Information
- Public Information

3.1.1.2.3.4 ldapDisplayName Generation

When lDAPDisplayName is not given explicitly when creating an attribute or class, the system will generate a default one from the value of cn with the following routine:

```
String generateLdapDisplayName(IN cn: String)
{
  Identify the substrings in cn that are delimited by
    one or more characters in the set {' ', '-', '_'},
    let S be a string array containing all the substrings;
  Let T be a string array with the same number of elements
    as S, such that
    1. First string in T (T[1]) is exactly the same string
       as S[1], except the first character  of T[1] is the
       lower case form of the first character of S[1];
    2. For the remaining strings, T[i] is the same as S[i],
       except the first character of T[i] is the upper case
       of the first character of S[i];
  Let string st be the concatenation of the strings in T;
  Return st;
}
```

For example, if the cn of a new class is Sam-Domain, the default lDAPDisplayName is samDomain.

3.1.1.2.3.5 Flag fRODCFilteredAttribute in Attribute searchFlags

An attribute cannot be a member of a filtered attribute set if one of the following conditions is true:

- The FLAG_ATTR_NOT_REPLICATED bit is set in attribute systemFlags of the attributeSchema object;
- The FLAG_ATTR_REQ_PARTIAL_SET_MEMBER bit is set in attribute systemFlags of the attributeSchema object;
- The FLAG_ATTR_IS_CONSTRUCTED bit is set in attribute systemFlags of the attributeSchema object;
- The FLAG_ATTR_IS_CRITICAL bit is set in attribute schemaFlagsEx of the attributeSchema object;
- Attribute systemOnly of the attributeSchema object is true;
- The attribute is in the following list: currentValue, dBCSPwd, unicodePwd, ntPwdHistory, priorValue, supplementalCredentials, trustAuthIncoming, trustAuthOutgoing, lmPwdHistory, initialAuthIncoming, initialAuthOutgoing, msDS-ExecuteScriptPassword, displayName, codePage, creationTime, lockoutDuration, lockOutObservationWindow, logonHours, lockoutThreshold, maxPwdAge, minPwdAge, minPwdLength, nETBIOSName, pwdProperties, pwdHistoryLength, pwdLastSet, securityIdentifier, trustDirection, trustPartner, trustPosixOffset, trustType, rid, domainReplica, accountExpires, nTMixedDomain, operatingSystem, operatingSystemVersion, operatingSystemServicePack, fSMORoleOwner, trustAttributes, trustParent, flatName, sIDHistory, dNSHostName, lockoutTime, servicePrincipalName, isCriticalSystemObject, msDS-TrustForestTrustInfo, msDS-SPNSuffixes, msDS-AdditionalDnsHostName, msDS-

AdditionalSamAccountName, msDS-AllowedToDelegateTo, msDS-KrbTgtLink, msDS-AuthenticatedAtDC, msDS-SupportedEncryptionTypes.

If one of the conditions is true, the attribute will not be in the filtered attribute set even if the flag fRODCFilteredAttribute is set in attribute searchFlags of the attributeSchema object.

3.1.1.2.4 Classes

3.1.1.2.4.1 Class Categories

There are four categories of classes:

Structural classes: Structural classes are the classes that can have instances in the directory.

Abstract classes: **Abstract classes** are templates that are used to derive new classes. Abstract classes cannot be instantiated in the directory.

Auxiliary classes: **Auxiliary classes** contain a list of attributes. Adding the auxiliary class to the definition of a structural or abstract class adds the auxiliary class's attributes to the definition. An auxiliary class cannot be instantiated by itself in the directory.

88 classes: 88 classes do not fall into any of the preceding categories. An 88 class can be used as an abstract class, a structural class, or an auxiliary class.

Structural class, abstract class, and auxiliary class are defined in [X501] section 8.3. 88 class corresponds to the definition of object classes described in [X501] section 8.3.4. 88 class is included for compatibility with this older standard and is not intended to be used in new schema extensions.

3.1.1.2.4.2 Inheritance

Inheritance is the ability to build new classes from existing classes. The new class is defined as a subclass of another class, called its *superclass*. A subclass inherits from its superclass the mandatory and optional attributes and its structural parent classes in the directory hierarchy. All classes are subclasses, directly or indirectly, of a single **abstract object class**, called top. In Active Directory, a class has exactly one superclass; top is its own superclass. An ordered set of superclasses of a class, ending with class top, is its *superclass chain* ([X501]). The superclass chain of a class does not include the class itself, except that the superclass chain of top is the single-element sequence [top].

Abstract classes can inherit only from abstract classes, auxiliary classes can inherit from all classes except structural classes, and structural classes can inherit from all classes except auxiliary classes. Classes of the category **88 class** (section 3.1.1.2.4.1) can inherit from all classes.

3.1.1.2.4.3 objectClass

Attribute objectClass is a multivalued attribute that appears on all the objects in the directory. When instantiating a structural class or an 88 object class, the objectClass attribute of the new object contains a sequence of class names. The first element is always class top. The last element is the name of the structural class or the 88 object class that was instantiated (referred to as the most specific class). The rest of the classes in the superclass chain are listed in between in the order of inheritance from class top. For example, a user object has the following four-element sequence as the value of objectClass:

[top, person, organizationalPerson, user]

For information on instantiating auxiliary classes see section 3.1.1.2.4.6.

3.1.1.2.4.4 Structure Rules

Structure rules define the possible tree structures. In Active Directory, the structure rules (for directory hierarchy, see section 3.1.1.2.4.2) are completely expressed by the possSuperiors and systemPossSuperiors attributes that are present on each classSchema object. The union of values in these two attributes specifies the list of classes, instances of which are allowed to be parents of an object instance of the class in question.

3.1.1.2.4.5 Content Rules

Content rules determine the mandatory and optional attributes of the class instances that are stored in the directory. In Active Directory, the content rules are completely expressed by the mustContain, mayContain, systemMustContain, and systemMayContain attributes of the schema definitions for each class. The union of values in the mustContain and systemMustContain attributes specifies the attributes that are required to be present on an object instance of the class in question. The union of values in the mustContain, systemMustContain, mayContain, and systemMayContain attributes specifies the attributes that are allowed to be present on an object instance of the class in question.

3.1.1.2.4.6 Auxiliary Class

Active Directory provides support for statically linking auxiliary classes to the classSchema definition of another object class. When an auxiliary class *aux* is statically linked to some other class *cl*, it is as if all of the mandatory and optional attributes of the auxiliary class *aux* are added to the class *cl*.

The governsID of auxiliary class *aux* is contained in the auxiliaryClass attribute of *cl* if *aux* was statically linked to *cl* by modifying the auxiliaryClass attribute of *cl*'s classSchema definition as specified in section 3.1.1.3.1.1.5. The governsID of auxiliary class *aux* is contained in the systemAuxiliaryClass attribute of *cl* if *aux* was statically linked to *cl* by modifying the systemAuxiliaryClass attribute of *cl*'s classSchema definition as specified in section 3.1.1.3.1.1.5.

A statically linked auxiliary class with mandatory attributes must be linked to the class *cl* through the systemAuxiliaryClass attribute of *cl* at the time *cl* is defined as described in section 3.1.1.3.1.1.5. The objectClass attribute of objects of class *cl* does not include the names of statically linked auxiliary classes or the classes in their superclass chains.

Active Directory also provides support for dynamically linking auxiliary classes on objects, which reflects the model of auxiliary object classes described in [X501] section 8.3.3. In this case, the dynamically linked auxiliary class affects only the individual object to which it is linked, as opposed to a statically linked auxiliary class, which is linked to a class and affects every object of that class. The classSchema of the class is not affected by dynamic auxiliary classes. When an auxiliary class is dynamically linked to an object, the mandatory and optional attributes of the auxiliary class become mandatory and optional attributes of the object. Refer to section 3.1.1.3.1.1.5 for auxiliary class related LDAP operations supported by Active Directory.

If an object is dynamically linked to one or more auxiliary classes, attribute objectClass of the object contains the following values in the order described below.

1. Class top remains as the first value;

2. Then it is followed by the set of dynamic auxiliary classes and the classes in their superclass chains, excluding those already present in the superclass chain of the most specific structural class. There is no specific order among the classes in this set, and no class is listed more than once.

3. Next, the classes in the superclass chain of the most specific structural class are listed after that, in the order of inheritance from top.

4. The most specific structural class remains last in the sequence.

The auxiliaryClass or systemAuxiliaryClass attributes are not affected by dynamic auxiliary classes.

For example, a user object with auxiliary class mailRecipient dynamically added has the following five-element sequence as the value of objectClass:

[top, mailRecipient, person, organizationalPerson, user]

Dynamic auxiliary classes are not supported when the forest functional level is DS_BEHAVIOR_WIN2000.

3.1.1.2.4.7 RDN Attribute of a Class

Each class designates an RDN attribute. The RDN attribute's name and value provide the RDN for the class, for example "ou=ntdev", "cn=Peter Houston". If not specified in a class by attribute rDNAttID, the RDN attribute is inherited from the superclass of the class. The RDN attribute is of syntax String(Unicode).

3.1.1.2.4.8 Class classSchema

The attributes of class classSchema are specified in the following table.

The term "Unique" (in quotation marks) in the table is defined in section 3.1.1.2.3.

The term system-only in the table is defined in section 3.1.1.2.3.

Attribute	Description
objectClass	Equals the sequence [top, classSchema]. System-only.
governsID	"Unique" OID that identifies this class. System-only.
schemaIDGUID	"Unique" GUID that identifies this class, used in security descriptors. If not specified on Add, the DC generates a fresh GUID. System-only.
msDS-IntId	Optional. 32-bit unsigned integer. System-only.
rDNAttID	Optional. attributeID of the RDN attribute. If the rDNAttID is not present, the RDN attribute is inherited from the superclass of this class. System-only.
subClassOf	governsID of the superclass of this class. System-only. Also see section 3.1.1.2.5.2 for auto-generated behavior when a new classSchema object is created.
systemMustContain	Optional. attributeIDs of the mandatory attributes of this class. This attribute is system-only.
systemMayContain	Optional. attributeIDs of the optional attributes of this class. This attribute is system-only.
systemPossSuperiors	Optional. governsIDs of the classes that can be parents of this class within an NC tree. This attribute is system-only.
systemAuxiliaryClass	Optional. governsIDs of the auxiliary classes that are statically linked to this class. This attribute is system-only.
objectClassCategory	Class category (section 3.1.1.2.4.1), encoded as follows:

Attribute	Description
	0: 88 Class 1: Structural class 2: Abstract class 3: Auxiliary class System-only.
systemFlags	Optional. Flags that determine specific system operations; see section 2.2.10 for values. The single systemFlags value specific to a classSchema object is: FLAG_SCHEMA_BASE_OBJECT: this class is part of the base schema. Modifications to a base schema object are restricted as described in section 3.1.1.2.5. System-only.
systemOnly	Optional. Only a DC can create (section 3.1.1.5.2.2) and modify (section 3.1.1.5.3.2) instances of a system-only class. System-only.
cn	RDN for the schema object.
lDAPDisplayName	"Unique" name that identifies this class, used by LDAP clients. If not specified on Add, the DC generates a value as specified in section 3.1.1.2.3.4. The syntax of lDAPDisplayName is described in [RFC2251] section 4.1.4.
mustContain	Optional. attributeIDs of the mandatory attributes of this class in addition to the systemMustContain attributes.
mayContain	Optional. attributeIDs of the optional attributes of this class in addition to the systemMayContain attributes.
possSuperiors	Optional. governsIDs of the classes that can be parents of this class within an NC tree, in addition to the systemPossSuperiors classes.
auxiliaryClass	Optional. governsIDs of the auxiliary classes that are statically linked to this class, in addition to the systemAuxiliaryClass classes.
defaultSecurityDescriptor	Optional. The default security descriptor (in SDDL format, [MS-DTYP] section 2.5.1) that is assigned to new instances of this class if no security descriptor is specified during creation of the class or is merged into a security descriptor if one is specified. The rules for security descriptor merging are specified in [MS-DTYP] section 2.5.3.4.
defaultObjectCategory	A reference to some classSchema object. This value is the default value of the objectCategory attribute of new instances of this class if none is specified during LDAP Add. Also see section 3.1.1.2.5.2 for auto-generated behavior when a new classSchema object is created.
defaultHidingValue	Optional. If defaultHidingValue is true on a classSchema object, then when an Add creates an instance of this class (that is, where this class is the most specific class) and the Add does not specify a value for the showInAdvancedViewOnly attribute, it is as if the Add had specified true for the showInAdvancedViewOnly attribute. The showInAdvancedViewOnly attribute is interpreted by LDAP clients, not by the DC. If true, certain user interfaces do not display the object.
showInAdvancedViewOnly	Specifies whether the attribute is to be visible in the advanced mode of user interfaces. Also see defaultHidingValue defined previously and section 3.1.1.2.5.2 for

Attribute	Description
	auto-generated behavior when a new classSchema object is created.

3.1.1.2.5 Schema Modifications

This section documents the special behavior of schema objects with respect to LDAP Add, Modify, Modify DN, and Delete requests.

Only the DC that owns the Schema Master FSMO role performs originating updates of objects in the schema NC, as specified in section 3.1.1.1.11.

All transactions that perform originating updates to objects in the schema NC are serialized, even if the updates do not appear to conflict and thus do not seem to require serialization.

Many attributes of attributeSchema and classSchema objects are system-only, as specified in sections 3.1.1.2.3 and 3.1.1.2.4. An LDAP Modify request that attempts to modify a system-only attribute (except as specified in section 3.1.1.5.3.2) fails with error *constraintViolation* / *ERROR_DS_CANT_MOD_SYSTEM_ONLY*.

A Delete of an attributeSchema or classSchema object fails, with error *unwillingToPerform* / *ERROR_DS_CANT_DELETE*.

There is no constraint on the amount of time between when an object in the schema NC is successfully added or modified and when the DC enforces the updated schema. Therefore, it is possible that there is a period of time during which the schema enforced by the DC does not reflect the schema represented by the objects in the schema NC. Although the protocol places no boundary or requirements on the length of this time period, it is recommended that implementations minimize the length of this time period to improve the usability of the directory for clients.

The server MUST guarantee that all successful schema modifications are eventually enforced.

3.1.1.2.5.1 Consistency and Safety Checks

This section documents schema object special behaviors that are not closely tied to the defunct state. These special behaviors are divided into two classes:

- Consistency checks
- Safety checks

Consistency checks maintain the consistency of the schema. Safety checks reduce the possibility of a schema update by one application breaking another application.

If an Add or Modify request fails either a consistency or a safety check, the response is error *unwillingToPerform* / *<unrestricted>*.

3.1.1.2.5.1.1 Consistency Checks

The term "Unique" (in quotation marks) in the following statements is defined in section 3.1.1.2.3.

An Add or Modify request on an attributeSchema object succeeds only if the resulting object passes all of the following tests:

- The value of lDAPDisplayName is syntactically valid, per [RFC2251] section 4.1.4.

- The values of attributeID, lDAPDisplayName, mAPIID (if present) and schemaIDGUID are "Unique".
- A nonzero linkID, if any, is unique among all values of the linkID attribute on objects in the schema NC, regardless of forest functional level. If a linkID is an odd number, it is not one, and an object exists whose linkID is the even number one smaller.
- The values of attributeSyntax, oMSyntax, and oMObjectClass match some defined syntax (section 3.1.1.2.2).
- Flag fANR is only present in the searchFlags attribute if the syntax is String(Unicode), String(IA5), String(Printable), String(Teletex) or String(Case).
- If rangeLower and rangeUpper are present, rangeLower is smaller than or equal to rangeUpper.

An Add or Modify request on a classSchema object succeeds only if the resulting object passes all of the following tests.

- The value of lDAPDisplayName is syntactically valid, per [RFC2251] section 4.1.4.
- The values of governsID, lDAPDisplayName, and schemaIDGUID are "Unique".
- All attributes that are referenced in the systemMayContain, mayContain, systemMustContain, and mustContain lists exist and are active.
- All classes that are referenced in the subClassOf, systemAuxiliaryClass, auxiliaryClass, systemPossSuperiors, and possSuperiors lists exist and are active.
- All classes in the systemAuxiliaryClass and auxiliaryClass attributes have either 88 class or auxiliary class specified as their objectClassCategory.
- All classes in the systemPossSuperiors and possSuperiors attributes have either 88 class or structural class specified as their objectClassCategory.
- The superclass chain of a class follows the rules for inheritance as specified in section 3.1.1.2.4.2.
- The dynamicObject class is not referenced by the subClassOf attribute of a class.
- The attribute specified in the rDNAttID attribute has syntax String(Unicode).
- Attribute defaultSecurityDescriptor, if present, is a valid SDDL string.

3.1.1.2.5.1.2 Safety Checks

The following checks reduce the possibility of schema updates by one application breaking another application.

These checks apply to all schema objects:

- A Modify adds no attributes to the mustContain or systemMustContain of an existing class.
- A Modify does not add an auxiliary class to the auxiliaryClass or systemAuxiliaryClass of an existing class, if doing so would effectively add either mustContain or systemMustContain attributes to the class.
- A Modify does not change the objectClassCategory of an existing class.

- A Modify does not change a constructed attribute (an attribute with FLAG_ATTR_IS_CONSTRUCTED in systemFlags).
- A Modify does not change class top, except to add back link attributes as may-contains, either by adding back link attributes to mayContain of top, or by adding auxiliary classes to auxiliaryClass of top whose only effect on top is adding back link attributes as may-contains.
- A Modify does not change the subSchema object.
- A Modify does not change the fRODCFilteredAttribute bit of the searchFlags attribute of an attributeSchema object, if the DC functional level is DS_BEHAVIOR_WIN2008 or higher, and the attributeSchema object cannot be a member of the filtered attribute set (see section 3.1.1.2.3.5).

These checks apply to schema objects that include FLAG_SCHEMA_BASE_OBJECT in the systemFlags attribute:

- A Modify does not change the lDAPDisplayName or cn of an attributeSchema or classSchema object, or the defaultObjectCategory of a classSchema object.
- A Modify does not change the classSchema objects attributeSchema, classSchema, subSchema and dMD.
- A Modify does not change the fCONFIDENTIAL bit of the searchFlags attribute of an attributeSchema object.
- A Modify does not change the attributeSecurityGUID on the following fixed list of attributeSchema objects: accountExpires, badPwdCount, codePage, countryCode, description, displayName, domainReplica, forceLogoff, homeDirectory, homeDrive, memberOf, lastLogoff, lastLogon, lockOutObservationWindow, lockoutDuration, lockoutThreshold, logonCount, logonHours, logonWorkstation, maxPwdAge, member, minPwdAge, minPwdLength, modifiedCount, objectSid, oEMInformation, profilePath, primaryGroupID, pwdHistoryLength, pwdProperties, sAMAccountName, scriptPath, serverState, serverRole, uASCompat, comment, pwdLastSet, userAccountControl, userParameters.

3.1.1.2.5.2 Auto-Generated Attributes

If a classSchema object is created with an LDAP Add operation and the following attributes are not included as part of the Add, they must be created on the object as specified in the following table.

Attribute	Default auto-generated value
subClassOf	Must refer to class top
showInAdvancedViewOnly	TRUE
defaultObjectCategory	Must refer to the new classSchema object itself

3.1.1.2.5.3 Defunct

A schema object with isDefunct = true is defunct; a schema object that is not defunct is active. This section documents the special behavior of attributeSchema and classSchema objects related to the defunct state.

The effect of being defunct depends upon the forest functional level as specified in the following subsections. The following statements are independent of the forest functional level.

- The isDefunct attribute being not present on an attributeSchema or classSchema object is equivalent to isDefunct = false; modifications that move between these two representations of the active state have no special behavior.
- If an LDAP Modify changes the isDefunct attribute (giving it a value of true or false, or removing it), this change must be the only change in the LDAP Modify request; otherwise, the request fails with error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- If a Modify sets isDefunct to true but the attributeSchema or classSchema object is base (that is, it has FLAG_SCHEMA_BASE_OBJECT present in its systemFlags attribute), the Modify fails, with error *unwillingToPerform* / *ERROR_DS_ILLEGAL_BASE_SCHEMA_MOD*.
- LDAP Add cannot create instances of a defunct class (section 3.1.1.5.2.2), and LDAP Add and Modify cannot create instances of a defunct attribute (see sections 3.1.1.5.2.2 and 3.1.1.5.3.2).
- Making an attributeSchema or classSchema object defunct has no effect on the state of existing objects that use the defunct attribute or class, but it changes the behavior of reads and updates of such objects as described in sections 3.1.1.4.8 (Search), 3.1.1.5.2.2 (Add), 3.1.1.5.3.2 (Modify), and 3.1.1.5.5 (Delete).

3.1.1.2.5.3.1 Forest Functional Level Less Than WIN2003

If the forest functional level is less than DS_BEHAVIOR_WIN2003, a DC behaves as follows with respect to the defunct state:

- The isDefunct attribute can be changed from not present (or false) to true on an attributeSchema or classSchema object. This modification is subject to the following checks:
  - If the modification is to an attributeSchema object and the object is a mustContain, systemMustContain, mayContain, or systemMayContain of an active class, the modification fails.
  - If the modification is to a classSchema object and the object is a subClassOf, auxiliaryClass, or possSuperiors of an active class, the modification fails.

  The error if the isDefunct modification fails is *unwillingToPerform* / *<unrestricted>*.
- When isDefunct is true on an attributeSchema or classSchema object, an LDAP Modify can set isDefunct to false (or remove the isDefunct attribute). This modification is subject to the following check:
  - If the modification is to a classSchema object and the object references any defunct attributes through its mustContain, systemMustContain, mayContain, or systemMayContain attributes, or references any defunct classes through its subClassOf, auxiliaryClass, or possSuperiors attributes, the modification fails.

  The error if the isDefunct modification fails is *unwillingToPerform* / *<unrestricted>*.
- No other modification to a defunct attributeSchema or classSchema object is allowed. The error if the modification fails is *noSuchObject* / *<unrestricted>*.

3.1.1.2.5.3.2 Forest Functional Level WIN2003 or Greater

If the forest functional level is DS_BEHAVIOR_WIN2003 or greater, a DC behaves as follows with respect to the defunct state:

- An LDAP Modify can change the isDefunct attribute from not present (or false) to true on an attributeSchema or classSchema object. This modification is subject to the following checks, in addition to the checks performed when the forest functional level is less than DS_BEHAVIOR_WIN2003:

  - If the modification is to an attributeSchema object and the object is a mustContain, systemMustContain, mayContain, systemMayContain, or rDNAttID of an active class, the modification fails.

  - If the modification is to a classSchema object and the object is a subClassOf, auxiliaryClass, or possSuperiors of an active class, the modification fails.

  The error if the isDefunct modification fails is *unwillingToPerform* / *<unrestricted>*.

- An LDAP Modify can change the isDefunct attribute from true to false (or not present) on an attributeSchema or classSchema object. This modification is subject to the following checks, in addition to the checks performed when the forest functional level is less than DS_BEHAVIOR_WIN2003:

  - If the modification is to a classSchema object and the object references any defunct attributes through its mustContain, systemMustContain, mayContain, systemMayContain or rDNAttID attributes, or references any defunct classes through its subClassOf, auxiliaryClass, or possSuperiors attributes, the modification fails.

  - The same uniqueness checks are performed when setting isDefunct to false as would have been performed if the same object were being added to a schema where it was not present. In particular, the uniqueness checks on attributeID, governsID, schemaIDGUID, mAPIID, linkID, and lDAPDisplayName must pass.

  The error if the isDefunct modification fails is *unwillingToPerform* / *<unrestricted>*.

- An LDAP Modify can change the other attributes of defunct schema objects subject to the same checks that apply to changes to active schema objects.

  Therefore, for instance, a Modify can change the lDAPDisplayName of a defunct attributeSchema object, or the lDAPDisplayName, mustContain, mayContain, subClassOf, auxiliaryClass, and possSuperiors of a defunct classSchema object.

  Because the checks that apply to changes to active schema objects are still in force, Modify cannot (for instance) change the attributeID, governsID, schemaIDGUID, mAPIID, linkID, attributeSyntax, oMSyntax, and oMObjectClass attributes of defunct schema objects.

- Section 3.1.1.4.8 specifies the effects of the defunct state on reads of OID-valued attributes that identify schema objects (mustContain, systemMustContain, mayContain, systemMayContain, subClassOf, auxiliaryClass, and possSuperiors).

3.1.1.2.6 ATTRTYP

Any OID-valued quantity stored on an object is stored as an ATTRTYP, a 32-bit unsigned integer. The ATTRTYP space is 32 bits wide and is divided into the following ranges.

Range	Description
[0x00000000..0x7FFFFFFF]	ATTRTYPs that map to OIDs via the **prefix table**.
[0x80000000..0xBFFFFFFF]	ATTRTYPs used as values of msDS-IntId attribute.

Range	Description
[0xC0000000..0xFFFEFFFF]	Reserved for future use.
[0xFFFF0000.. 0xFFFFFFFF]	Reserved for internal use (never appear on the wire).

The mapping from ATTRTYPs A to OID O works as follows:

- If A in [0x00000000..0x7FFFFFFF], A maps to O via a prefix table as specified in [MS-DRSR] section 5.16.4 (the OidFromAttid procedure).
- If A in [0x80000000..0xBFFFFFFF], let X be the object such that X!msDS-IntId equals A. If X is an attributeSchema object, O is X!attributeID; otherwise X is an classSchema object, and O is X!governsID.

Given an OID O, the schema object X representing the class or attribute identified by O is the object X such that either X!attributeID equals O or X!governsID equals O.

3.1.1.3 LDAP

Active Directory is a server for LDAP. This section specifies the extensions and variations of LDAP that are supported by Active Directory. Except as otherwise noted, all material applies to both AD DS and AD LDS. Also, except as noted, all information applies to all versions of AD DS and AD LDS.

This section is structured as follows:

- Section 3.1.1.3.1 documents the interpretation of the LDAP RFCs made by Active Directory and deviations from the LDAP RFCs.
- The rootDSE (empty DN) is a mechanism for clients of an LDAP server to interact with the server itself, rather than with particular objects contained by the server. Section 3.1.1.3.2 specifies the rootDSE reads supported by Active Directory, and section 3.1.1.3.3 specifies the rootDSE updates.
- LDAP has several extension mechanisms in addition to the rootDSE. Section 3.1.1.3.4 specifies the LDAP extensions that Active Directory supports.

3.1.1.3.1 LDAP Conformance

The purpose of this section is to document how the implementation of Active Directory DCs interprets the LDAP v3 RFCs, including differences from those RFCs. Except as noted in the following subsections, Active Directory is compliant to [RFC3377].

Active Directory DCs nominally implement support for LDAP v2 [RFC1777]. However, except as noted in the next paragraph, Active Directory processes LDAP v2 requests and generates responses as if LDAP v3 had been requested by the client.

When processing an LDAP v2 request, Active Directory exhibits the following behavioral differences from processing an LDAP v3 request:

- Instead of using the UTF-8 character encoding for LDAPString [RFC2251], the system's configured **code page** is used. The code page is configured locally on the DC by the DC's administrator.
- Referrals and continuation references are generated using the format for LDAP v2 referrals as specified in section 3.1.1.3.4.

All LDAP error codes returned by Active Directory are taken from the resultCode enumeration of the LDAPResult structure defined in [RFC2251] section 4.1.10.

3.1.1.3.1.1 Schema

This section discusses the implementation of the schema in Active Directory DCs, as it relates to the IETF RFC standards for LDAP schemas.

3.1.1.3.1.1.1 subSchema

Per [RFC2251] and [RFC2252], Active Directory exposes a subSchema object that is pointed to by the subschemaSubentry attribute on the rootDSE. In accord with [RFC2251] section 3.2.2, this subSchema object contains the required cn, objectClass, objectClasses, and attributeTypes attributes. Additionally, it contains the dITContentRules attribute. It does not contain the matchingRules, matchingRuleUse, dITStructureRules, nameForms, or ldapSyntaxes attributes. It contains the modifyTimeStamp attribute but not the createTimeStamp attribute. The subSchema object does not support the createTimeStamp attribute even though its object class derives from top, which contains the createTimeStamp attribute as part of systemMayContain. In contrast to [RFC2252] section 7.2, in Active Directory the subSchema class is defined to be structural rather than auxiliary.

The meaning of the attributeTypes, objectClasses, and dITContentRules attributes are as described in those RFCs. However, the values stored in these attributes use only a subset of the AttributeTypeDescription, ObjectClassDescription, and DITContentRuleDescription grammars described in [RFC2252]. The following grammars are used by Active Directory. Other than the removal of certain elements, these grammars are identical to those of [RFC2252].

```
AttributeTypeDescription = "(" whsp
    numericoid whsp                 ; attributeID
    [ "NAME" qdescrs ]              ; lDAPDisplayName
    [ "SYNTAX" whsp noidlen whsp ]  ; see RFC 2252 section 4.3
    [ "SINGLE-VALUE" whsp ]         ; default multi-valued
    [ "NO-USER-MODIFICATION" whsp ] ; default user modifiable
    whsp ")"

ObjectClassDescription = "(" whsp
    numericoid whsp         ; governsID
    [ "NAME" qdescrs ]      ; lDAPDisplayName
    [ "SUP" oids ]          ; governsIDs of superior object classes
    [ ( "ABSTRACT" / "STRUCTURAL" / "AUXILIARY" ) whsp ]
                            ; default structural
    [ "MUST" oids ]         ; attributeIDs of required attributes
    [ "MAY" oids ]          ; attributeIDs of optional attributes
    whsp ")"

DITContentRuleDescription = "("
    numericoid            ; governsID of structural object class
    [ "NAME" qdescrs ]    ; lDAPDisplayName
    [ "AUX" oids ]        ; governsIDs of auxiliary classes
    [ "MUST" oids ]       ; attributeIDs of required attributes
    [ "MAY" oids ]        ; attributeIDs of optional attributes
    ")"
```

Active Directory supports additional SYNTAX values not defined in [RFC2252]. The following table lists the SYNTAX values returned for each LDAP syntax name. See section 3.1.1.2.2 for more information on syntaxes.

LDAP syntax name	SYNTAX Value
Boolean	1.3.6.1.4.1.1466.115.121.1.7
Enumeration	1.3.6.1.4.1.1466.115.121.1.27
Integer	1.3.6.1.4.1.1466.115.121.1.27
LargeInteger	1.2.840.113556.1.4.906
Object(Access-Point)	1.3.6.1.4.1.1466.115.121.1.2
Object(DN-Binary)	1.2.840.113556.1.4.903
Object(DN-String)	1.2.840.113556.1.4.904
Object(DS-DN)	1.3.6.1.4.1.1466.115.121.1.12
Object(OR-Name)	1.2.840.113556.1.4.1221
Object(Presentation-Address)	1.3.6.1.4.1.1466.115.121.1.43
Object(Replica-Link)	OctetString
String(Case)	1.2.840.113556.1.4.1362
String(Generalized-Time)	1.3.6.1.4.1.1466.115.121.1.24
String(IA5)	1.3.6.1.4.1.1466.115.121.1.26
String(NT-Sec-Desc)	1.2.840.113556.1.4.907
String(Numeric)	1.3.6.1.4.1.1466.115.121.1.36
String(Object-Identifier)	1.3.6.1.4.1.1466.115.121.1.38
String(Octet)	1.3.6.1.4.1.1466.115.121.1.40
String(Printable)	1.3.6.1.4.1.1466.115.121.1.44
String(Sid)	1.3.6.1.4.1.1466.115.121.1.40
String(Teletex)	1.2.840.113556.1.4.905
String(Unicode)	1.3.6.1.4.1.1466.115.121.1.15
String(UTC-Time)	1.3.6.1.4.1.1466.115.121.1.53

In addition to the preceding attributes, Active Directory contains two additional subSchema attributes, named extendedClassInfo and extendedAttributeInfo. These return additional data about the classes and attributes in a format similar to objectClasses and attributeTypes, respectively. The grammar used for extendedClassInfo is as follows.

```
ObjectClassDescriptionExtended = "(" whsp
    numericoid whsp                 ; governsID
    [ "NAME" qdescrs ]              ; lDAPDisplayName
    [ "CLASS-GUID" whsp guid ]      ; schemaIDGUID
    whsp ")"
```

The NAME field is as in the ObjectClassDescription grammar. The CLASS-GUID field contains the value of the class's schemaIDGUID attribute. That value, which is a GUID, is expressed not in the dashed-string GUID format of [RFC4122] section 3 but rather as the hexadecimal representation of the binary format of the GUID. For example, the GUID whose dashed-string representation is "3fdfee4f-47f4-11d1-a9c3-0000f80367c1" would be expressed as "4feedf3ff447d111a9c30000f80367c1" in the CLASS-GUID field.

The grammar for extendedAttributeInfo is as follows.

```
AttributeTypeDescriptionExtended = "(" whsp
    numericoid whsp                       ; attributeID
    [ "NAME" qdescrs ]                    ; lDAPDisplayName
    [ "RANGE-LOWER" whsp numericstring ] ; rangeLower
    [ "RANGE-UPPER" whsp numericstring ] ; rangeUpper
    [ "PROPERTY-GUID" whsp guid ]         ; schemaIDGUID
    [ "PROPERTY-SET-GUID" whsp guid ]     ; attributeSecurityGUID
    [ "INDEXED" whsp ]                    ; fATTINDEX in searchFlags
    [ "SYSTEM-ONLY" whsp ]                ; systemOnly
    whsp ")"
```

The NAME field is as in the AttributeTypeDescription grammar. The RANGE-LOWER and RANGE-UPPER fields are only present if the attribute's attributeSchema contains values for the rangeLower and rangeUpper attributes, respectively. If present, those fields contain the values of those attributes. The PROPERTY-GUID field contains the value of the attribute's schemaIDGUID. If the attribute has a attributeSecurityGUID attribute, the PROPERTY-SET-GUID field contains the value of that attribute; otherwise, it contains the value of the NULL GUID (the GUID consisting entirely of zeros). For both PROPERTY-GUID and PROPERTY-SET-GUID, the GUID is represented in the same form as that CLASS-GUID from the ObjectClassDescriptionExtended grammar. If the fATTINDEX bit of the attribute's searchFlags is set, the INDEXED field is present. If the attribute's systemOnly attribute is true, the SYSTEM-ONLY field is present.

The attributeTypes, objectClasses, dITContentRules, extendedClassInfo, and extendedAttributeInfo attributes on the subSchema object are read-only. They permit applications to discover the schema on the DC, but they are not the mechanism for changing the schema on the DC. DCs change their schema in response to the addition or modification of classSchema and attributeSchema objects in the schema NC. These objects also contain attributes that supply additional information about the schema that is not present in the attributes of the subSchema object, such as the systemFlags attribute, which specifies additional properties of an attribute (for example, whether it is a constructed attribute). The attributeSchema and classSchema objects and their associated attributes are specified in section 3.1.1.2.

If the forest functional level is DS_BEHAVIOR_WIN2003 or greater, the attributeTypes, dITContentRules, extendedAttributeInfo, extendedClassInfo, and objectClasses attributes on the subSchema object do not contain defunct attributes or classes, only active attributes or classes.

3.1.1.3.1.1.2 Syntaxes

The syntaxes used in Active Directory are based on [RFC2252] section 6. Where Active Directory and [RFC2252] have syntaxes in common, the same means of encoding the value into the syntax is used. However, Active Directory has a number of syntaxes that are not defined in [RFC2252], and vice versa. Additionally, even when Active Directory and [RFC2252] have syntaxes in common, in many cases they use different names for the same syntax, and in all cases they use different OIDs to identify the same syntax.

Active Directory does not use the syntaxes defined in [RFC2256] section 6. The list of syntaxes in Active Directory, their encodings, and how they map to the [RFC2252] syntaxes are documented in section 3.1.1.2.2.

3.1.1.3.1.1.3 Attributes

Sections 5.1 through 5.4 of [RFC2252], as well as section 5 of [RFC2256] and section 2 of [RFC2798], define a set of attributes common to LDAP directories. Additionally, portions of the Active Directory schema are derived from [RFC1274] and [RFC2307]. The following tables show, for each of these RFCs, the attributes that are either included in the Active Directory **default schemas** of Windows Server 2003 operating system, Active Directory Application Mode (ADAM), Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, or present as readable attributes of the rootDSE of Windows 2000 operating system, Windows Server 2003, ADAM, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 DCs (both AD DS and AD LDS). Some of these attributes were added to the schema of Windows Server 2003 or Windows Server 2003 R2 operating system but were not present in the Windows 2000 schema; [MS-ADA1], [MS-ADA2], and [MS-ADA3] specify the attributes included in each version of the schema. For more information about rootDSE attributes, which are not part of the schema, see section 3.1.1.3.2.

RFC 1274

Attribute	Included by AD DS?	Included by AD LDS?
objectClass	Yes	Yes
knowledgeInformation	Yes	No
serialNumber	Yes	Yes
streetAddress	Yes	Yes
title	Yes	Yes
description	Yes	Yes
searchGuide	Yes	Yes
businessCategory	Yes	Yes
postalAddress	Yes	Yes
postalCode	Yes	Yes
postOfficeBox	Yes	Yes
physicalDeliveryOfficeName	Yes	Yes
telephoneNumber	Yes	Yes
telexNumber	Yes	Yes
teletexTerminalIdentifier	Yes	Yes
facsimileTelephoneNumber	Yes	Yes
x121Address	Yes	Yes

Attribute	Included by AD DS?	Included by AD LDS?
internationalISDNNumber	Yes	Yes
registeredAddress	Yes	Yes
destinationIndicator	Yes	Yes
preferredDeliveryMethod	Yes	Yes
presentationAddress	Yes	No
supportedApplicationContext	Yes	No
member	Yes	Yes
owner	Yes	Yes
roleOccupant	Yes	No
seeAlso	Yes	Yes
userPassword	Yes*	Yes*
userCertificate	Yes	Yes
cACertificate	Yes	No
authorityRevocationList	Yes	No
certificateRevocationList	Yes	No
crossCertificatePair	Yes	No
textEncodedORAddress	Yes	No
roomNumber	Yes	Yes
photo	Yes	Yes
userClass	Yes	No
host	Yes	No
manager	Yes	Yes
documentIdentifier	Yes	No
documentTitle	Yes	No
documentVersion	Yes	No
documentAuthor	Yes	No
documentLocation	Yes	No
secretary	Yes	Yes
otherMailbox	Yes	No
associatedDomain	Yes	No

Attribute	Included by AD DS?	Included by AD LDS?
associatedName	Yes	No
homePostalAddress	Yes	Yes
personalTitle	Yes	Yes
organizationalStatus	Yes	No
buildingName	Yes	No
audio	Yes	Yes
documentPublisher	Yes	No
aliasedObjectName	No	No
commonName	No	No
surname	No	No
countryName	No	No
localityName	No	No
stateOrProvinceName	No	No
organizationName	No	No
mhsDeliverableContentLength	No	No
mhsDeliverableContentTypes	No	No
mhsDeliverableEits	No	No
mhsDLMembers	No	No
mhsDLSubmitPermissions	No	No
mhsMessageStoreName	No	No
mhsORAddresses	No	No
mhsPreferredDeliveryMethods	No	No
mhsSupportedAutomaticActions	No	No
mhsSupportedContentTypes	No	No
mhsSupportedOptionalAttributes	No	No
userid	No	No
rfc822Mailbox	No	No
info	No	No
favouriteDrink	No	No
homeTelephoneNumber	No	No

Attribute	Included by AD DS?	Included by AD LDS?
lastModifiedTime	No	No
lastModifiedBy	No	No
domainComponent	No	No
aRecord	No	No
mXRecord	No	No
nSRecord	No	No
sOARecord	No	No
cNAMERecord	No	No
mobileTelephoneNumber	No	No
pagerTelephoneNumber	No	No
friendlyCountryName	No	No
uniqueIdentifier	No	No
janetMailbox	No	No
mailPreferenceOption	No	No
dSAQuality	No	No
singleLevelQuality	No	No
subtreeMinimumQuality	No	No
subtreeMaximumQuality	No	No
personalSignature	No	No
dITRedirect	No	No

* Active Directory uses the userPassword attribute to set or change passwords only in limited circumstances. See section 3.1.1.3.1.5.

RFC 2252

Attribute	Included by AD DS?	Included by AD LDS?
createTimeStamp	Yes	Yes
modifyTimeStamp	Yes	Yes
subSchemaSubEntry	Yes	Yes
attributeTypes	Yes	Yes
objectClasses	Yes	Yes

Attribute	Included by AD DS?	Included by AD LDS?
namingContexts	Yes	Yes
supportedExtension	Yes	Yes
supportedControl	Yes	Yes
supportedSASLMechanisms	Yes	Yes
supportedLDAPVersion	Yes	Yes
dITContentRules	Yes	Yes
creatorsName	No	No
modifiersName	No	No
matchingRules	No	No
matchingRulesUse	No	No
altServer	No	No
ldapSyntaxes	No	No
dITStructureRules	No	No
nameForms	No	No

RFC 2256

Attribute	Included by AD DS?	Included by AD LDS?
objectClass	Yes	Yes
knowledgeInformation	Yes	No
cn	Yes	Yes
sn	Yes	Yes
serialNumber	Yes	Yes
c	Yes	Yes
l	Yes	Yes
st	Yes	Yes
street	Yes	Yes
o	Yes	Yes
ou	Yes	Yes
title	Yes	Yes
description	Yes	Yes

Attribute	Included by AD DS?	Included by AD LDS?
searchGuide	Yes	Yes
businessCategory	Yes	Yes
postalAddress	Yes	Yes
postalCode	Yes	Yes
postOfficeBox	Yes	Yes
physicalDeliveryOfficeName	Yes	Yes
telephoneNumber	Yes	Yes
telexNumber	Yes	Yes
teletexTerminalIdentifier	Yes	Yes
facsimileTelephoneNumber	Yes	Yes
x121Address	Yes	Yes
internationalISDNNumber	Yes	Yes
registeredAddress	Yes	Yes
destinationIndicator	Yes	Yes
preferredDeliveryMethod	Yes	Yes
presentationAddress	Yes	No
supportedApplicationContext	Yes	No
member	Yes	Yes
owner	Yes	Yes
roleOccupant	Yes	No
seeAlso	Yes	Yes
userPassword	Yes*	Yes*
userCertificate	Yes	Yes
cACertificate	Yes	No
authorityRevocationList	Yes	No
certificateRevocationList	Yes	No
crossCertificatePair	Yes	No
name	Yes	Yes
givenName	Yes	Yes
initials	Yes	Yes

Attribute	Included by AD DS?	Included by AD LDS?
generationQualifier	Yes	Yes
x500uniqueIdentifier	Yes	Yes
distinguishedName	Yes	Yes
uniqueMember	Yes	Yes
houseIdentifier	Yes	No
deltaRevocationList	Yes	No
dmdName	Yes	Yes
aliasedObjectName	No	No
dnQualifier	No	No
protocolInformation	No	No
supportedAlgorithms	No	No

* Active Directory uses the userPassword attribute to set or change passwords only in limited circumstances. See section 3.1.1.3.1.5.

RFC 2798

Attribute	Included by AD DS?	Included by AD LDS?
carLicense	Yes	Yes
departmentNumber	Yes	Yes
displayName	Yes	Yes
employeeNumber	Yes	Yes
employeeType	Yes	Yes
jpegPhoto	Yes	Yes
preferredLanguage	Yes	Yes
userSMIMECertificate	Yes	Yes
userPKCS12	Yes	Yes

RFC 2307

Attribute	Included by AD DS?	Included by AD LDS?
uidNumber	Yes	No
gidNumber	Yes	No
gecos	Yes	No

Attribute	Included by AD DS?	Included by AD LDS?
homeDirectory	Yes	No
loginShell	Yes	No
shadowLastChange	Yes	No
shadowMin	Yes	No
shadowMax	Yes	No
shadowWarning	Yes	No
shadowInactive	Yes	No
shadowExpire	Yes	No
shadowFlag	Yes	No
memberUid	Yes	No
memberNisNetgroup	Yes	No
nisNetgroupTriple	Yes	No
ipServicePort	Yes	No
ipServiceProtocol	Yes	No
ipProtocolNumber	Yes	No
oncRpcNumber	Yes	No
ipHostNumber	Yes	No
ipNetworkNumber	Yes	No
ipNetmaskNumber	Yes	No
macAddress	Yes	No
bootParameter	Yes	No
bootFile	Yes	No
nisMapName	Yes	No
nisMapEntry	Yes	No

3.1.1.3.1.1.4 Classes

Section 7 of [RFC2252], as well as section 7 of [RFC2256] and section 3 of [RFC2798], defines a set of classes common to LDAP directories. In addition, portions of the Active Directory schema are derived from [RFC1274] and [RFC2307]. The following tables show, for each of these RFCs, the classes included in the Active Directory default schemas of Windows Server 2003 operating system, Active Directory Application Mode (ADAM), Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system. Some of these classes were added to the schema of Windows

Server 2003 or Windows Server 2003 R2 operating system but were not present in the Windows 2000 operating system schema; [MS-ADSC] specifies the classes included in each version of the schema.

RFC 1274

Class	Included by AD DS?	Included by AD LDS?
top	Yes	Yes
country	Yes	Yes
locality	Yes	Yes
organization	Yes	Yes
organizationalUnit	Yes	Yes
person	Yes	Yes
organizationalPerson	Yes	Yes
organizationalRole	Yes	No
groupOfNames	Yes	Yes
residentialPerson	Yes	No
applicationProcess	Yes	No
applicationEntity	Yes	No
dSA	Yes	No
device	Yes	No
certificationAuthority	Yes	No
account	Yes	No
document	Yes	No
room	Yes	No
documentSeries	Yes	No
domain	Yes	Yes
rFC822LocalPart	Yes	No
domainRelatedObject	Yes	No
friendlyCountry	Yes	No
simpleSecurityObject	Yes	No
Alias	No	No
strongAuthenticationUser	No	No

Class	Included by AD DS?	Included by AD LDS?
mhsDistributionList	No	No
mhsMessageStore	No	No
mhsMessageTransferAgent	No	No
mhsOrganizationalUser	No	No
mhsResidentialUser	No	No
mhsUserAgent	No	No
pilotObject	No	No
pilotPerson	No	No
dNSDomain	No	No
pilotOrganization	No	No
pilotDSA	No	No
qualityLabelledData	No	No

RFC 2252

Class	Included by AD DS?	Included by AD LDS?
subSchema	Yes	Yes
extensibleObject	No	No

RFC 2256

Class	Included by AD DS?	Included by AD LDS?
top	Yes	Yes
country	Yes	Yes
locality	Yes	Yes
organization	Yes	Yes
organizationalUnit	Yes	Yes
person	Yes	Yes
organizationalPerson	Yes	Yes
organizationalRole	Yes	No
groupOfNames	Yes	Yes
residentialPerson	Yes	No
applicationProcess	Yes	No

Class	Included by AD DS?	Included by AD LDS?
applicationEntity	Yes	No
dSA	Yes	No
device	Yes	No
certificationAuthority	Yes	No
groupOfUniqueNames	Yes	No
cRLDistributionPoint	Yes	No
dMD	Yes	Yes
alias	No	No
strongAuthenticationUser	No	No
userSecurityInformation	No	No
certificationAuthority-V2	No	No

RFC 2798

Class	Included by AD DS?	Included by AD LDS?
inetOrgPerson	Yes	Yes

RFC 2307

Class	Included by AD DS?	Included by AD LDS?
posixAccount	Yes	No
shadowAccount	Yes	No
posixGroup	Yes	No
ipService	Yes	No
ipProtocol	Yes	No
oncRpc	Yes	No
ipHost	Yes	No
ipNetwork	Yes	No
nisNetgroup	Yes	No
nisMap	Yes	No
nisObject	Yes	No
ieee802Device	Yes	No
bootableDevice	Yes	No

3.1.1.3.1.1.5 Auxiliary Classes

Windows 2000 operating system had limited support for LDAP auxiliary classes. An auxiliary class would be associated with the schema definition of a particular class C when the auxiliary class was added to the auxiliaryClass or systemAuxiliaryClass attribute of the classSchema object that defines C. In this case, all instances of C will inherit the attributes of the auxiliary class.

The server permits adding or removing an auxiliary class to or from the auxiliaryClass attribute of C at any point in time. Doing so adds or removes the auxiliary class from every existing instance of C but does not cause the object class of the auxiliary class to appear in the objectClass attribute of those instances. Such an auxiliary class can have optional (mayContain) attributes but not mandatory (mustContain) attributes. This is because there can be existing instances of C, in which case adding a new mandatory attribute would cause those existing instances to violate the modified schema.

The server permits adding an auxiliary class to the systemAuxiliaryClass attribute of C only when C is defined, that is, when C's classSchema object is added to the schema NC. After a classSchema object has been created, its systemAuxiliaryClass attribute cannot be modified. An auxiliary class that is associated with C by the addition of it to C!systemAuxiliaryClass can have mandatory (mustContain) as well as optional (mayContain) attributes. As in the previous case, the auxiliary classes added in this manner are not shown in the objectClass attribute of the instances of C.

Dynamic auxiliary class support was introduced in Windows Server 2003 operating system in addition to the Windows 2000 auxiliary class mechanism, and continues to be supported in Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system. This dynamic auxiliary class mechanism reflects the model of auxiliary object classes described in [X501] section 8.3.3. The server permits adding an auxiliary class to any instance I of a class by a request to add that auxiliary class to I!objectClass. This will cause only that instance I to inherit the attributes of the auxiliary class. The dynamic auxiliary class will be removed from I, after the values of all attributes in the auxiliary class have been cleared by the client, by a request to remove the auxiliary class from I!objectClass. Dynamic auxiliary classes can have both mandatory (mustContain) and optional (mayContain) attributes.

If the dynamic auxiliary class that is added to I is a subclass of another auxiliary class, both auxiliary classes are added to I when the child auxiliary class is added to I. However, removing the child auxiliary class does not cause the server to remove its parent from I. A parent auxiliary class can be removed from I only when all child auxiliary classes that inherit from the parent are also removed from I.

For each I, I!objectClass contains the structural, abstract, and dynamic **auxiliary object classes** of which I is an instance (and their inheritance chains). I!structuralObjectClass includes only the structural class of which I is an instance and its inheritance chain. I!msDS-Auxiliary-Classes contains the dynamic auxiliary classes of which I is an instance along with their inheritance chain, except it does not include those classes in the inheritance chain that are in I!structuralObjectClass.

3.1.1.3.1.2 Object Naming

This section discusses the naming of objects via distinguished names in Active Directory, as it differs from the appropriate RFCs.

3.1.1.3.1.2.1 Naming Attributes

As with [RFC2253] section 2.3, Active Directory permits any attribute to be used as the AttributeType in an RDN. However, Active Directory imposes the additional restriction that the

AttributeType used must be of String(Unicode) syntax. Furthermore, all objects of the same class use the same attribute in their RDN. The attribute to be used in the RDN is specified by the rDNAttID attribute in the classSchema object that defines the class. The rDNAttID attribute contains the attribute to be used in the RDN. Multivalued RDNs are not permitted (see section 3.1.1.3.1.2.3), so if the attribute A specified by rDNAttID is multivalued, an attempt to add an additional value to A on an object O for which O!rDNAttID = A is rejected with the error *invalidDNSyntax* / *ERROR_DS_BAD_NAME_SYNTAX* if it takes place at the time of the object's creation, or the error *notAllowedOnRDN* / <*unrestricted*> if it takes place in a subsequent LDAP Modify operation.

The AttributeValue of the RDN must be unique among sibling objects. For example, the following two DNs cannot coexist in the directory, because two identical AttributeValues ("Abc") would exist in the same container ("OU=Users,DC=Fabrikam,DC=com"):

- CN=Abc,OU=Users,DC=Fabrikam,DC=com
- L=Abc,OU=Users,DC=Fabrikam,DC=com

The server will reject an attempt to create such a non-uniquely named object with the error *entryAlreadyExists* / <*unrestricted*>. This requirement for unique AttributeValues guarantees the uniqueness of canonical names.

3.1.1.3.1.2.2 NC Naming

The DN of a domain NC is derived from the DNS name of the domain using the transformation algorithm of [RFC2247] section 3. The object at the root of each domain NC is a domainDNS object, in accord with section 5.2 of that RFC. The rDNAttID for the domainDNS class is dc, in accord with section 4 of the RFC. While the same attribute OID is used for the dc attribute in Active Directory as in section 4 of the RFC, the syntax of the attribute in Active Directory is String(Unicode) rather than the specified String(IA5). The dcObject auxiliary class, specified in section 5.1 of the RFC, is not present in Active Directory.

When operating as AD DS, the DN for the config NC is the RDN "CN=Configuration", followed by the DN of the domain NC of the forest root domain. When operating as AD LDS, the DN for the config NC is the RDN "CN=Configuration, CN={guid}", where **guid** is a GUID in dashed-string form ([RFC4122] section 3). For example,

CN=Configuration, CN={FD783EE9-0216-4B83-8A2A-60E45AECCB81}

is a possible DN of the config NC when operating as AD LDS.

The DN for the schema NC is the RDN "CN=Schema" followed by the DN of the config NC.

When operating as AD DS, an application NC is named in the same way as a domain NC; the root of each AD DS application NC is a domainDNS object. When operating as AD LDS, the DN of an application NC consists of one or more RDNs.

3.1.1.3.1.2.3 Multivalued and Multiple-Attribute RDNs

[RFC2253] section 2 defines the following grammar rule for RelativeDistinguishedName, which explicitly allows RDNs to contain multiple attributes and values:

- RelativeDistinguishedName ::= SET SIZE (1..MAX) OF AttributeTypeAndValue

Active Directory is conformant with this rule, with the restriction that MAX equals 1 within the scope of the rule. As a result, multivalued RDNs that consist of multiple attributes (sometimes referred to as "multi-AVA RDNs"), or multiple instances of the same attribute, are both disallowed in Active

Directory. An attempt to create such a DN is considered an attempt to create a syntactically invalid DN, and returns the error *invalidDNSyntax* / *ERROR_DS_BAD_NAME_SYNTAX*. For example, assuming that F is a multivalued attribute of String(Unicode) syntax, the following two DNs are both disallowed because they contain multivalued RDNs:

- F=John Smith+F=David Jones, OU=Users,DC=Fabrikam,DC=com
- F=John Smith+l=Redmond, OU=Users,DC=Fabrikam,DC=com

(Note that, if it is assumed that these DNs represent an object of a class C for which C!rDNAttID = F, the second example is also disallowed because it contains the *l* attribute in the RDN. The server will return a *namingViolation* / <*unrestricted*> error when an attempt is made to add an **object of class** C whose RDN contains a different AttributeType than that declared in C!rDNAttID.)

3.1.1.3.1.2.4 Alternative Forms of DNs

In addition to the form of the DN defined in [RFC2253], Active Directory supports several alternative forms of DNs that can be used to specify objects in requests sent to the DC, for example, as the baseObject in a SearchRequest or as a AttributeValue in a ModifyRequest.

The first alternative form is in the format

```
<GUID=object_guid>
```

where **object_guid** is a GUID that corresponds to the value of the objectGUID attribute of the object being specified. All DCs support **object_guid** expressed as the hexadecimal representation of the binary form of a GUID ([MS-DTYP] section 2.3.4). Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system DCs also support the dashed-string form of a GUID ([RFC4122] section 3).

The second alternative form is in the format

```
<SID=sid>
```

where **sid** is the security identifier (SID) that corresponds to the value of the objectSid attribute of the object being specified. The **sid** is expressed as either the hexadecimal representation of a binary SID structure ([MS-DTYP] section 2.4.2.2) in little-endian byte order, or as a SID string ([MS-DTYP] section 2.4.2.1). Windows 2000 operating system DCs support only the hexadecimal representation.

The third alternative form is in the format

```
<WKGUID=guid, object_DN>
```

where **guid** is a GUID expressed as the hexadecimal representation of the binary form of the GUID. A DN of this form is resolved to an object O by applying the following algorithm.

```
MapWellKnownGuidToDN(GUID guid, DN object_DN)
```

This algorithm resolves a well-known GUID, expressed as a GUID, **guid**, and an object, **object_DN**, into the DN of the object O that is identified by that well-known GUID.

- If **object_DN** does not name an object in the directory, reject the DN.
- Otherwise, let C be the object named by **object_DN**.
- If there exists a value V in C!wellKnownObjects such that the binary portion of V contains the same GUID as **guid**, then the DN of O is the DN portion of V.
- Otherwise, if there exists a value V' in C!otherWellKnownObjects such that the binary portion of V' contains the same GUID as **guid**, then the DN of O is the DN portion of V'.
- Otherwise, reject the DN.

Normally, Active Directory will return DNs in the [RFC2253] format. However, clients can request that Active Directory return DNs in the "extended DN" format. This format combines an RFC 2253-style DN with a representation of the object's objectGUID and objectSid attributes. This form is documented in the LDAP section 3.1.1.3.4.1.5, which defines the LDAP_SERVER_EXTENDED_DN_OID control that is used by the client to request that the DC use the "extended DN" form when returning DNs. The "extended DN" form is not accepted as a means of specifying DNs in requests sent to the DC. The "extended DN" form is only used in LDAP responses from the DC, and only when the LDAP_SERVER_EXTENDED_DN_OID control is used to request such a form.

3.1.1.3.1.2.5 Alternative Form of SIDs

Attributes of String(SID) syntax contain a SID in binary form. However, a client may instead specify a value for such an attribute as a UTF-8 string that is a valid SDDL SID string beginning with "S-" (see [MS-DTYP] section 2.4.2.1). The server will convert such a string to the binary form of the SID and use that binary form as the value of the attribute.

3.1.1.3.1.3 Search Operations

3.1.1.3.1.3.1 Search Filters

Active Directory does not support the extensible match rules defined in [RFC2252] section 8, [RFC2256] section 8, and [RFC2798] section 9. Active Directory exposes three extensible match rules that are defined in section 3.1.1.3.4.4. Other than these three rules, the rules that Active Directory uses for comparing values (for example, comparing two String(Unicode) attributes for equality or ordering) are not exposed as extensible match rules. These comparison rules are documented for each syntax type in section 3.1.1.2.2.4. When performing an extensible match search against Active Directory, if the type field of the MatchingRuleAssertion is not specified ([RFC2251] section 4.5.1), the extensible match filter clause is evaluated to "Undefined". The dnAttributes field of the MatchingRuleAssertion is ignored and always treated as if set to false.

Active Directory supports the approxMatch filter clause of [RFC2251] section 4.5.1. However, it is implemented identically to equalityMatch; for example, the filter is true if the values are equal. No approximation is performed. Filter clauses of the form "(X=Y)" and "(X~=Y)" may be freely substituted for each other.

Active Directory in Windows 2000 operating system does not implement three-value logic for search filter evaluation as defined in [RFC2251] section 4.5.1. In Windows 2000, filters evaluate to either "true" or "false". Filters that would evaluate to "Undefined", as per the RFC, are instead evaluated to "false". Active Directory in Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system uses three-value logic for evaluating search filters, in conformance with the RFC.

Active Directory does not support constructed attributes (defined in section 3.1.1.4.5) in search filters. When a search operation is performed with such a search filter, Active Directory fails with inappropriateMatching ([RFC2251] section 4.1.10).

3.1.1.3.1.3.2 Selection Filters

Active Directory supports the ability to filter the values of an attribute that are returned. By default, all values up to the default range of a given attribute are returned. A selection filter is used to filter values to be returned by the server. When no selection filter is specified, the returned values of an attribute MUST NOT be filtered. An explicit selection filter specifies the filtering on the attribute values to be returned by the server.

Selection filtering is requested by specifying an Attribute Description ([RFC2251] section 4.1.5) with the "filtered" option. This option takes the form:

filtered=B:**char_count**:**binary_value**

where **char_count** is the number (in decimal) of hexadecimal digits in **binary_value** and **binary_value** is the hexadecimal representation of a binary value. Each byte is represented by a pair of hexadecimal characters in **binary_value**, with the first character of each pair corresponding to the most-significant nibble of the byte. The first pair in **binary_value** corresponds to the first byte of the binary value, with subsequent pairs corresponding to the remaining bytes in sequential order. Note that **char_count** is always even in a syntactically valid selection filter.

The binary value is a BER encoded filter, as specified in [RFC2251] section 4.5.1.

Selection filters are available in DCs with a functional level of DS_BEHAVIOR_WIN2012R2 or greater.

3.1.1.3.1.3.3 Range Retrieval of Attribute Values

When retrieving the values from a multivalued attribute, Active Directory limits the number of values that can be retrieved from one attribute in a single search request. The maximum number of values that will be returned by Active Directory at one time is determined by the MaxValRange policy (see section 3.1.1.3.4.6). To permit all the values of a multivalued attribute to be retrieved, Active Directory provides a "range retrieval" mechanism. This mechanism permits a client-specified subset of the values to be retrieved in a search request. By performing multiple search requests, each retrieving a distinct subset, the complete set of values for the attribute can be retrieved.

Range retrieval is requested by attaching a range option to the name of the attribute (for example, the AttributeDescription, as specified in [RFC2251] section 4.1.5) to be retrieved by the search request. This option takes the form

range=**low**-**high**

where **low** is the zero-based index of the first value of the attribute to retrieve, and **high** is the zero-based index of the last value of the attribute to retrieve. For example, to retrieve the 100th through the 500th values of the member attribute, the attributes list in the SearchRequest would specify the AttributeDescription "member;range=99-499". Zero is used for **low** to specify the first entry. A client can substitute an asterisk for **high** to indicate all remaining entries (subject to any limitations imposed by the server on the maximum number of values to return). The server may return fewer values than requested.

When the server receives a range retrieval request, it will include a range option in the AttributeDescription returned. This range option will take the same form as described previously, with **low** indicating the zero-based index of the first value of the attribute that the server returned

and **high** indicating the zero-based index of the last value of the attribute that the server returned. However, if the set of attributes returned includes the last value in the attribute, the server will substitute an asterisk for **high**, indicating to the client that there are no more values to be retrieved.

If a SearchRequest does not contain a range option for a given attribute, but that attribute has too many values to be returned at one time, the server returns a SearchResultEntry containing (1) the attribute requested without the range option and with no values, and (2) the attribute requested with a range option attached and with the values corresponding to that range option.

The ordering of the values returned in a range retrieval request is arbitrary but consistent across multiple range retrieval requests on the same **LDAP connection**, provided that the attribute is not modified between successive range retrieval requests.

3.1.1.3.1.3.4 Ambiguous Name Resolution

ANR is a search algorithm in Active Directory that permits a client to search multiple naming-related attributes on objects via a single clause in a search filter. A substring search against the aNR attribute is interpreted by the DC as a substring search against a set of attributes, known as the "ANR attribute set". The intent is that the attributes in the ANR attribute set are those attributes that are commonly used to identify an object, such as the displayName and name attributes, thereby permitting a client to query for an object when the client possesses some identifying material related to the object but does not know the attribute of the object that contains that identifying material. The ANR attribute set consists of those attributes whose searchFlags attribute contains the fANR flag (see section 3.1.1.2.3).

A server performs an ANR search by rewriting a search filter that contains one or more occurrences of the aNR attribute so that the filter no longer contains any occurrences of the aNR attribute, then performing a regular LDAP search using the rewritten search filter. The search filter is rewritten according to the following algorithm:

1. If the ANR attribute set does not contain the attribute legacyExchangeDN, then let S be the ANR attribute set and let PLegacy be false. Otherwise, let S be the ANR attribute set excluding legacyExchangeDN and let PLegacy be true. In either case, S is a set containing attributes A1...An.

2. Let P1 be the value of the fSupFirstLastANR heuristic of the dSHeuristics attribute (see section 6.1.1.2.4.1.2). Let P2 be the value of the fSupLastFirstANR heuristic of the dSHeuristics attribute.

3. Let F be the search filter of the search request.

4. For each LDAP search filter clause C of the form "(aNR=*)" in F, resolve the clause to "false". (Such a clause tests for the presence of a value for the aNR attribute itself, and this attribute is not present on any object.)

5. For each LDAP search filter clause C of the form "(aNR=**substringFilter**)", where **substringFilter** is an LDAP substring filter of the form "*i***f*", in F:

    1. If *i* is the empty string, resolve clause C to the value "Undefined" (see [RFC2251] section 4.5.1).

    2. If *i* is non-empty, replace clause C with the clause "(aNR=*i*)" and apply the rule for "(aNR=**value**) in the next step of this algorithm.

6. For each LDAP search filter clause C of the form "(aNR=**value**)" or "(aNR~=**value**)" or "(aNR>=**value**)" or "(aNR<=**value**)" in F:

1. If **value**'s first non-space character is an equal sign ("=") similar to "=**value1**" or " =**value1**", it is used for an exact string search instead of a substring search. Set "**value**" to "**value1**", apply the following steps in rule 6, and replace all the "**value***" with "**value**".

2. If **value** does not contain any space characters, or if P1 is true and P2 is true, construct an LDAP search filter clause C' of the form "(|(A1=**value***)...(An=**value***))" if PLegacy is false, or of the form "(|(A1=**value***)...(An=**value***)(legacyExchangeDN=**value**)" if PLegacy is true. (This clause resolves to "true" for an object if **value** is a prefix of the value of any attribute in the ANR set on that object, except an exact match is always performed on the legacyExchangeDN attribute.)

3. If **value** does contain one or more space characters, then:

    1. Split **value** into two components, **value1** and **value2**, at the location of the first space, discarding that space.

    2. If PLegacy is false, do the following:

        1. If P1 is false and P2 is false, then construct an LDAP search filter clause C' of the form "(|(A1=**value***)...(An=**value***)(&(givenName=**value1***) (sn=**value2***)) (&(givenName=**value2***)(sn=**value1***)))". (This clause resolves to "true" for an object if **value** is a prefix of the value of any attribute in the ANR set on that object, or if the two parts of the split **value** are prefixes of the givenName and sn attributes on that object, regardless of which part matches which attribute.)

        2. If P1 is true and P2 is false, then construct an LDAP search filter clause C' of the form "(|(A1=**value***)...(An=**value***)(&(givenName=**value2***) (sn=**value1***)))". (This clause will resolve to "true" for an object if **value** is a prefix of the value of any attribute in the ANR set on that object, or if the first part of the split **value** is a prefix of the sn attribute and the second part is a prefix of the givenName attribute on that object.)

        3. If P1 is false and P2 is true, then construct an LDAP search filter clause C' of the form "(|(A1=**value***)...(An=**value***)(&(givenName=**value1***) (sn=**value2***)))". (This clause will resolve to "true" for an object if **value** is a prefix of the value of any attribute in the ANR set on that object, or if the first part of the split **value** is a prefix of the givenName attribute and the second part is a prefix of the sn attribute on that object.)

    3. If PLegacy is true, do the following:

        1. If P1 is false and P2 is false, then construct an LDAP search filter clause C' of the form "(|(A1=**value***)...(An=**value***)(legacyExchangeDN=**value**)(&(givenName=**value1***) (sn=**value2***)) (&(givenName=**value2***)(sn=**value1***)))". (This clause resolves to "true" for an object if **value** equals the value of legacyExchangeDN on that object or **value** is a prefix of the value of any attribute in the ANR set on that object, or if the two parts of the split **value** are prefixes of the givenName and sn attributes on that object, regardless of which part matches which attribute.)

        2. If P1 is true and P2 is false, then construct an LDAP search filter clause C' of the form "(|(A1=**value***)...(An=**value***)(legacyExchangeDN=**value**) (&(givenName=**value2***) (sn=**value1***)))". (This clause will resolve to "true" for an object if **value** equals the value of legacyExchangeDN on that object or **value** is a prefix of the value of any attribute in the ANR set on that object, or if the first part of the split **value** is a prefix of the sn attribute and the second part is a prefix of the givenName attribute on that object.)

        3. If P1 is false and P2 is true, then construct an LDAP search filter clause C' of the form "(|(A1=**value***)...(An=**value***)(legacyExchangeDN=**value**) (&(givenName=**value1***)

(sn=**value2***)))". (This clause will resolve to "true" for an object if **value** equals the value of legacyExchangeDN on that object or **value** is a prefix of the value of any other attribute in the ANR set on that object, or if the first part of the split **value** is a prefix of the givenName attribute and the second part is a prefix of the sn attribute on that object.)

4. Remove clause C from F, and insert C' into F at the position vacated by C.

Note that the replacement clause C' always contains equality matches, regardless of the type of match in the original clause C.

3.1.1.3.1.3.5 Searches Using the objectCategory Attribute

When an LDAP search filter F contains a clause C of the form "(objectCategory=**V**)", if **V** is not a DN but there exists an object O such that O!objectClass = classSchema and O!lDAPDisplayName = **V**, then the server treats the search filter as if clause C was replaced in F with the clause "(objectCategory=**V'**)", where **V'** is O!defaultObjectCategory.

For example, if the LDAP search filter contains clause "(objectCategory=contact)", because the defaultObjectCategory of class contact is CN=person,CN=schema,CN=configuration,DC=Fabrikam,DC=com, Active Directory will treat the clause as "(objectCategory=CN=person,CN=schema,CN=configuration,DC=Fabrikam,DC=com)".

3.1.1.3.1.3.6 Restrictions on rootDSE Searches

When performing a search against the rootDSE and specifying a list of attributes to be returned, the attributes to be returned must be specified by their LDAP display name. Specifying the attribute by their numeric OID will be treated by the server the same as specifying a nonexistent attribute. The server supports specifying the attributes to be returned by their numeric OIDs in searches that do not use the rootDSE as the search base.

When performing a search against the rootDSE, the server will ignore the contents of the search filter, except as noted in section 6.3.

3.1.1.3.1.4 Referrals in LDAPv2 and LDAPv3

When using the LDAPv3 protocol, Active Directory returns referrals and continuation references in accord with [RFC2251] section 4.5.3. When using the LDAPv2 protocol, Active Directory also returns referrals and continuation references, although these are not part of the LDAPv2 protocol, as defined in [RFC1777].

When Active Directory generates a referral in the LDAPv2 protocol, it sets the resultCode field in the LDAPResult structure (defined in [RFC1777]) to the value 9. This is a value not defined in [RFC1777] or [RFC2251] but that, by convention, is used by LDAPv2 servers to indicate the presence of a referral in the response.

The contents of the referral are conveyed in the errorMessage field of the LDAPResult. This field consists of the string "Referral:", followed by a newline character, followed by one or more LDAPURLs (defined in [RFC2255]). Each LDAPURL is separated by a newline character. The meaning of these LDAPURLs is equivalent to that of an LDAPURL in an LDAPv3 referral; that is, they indicate a server or servers against which the operation can be retried.

Active Directory uses the same mechanism to return continuation references in LDAPv2. When a continuation reference is required, the DC will return a SearchResponse message (defined in [RFC1777]) in which the resultCode and errorMessage fields in the embedded LDAPResult are set as described previously for LDAPv2 referrals. As with the LDAPv2 referrals, the meaning of the

LDAPURLs embedded in the errorMessage field is equivalent to their LDAPv3 equivalent; that is, they indicate another server or NC in which the search can be continued.

3.1.1.3.1.5 Password Modify Operations

Active Directory provides the ability to change a security principal's password (that is, the Windows password for that security principal) by performing LDAP Modify operations. The password change is modeled as an LDAP modify of either the unicodePwd or userPassword attribute of the security principal object. The difference between these two attributes is discussed in the sections that follow. However, regardless of whether the password is modified via unicodePwd or userPassword, the same attribute on the object is modified. If running as AD DS, both are treated like a write to the clearTextPassword attribute in [MS-SAMR] section 3.1.1.8.5. If running as AD LDS, a write to userPassword updates unicodePwd.

3.1.1.3.1.5.1 unicodePwd

Active Directory stores the password on a user object or inetOrgPerson object in the unicodePwd attribute. This attribute is written by an LDAP Modify under the following restricted conditions. Windows 2000 operating system servers require that the client have a 128-bit (or better) SSL/TLS-encrypted connection to the DC in order to modify this attribute. On Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, the DC also permits modification of the unicodePwd attribute on a connection protected by 128-bit (or better) **SASL**-layer encryption instead of SSL/TLS. In Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2, if the fAllowPasswordOperationsOverNonSecureConnection heuristic of the dSHeuristics attribute (section 6.1.1.2.4.1.2) is true and Active Directory is operating as AD LDS, then the DC permits modification of the unicodePwd attribute over a connection that is neither SSL/TLS-encrypted nor SASL-encrypted. The unicodePwd attribute is never returned by an LDAP search.

When a DC receives an LDAP Modify request to modify this attribute, it follows the following procedure:

- If the Modify request contains a delete operation containing a value **Vdel** for unicodePwd followed by an add operation containing a value **Vadd** for unicodePwd, the server considers the request to be a request to change the password. The server decodes **Vadd** and **Vdel** using the password decoding procedure documented later in this section. **Vdel** is the old password, while **Vadd** is the new password.
- If the Modify request contains a single replace operation containing a value **Vrep** for unicodePwd, the server considers the request to be a administrative reset of the password, that is, a password modification without knowledge of the old password. The server decodes **Vrep** using the password decoding procedure documented later in this section and uses it as the new password.

For the password change operation to succeed, the server enforces the requirement that the user or inetOrgPerson object whose password is being changed must possess the "User-Change-Password" **control access right** on itself, and that **Vdel** must be the current password on the object. For the password reset to succeed, the server enforces the requirement that the client possess the "User-Force-Change-Password" control access right on the user or inetOrgPerson object whose password is to be reset.

The syntax of the unicodePwd attribute is Object(Replica-Link). However, the DC requires that the password value be specified in a **UTF-16** encoded Unicode string containing the password surrounded by quotation marks, which has been BER-encoded as an octet string per the

Object(Replica-Link) syntax. BER encoding and decoding is defined in [ITUX690]. To decode such a value **V**, the server follows this password decoding procedure:

- If **V** is not a valid BER-encoding of an octet string, reject the password operation with the error *protocolError* / *ERROR_DS_DECODING_ERROR*.
- BER-decode **V** to produce **Vdecoded**.
- If the first and last characters of **Vdecoded** are not the UTF-16 Unicode representation of quotation marks, reject the password operation with the error *constraintViolation*/ *ERROR_DS_UNICODEPWD_NOT_IN_QUOTES*.
- Remove the first and last characters from **Vdecoded** to produce **Vpassword**.

Vpassword is the value the DC uses for the password—the actual password, not a password hash. This encoding is used for both the old and the new passwords in a password change request.

Following is an example of the first steps of password encoding. Suppose the implementer wants to set unicodePwd to the string "new".

```
ASCII "new":      0x6E 0x65 0x77
UTF-16 "new":     0x6E 0x00 0x65 0x00 0x77 0x00
UTF-16 "new"
    with quotes: 0x22 0x00 0x6E 0x00 0x65 0x00 0x77 0x00 0x22 0x00
```

The 10-byte octet string is then BER-encoded and sent in an LDAP Modify request as described previously.

3.1.1.3.1.5.2 userPassword

Active Directory supports modifying passwords on objects via the userPassword attribute, provided that (1) either the DC is running as AD LDS, or the DC is running as AD DS and the **domain functional level** is DS_BEHAVIOR_WIN2003 or greater, and (2) the fUserPwdSupport heuristic is true in the dSHeuristics attribute (section 6.1.1.2.4.1.2). If fUserPwdSupport is false, the userPassword attribute is treated as an ordinary attribute and has no special semantics associated with it. If fUserPwdSupport is true but the DC is running as AD DS and the domain functional level is less than DS_BEHAVIOR_WIN2003, the DC fails the operation with the error *constraintViolation* / *ERROR_NOT_SUPPORTED*.

As with the unicodePwd attribute, changing a password via the userPassword attribute is modeled as an LDAP Modify operation containing a Delete operation followed by an Add operation, and resetting a password is modeled as an LDAP Modify operation containing a single Replace operation. The control access rights required are the same as for the unicodePwd attribute, as is the requirement that when changing a password, **Vdel** must match the object's current password.

The special encoding required for updating the unicodePwd attribute is not used with the userPassword attribute; that is, **Vpassword** = **V**. The same restrictions on SSL/TLS- or SASL-protected connections are enforced. The password values are sent to the server as UTF-8 strings, and surrounding quotation marks are not used. For example, the following LDAP Data Interchange Format (LDIF) sample changes a password from oldPassword to newPassword.

```
dn: CN=John Smith, OU=Users,DC=Fabrikam,DC=com
changetype: modify
delete: userPassword
userPassword: oldPassword
```

```
-
add: userPassword
userPassword: newPassword
-
```

The following example uses LDIF to reset the password to newPassword.

```
dn: CN=John Smith, OU=Users,DC=Fabrikam,DC=com
changetype: modify
replace: userPassword
userPassword: newPassword
-
```

Optionally, when performing a password change operation, the add operation portion of the LDAP modify can be omitted. The server treats this as a request to change the user or inetOrgPerson object's password to the empty string.

3.1.1.3.1.6 Dynamic Objects

The Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system versions of Active Directory have support for dynamic objects, as specified in [RFC2589]. The Active Directory implementation is conformant to that RFC, except that it does not implement the *dynamicSubtrees* attribute used to represent which NCs support dynamic objects.

Dynamic objects are supported in all NCs except for the schema NC and the config NC. A dynamic object cannot be the parent of an object that is not dynamic, and the server will reject such a request with the error *unwillingToPerform* / *ERROR_DS_UNWILLING_TO_PERFORM*. When a dynamic object reaches the end of its time-to-live, the object is expunged from the directory by the server and does not leave behind a tombstone.

3.1.1.3.1.7 Modify DN Operations

Because Active Directory does not support multivalued RDNs (see section 3.1.1.3.1.2.3), the **deleteoldrdn** field of a ModifyDNRequest (defined in [RFC2251] section 4.9) must always be set to true. If **deleteoldrdn** is set to false, the server fails the request with the error *unwillingToPerform* / *ERROR_INVALID_PARAMETER*.

3.1.1.3.1.8 Aliases

LDAP aliases, the class for which is defined in [RFC2256] section 7.2 and which are discussed in [RFC2251] section 4.1.10, are not supported in Active Directory.

3.1.1.3.1.9 Error Message Strings

When the server fails an LDAP operation with an error, and the server has sufficient resources to compute a string value for the **errorMessage** field of the LDAPResult, it includes a string in the **errorMessage** field of the LDAPResult (see [RFC2251] section 4.1.10). The string contains further information about the error.

The first eight characters of the **errorMessage** string are a 32-bit integer, expressed in hexadecimal. Where protocol specifies the extended error code "<unrestricted>" there is no restriction on the value of the 32-bit integer. It is recommended that implementations use a Windows error code for the 32-bit integer in this case in order to improve usability of the directory for clients. Where protocol specifies an extended error code which is a Windows error code, the 32-bit integer is the specified Windows error code. Any data after the eighth character is strictly informational and used only for debugging. Conformant implementations need not put any value beyond the eighth character of the **errorMessage** field.

When the server returns a referral and not an error, the **errorMessage** field is used as described in section 3.1.1.3.1.1.4.

3.1.1.3.1.10 Ports

An AD DS DC accepts LDAP connections on the standard LDAP and LDAPS (LDAP over SSL/TLS) ports: 389 and 636. If the AD DS DC is a GC server, it also accepts LDAP connections for GC access on port 3268 and LDAPS connections for GC access on port 3269.

An AD LDS DC accepts LDAP and LDAPS connections on ports that are configured when creating the DC.

3.1.1.3.1.11 LDAP Search Over UDP

Active Directory supports search over UDP only for searches against rootDSE. It encodes the results of an LDAP search performed over UDP in the same manner as it does a search performed over TCP; specifically, as one or more SearchResultEntry messages followed by a SearchResultDone message, as described in [RFC2251]. This means that the search response is not encoded as described in [RFC1798]. Only LDAP search and LDAP abandon operations are supported over UDP by Active Directory.

3.1.1.3.1.12 Unbind Operation

Upon receipt of an unbind request on an LDAP connection, all outstanding requests on the connection are abandoned, and the Active Directory DC closes the connection.

3.1.1.3.2 rootDSE Attributes

This section specifies the readable attributes on the rootDSE of Windows 2000 operating system, Windows Server 2003 operating system, Active Directory Application Mode (ADAM), Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system DCs (both AD DS and AD LDS).

All of these rootDSE attributes are read-only; an LDAP request to modify any of them will be rejected with the error *unwillingToPerform* / *<unrestricted>*.

The rootDSE attributes are not described by the schema, but occurrences of rootDSE attribute names are underlined in this document as per the convention for any other LDAP attribute.

The following table specifies which of these rootDSE attributes are supported by each Windows Server operating system or ADAM version.

Attribute name	Windows 2000	Windows Server 2003	ADAM	Windows Server 2008 ADDS	Windows Server 2008 ADLDS	Windows Server 2008 R2 ADDS	Windows Server 2008 R2 ADLDS	Windows Server 2012 ADDS	Windows Server 2012 ADLDS	Windows Server 2012 R2 ADDS	Windows Server 2012 R2 ADLDS
configurationNamingContext	X	X	X	X	X	X	X	X	X	X	X
currentTime	X	X	X	X	X	X	X	X	X	X	X
defaultNamingContext	X	X	X	X	X	X	X	X	X	X	X
dNSHostName	X	X	X	X	X	X	X	X	X	X	X
dsSchemaAttrCount	X	X	X	X	X	X	X	X	X	X	X
dsSchemaClassCount	X	X	X	X	X	X	X	X	X	X	X
dsSchemaPrefixCount	X	X	X	X	X	X	X	X	X	X	X
dsServiceName	X	X	X	X	X	X	X	X	X	X	X
highestCommittedUSN	X	X	X	X	X	X	X	X	X	X	X
isGlobalCatalogReady	X	X		X		X		X		X	
isSynchronized	X	X	X	X	X	X	X	X	X	X	X
ldapServiceName	X	X		X		X		X		X	
namingContexts	X	X	X	X	X	X	X	X	X	X	X
netlogon	X	X		X		X		X		X	
pendingPropagations	X	X	X	X	X	X	X	X	X	X	X
rootDomainNamingContext	X	X		X		X		X		X	
schemaNami	X	X	X	X	X	X	X	X	X	X	X

Attribute name	Windows 2000	Windows Server 2003	ADAM	Windows Server 2008 ADDS	Windows Server 2008 ADLDS	Windows Server 2008 R2 ADDS	Windows Server 2008 R2 ADLDS	Windows Server 2012 ADDS	Windows Server 2012 ADLDS	Windows Server 2012 R2 ADDS	Windows Server 2012 R2 ADLDS
ngContext											
serverName	X	X	X	X	X	X	X	X	X	X	X
subschemaSubentry	X	X	X	X	X	X	X	X	X	X	X
supportedCapabilities	X	X	X	X	X	X	X	X	X	X	X
supportedControl	X	X	X	X	X	X	X	X	X	X	X
supportedLDAPPolicies	X	X	X	X	X	X	X	X	X	X	X
supportedLDAPVersion	X	X	X	X	X	X	X	X	X	X	X
supportedSASLMechanisms	X	X	X	X	X	X	X	X	X	X	X
domainControllerFunctionality		X	X	X	X	X	X	X	X	X	X
domainFunctionality		X		X		X		X		X	
forestFunctionality		X	X	X	X	X	X	X	X	X	X
msDS-ReplAllInboundNeighbors		X	X	X	X	X	X	X	X	X	X
msDS-ReplAllOutboundNeighbors		X	X	X	X	X	X	X	X	X	X
msDS-ReplConnectionFailures		X	X	X	X	X	X	X	X	X	X
msDS-ReplLinkFailures		X	X	X	X	X	X	X	X	X	X

Attribute name	Windows 2000	Windows Server 2003	ADAM	Windows Server 2008 AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 AD DS	Windows Server 2012 R2 AD LDS
msDS-ReplPendingOps		X	X	X	X	X	X	X	X	X	X
msDS-ReplQueueStatistics		X	X	X	X	X	X	X	X	X	X
msDS-TopQuotaUsage		X	X	X	X	X	X	X	X	X	X
supportedConfigurableSettings		X	X	X	X	X	X	X	X	X	X
supportedExtension		X	X	X	X	X	X	X	X	X	X
validFSMOs		X	X	X	X	X	X	X	X	X	X
dsaVersionString			X	X	X	X	X	X	X	X	X
msDS-PortLDAP			X	X	X	X	X	X	X	X	X
msDS-PortSSL			X	X	X	X	X	X	X	X	X
msDS-PrincipalName			X	X	X	X	X	X	X	X	X
serviceAccountInfo			X	X	X	X	X	X	X	X	X
spnRegistrationResult			X	X	X	X	X	X	X	X	X
tokenGroups			X	X	X	X	X	X	X	X	X
usnAtRifm				X	X	X	X	X	X	X	X

The following table shows, for each rootDSE attribute, whether or not the attribute is operational (that is, whether the server returns the attribute only when it is explicitly requested) and the LDAP syntax of the returned value.

Attribute name	Operational?	LDAP syntax
configurationNamingContext	N	Object(DS-DN)
currentTime	N	String(Generalized-Time)
defaultNamingContext	N	Object(DS-DN)
dNSHostName	N	String(Unicode)
dsSchemaAttrCount	Y	Integer
dsSchemaClassCount	Y	Integer
dsSchemaPrefixCount	Y	Integer
dsServiceName	N	Object(DS-DN)
highestCommittedUSN	N	LargeInteger
isGlobalCatalogReady	N	Boolean
isSynchronized	N	Boolean
ldapServiceName	N	String(Unicode)
namingContexts	N	Object(DS-DN)
netlogon	Y	String(Octet)
pendingPropagations	Y	Object(DS-DN)
rootDomainNamingContext	N	Object(DS-DN)
schemaNamingContext	N	Object(DS-DN)
serverName	N	Object(DS-DN)
subschemaSubentry	N	Object(DS-DN)
supportedCapabilities	N	String(Object-Identifier)
supportedControl	N	String(Object-Identifier)
supportedLDAPPolicies	N	String(Unicode)
supportedLDAPVersion	N	Integer
supportedSASLMechanisms	N	String(Unicode)
domainControllerFunctionality	N	Integer
domainFunctionality	N	Integer
forestFunctionality	N	Integer
msDS-ReplAllInboundNeighbors	Y	String(Unicode)*
msDS-ReplAllOutboundNeighbors	Y	String(Unicode)*
msDS-ReplConnectionFailures	Y	String(Unicode)*

Attribute name	Operational?	LDAP syntax
msDS-ReplLinkFailures	Y	String(Unicode)*
msDS-ReplPendingOps	Y	String(Unicode)*
msDS-ReplQueueStatistics	Y	String(Unicode)*
msDS-TopQuotaUsage	Y	String(Unicode)**
supportedConfigurableSettings	Y	String(Unicode)
supportedExtension	Y	String(Object-Identifier)
validFSMOs	Y	Object(DS-DN)
dsaVersionString	Y	String(Unicode)
msDS-PortLDAP	Y	Integer
msDS-PortSSL	Y	Integer
msDS-PrincipalName	Y	String(Unicode)
serviceAccountInfo	Y	String(Unicode)
spnRegistrationResult	Y	Integer
tokenGroups	Y	String (SID)
usnAtRifm	Y	LargeInteger

* These values contain XML. At the client's request, the server will return the value as binary data in String(Octet) syntax instead.

** This value contains XML.

3.1.1.3.2.1 configurationNamingContext

Returns the DN of the root of the config NC on this DC.

3.1.1.3.2.2 currentTime

Returns the current system time on the DC, as expressed as a string in the Generalized Time format defined by ASN.1 (see [ISO-8601] and [ITUX680], as well as the documentation for the LDAP String(Generalized-Time) syntax in 3.1.1.2.2.2).

3.1.1.3.2.3 defaultNamingContext

Returns the DN of the root of the default NC of this DC. For AD LDS, the defaultNamingContext attribute does not exist if a value has not been set for the msDS-DefaultNamingContext attribute of the DC's nTDSDSA object.

3.1.1.3.2.4 dNSHostName

Returns the DNS address of this DC.

3.1.1.3.2.5 dsSchemaAttrCount

Returns an integer specifying the total number of attributes that are defined in the schema.

3.1.1.3.2.6 dsSchemaClassCount

Returns an integer specifying the total number of classes that are defined in the schema.

3.1.1.3.2.7 dsSchemaPrefixCount

Returns the number of entries in the DC's prefix table: the field prefixTable of the variable dc specified in [MS-DRSR] section 5.30.

3.1.1.3.2.8 dsServiceName

Returns the DN of the nTDSDSA object for the DC.

3.1.1.3.2.9 highestCommittedUSN

Returns the USN of this DC. In terms of the state model of section 3.1.1.1 this is *dc.usn*.

3.1.1.3.2.10 isGlobalCatalogReady

Returns a Boolean value indicating if this DC is a global catalog that has completed at least one synchronization of its global catalog data with its replication partners. Returns true if it meets this criteria or false if either the global catalog on this DC has not completed synchronization or this DC does not host a global catalog.

3.1.1.3.2.11 isSynchronized

Returns a Boolean value indicating if the DC has completed at least one synchronization with its replication partners. Returns either true, if it is synchronized, or false, if it is not.

3.1.1.3.2.12 ldapServiceName

Returns the LDAP service name for the LDAP server on the DC. The format of the value is <**DNS name of the forest root domain**>:<**Kerberos principal name**>, where **Kerberos principal name** is a string representation of the Kerberos **principal** name for the DC's computer object, as defined in [RFC1964] section 2.1.1.

3.1.1.3.2.13 namingContexts

Returns a multivalued set of DNs. For each NC-replica n hosted on this DC, this attribute contains the DN of the root of n.

3.1.1.3.2.14 netlogon

LDAP searches that request this rootDSE attribute get resolved as LDAP ping operations, as specified in section 6.3. Active Directory supports LDAP searches for this attribute via both UDP and TCP/IP. See section 3.1.1.3.1.11 for details on LDAP over UDP.

3.1.1.3.2.15 pendingPropagations

Returns a set of DNs of objects whose nTSecurityDescriptor attribute (that is, the object's security descriptor) has been updated but the inheritable portion of the update has not yet been propagated to descendant objects (see Security Descriptor Requirements, section 6.1.3). An object is included in the set only if the update that caused the temporary inconsistency in the object's nTSecurityDescriptor was performed on the LDAP connection that is reading the pendingPropagations rootDSE attribute.

3.1.1.3.2.16 rootDomainNamingContext

Returns the DN of the root domain NC for this DC's forest.

3.1.1.3.2.17 schemaNamingContext

Returns the DN of the root of the schema NC on this DC.

3.1.1.3.2.18 serverName

Returns the DN of the **server object**, contained in the config NC, that represents this DC.

3.1.1.3.2.19 subschemaSubentry

Returns the DN for the location of the subSchema object where the classes and attributes in the directory are defined. The subSchema object pointed to by this attribute contains a read-only copy of the schema described in the format specified in section 3.1.1.3.1.1.1

3.1.1.3.2.20 supportedCapabilities

Returns a multivalued set of OIDs specifying the capabilities supported by this DC. The definition of each OID is explained in section 3.1.1.3.4.3.

3.1.1.3.2.21 supportedControl

Returns a multivalued set of OIDs specifying the LDAP controls supported by this DC. The definition of each OID is explained in section 3.1.1.3.4.1

3.1.1.3.2.22 supportedLDAPPolicies

Returns a multivalued set of strings specifying the LDAP administrative query policies supported by this DC. The policy strings returned are listed in section 3.1.1.3.4.6.

3.1.1.3.2.23 supportedLDAPVersion

Returns a set of integers specifying the versions of LDAP supported by this DC. Active Directory supports version 2 and version 3 of LDAP, so it returns {2,3} as an LDAP multivalue.

3.1.1.3.2.24 supportedSASLMechanisms

Returns a multivalued set of strings specifying the security mechanisms supported for SASL negotiation (see [RFC2222], [RFC2829], and [RFC2831]). The definition of each value is explained in section 3.1.1.3.4.5.

3.1.1.3.2.25 domainControllerFunctionality

Returns an integer indicating the functional level of the DC. This value is populated from the msDS-Behavior-Version attribute on the nTDSDSA object that represents the DC (section 6.1.4.2).

Value	Identifier
0	DS_BEHAVIOR_WIN2000
2	DS_BEHAVIOR_WIN2003
3	DS_BEHAVIOR_WIN2008
4	DS_BEHAVIOR_WIN2008R2
5	DS_BEHAVIOR_WIN2012
6	DS_BEHAVIOR_WIN2012R2

3.1.1.3.2.26 domainFunctionality

Returns an integer indicating the functional level of the domain. This value is populated from the msDS-Behavior-Version attribute on the domain NC root object and the crossRef object that represents the domain (section 6.1.4.3).

Value	Identifier
0	DS_BEHAVIOR_WIN2000
1	DS_BEHAVIOR_WIN2003_WITH_MIXED_DOMAINS
2	DS_BEHAVIOR_WIN2003
3	DS_BEHAVIOR_WIN2008
4	DS_BEHAVIOR_WIN2008R2
5	DS_BEHAVIOR_WIN2012
6	DS_BEHAVIOR_WIN2012R2

3.1.1.3.2.27 forestFunctionality

Returns an integer indicating the functional level of the forest. This value is populated from the msDS-Behavior-Version attribute on the crossRefContainer object (section 6.1.4.4).

Value	Identifier
0	DS_BEHAVIOR_WIN2000
1	DS_BEHAVIOR_WIN2003_WITH_MIXED_DOMAINS
2	DS_BEHAVIOR_WIN2003
3	DS_BEHAVIOR_WIN2008
4	DS_BEHAVIOR_WIN2008R2

Value	Identifier
5	DS_BEHAVIOR_WIN2012
6	DS_BEHAVIOR_WIN2012R2

3.1.1.3.2.28 msDS-ReplAllInboundNeighbors, msDS-ReplConnectionFailures, msDS-ReplLinkFailures, and msDS-ReplPendingOps

Returns alternate representations of the structures returned by IDL_DRSGetReplInfo() (see [MS-DRSR] section 4.1.13), either as binary data structures or as XML. The relationship between each of these rootDSE attributes and the IDL_DRSGetReplInfo data is shown in the following table.

rootDSE attribute name	Equivalent DS_REPL_ INFO_TYPE	XML structure	Binary structure
msDS-ReplAllInboundNeighbors	DS_REPL_INFO_NEIGHBORS	DS_REPL_NEIGHBORW	DS_REPL_NEIGHBORW_BLOB
msDS-ReplConnectionFailures	DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES	DS_REPL_KCC_DSA_FAILUREW	DS_REPL_KCC_DSA_FAILUREW_BLOB
msDS-ReplLinkFailures	DS_REPL_INFO_KCC_DSA_LINK_FAILURES	DS_REPL_KCC_DSA_FAILUREW	DS_REPL_KCC_DSA_FAILUREW_BLOB
msDS-ReplPendingOps	DS_REPL_INFO_PENDING_OPS	DS_REPL_OPW	DS_REPL_OPW_BLOB

For each rootDSE attribute named in the first column, the information returned is exactly the same information that is returned by a call to IDL_DRSGetReplInfo, specifying the value in the second column as the DRS_MSG_GETREPLINFO_REQ_V1.InfoType or DRS_MSG_GETREPLINFO_REQ_V2.InfoType. See [MS-DRSR] for the definition of these, as well as for the definition of the following constants and structures used in the table above:

DS_REPL_INFO_NEIGHBORS

DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES

DS_REPL_INFO_KCC_DSA_LINK_FAILURES

DS_REPL_INFO_PENDING_OPS

DS_REPL_NEIGHBORW

DS_REPL_KCC_DSA_FAILUREW

DS_REPL_OPW

The remaining structures in the table above are documented in section 2.2.

Without any attribute qualifier, the data is returned as XML. The parent element of the XML is the name of the structure contained in the "XML structure" column in the table, and the child element names and order in the XML exactly follow the names of the fields in that structure as well. The meaning of each child element is the same as the meaning of the corresponding field in the structure. Values of integer types are represented as decimal strings. Values of FILETIME type are

represented as XML dateTime values in **Coordinated Universal Time (UTC)**, for example, "04-07T18:39:09Z", as defined in [XMLSCHEMA2/2]. Values of GUID fields are represented as **GUIDStrings**.

If the ";binary" attribute qualifier is specified when the attribute is requested, the value of this attribute is returned as binary data, specifically, the structure contained in the "Binary structure" column. In this representation, fields that would contain strings are represented as integer offsets (relative to the beginning of the binary data) to a null-terminated UTF-16 encoded string embedded in the returned binary data.

3.1.1.3.2.29 msDS-ReplAllOutboundNeighbors

This attribute is equivalent to msDS-ReplAllInboundNeighbors, except that it returns representations of each value of the repsTo abstract attribute for each NC-replica (for example, outbound replication), while msDS-ReplAllInboundNeighbors returns representations of each value of the repsFrom abstract attribute (for example, inbound replication). Like msDS-ReplAllInboundNeighbors, the server will return the data in either XML or binary form, depending on the presence of the ";binary" attribute qualifier, and uses the DS_REPL_NEIGHBOR and DS_REPL_NEIGHBORW_BLOB structures for its XML and binary representations, respectively.

3.1.1.3.2.30 msDS-ReplQueueStatistics

Reading the msDS-ReplQueueStatistics attribute returns replication queue statistics.

Like the other ms-dsRepl* rootDSE attributes, the server returns either XML or binary data, depending on the presence of the ";binary" attribute qualifier. For XML, it returns the following representation:

<DS_REPL_QUEUE_STATISTICSW>

<ftimeCurrentOpStarted> **ftimeCurrentOpStartedValue** </ftimeCurrentOpStarted>

<cNumPendingOps> **cNumPendingOpsValue** </cNumPendingOps>

<ftimeOldestSync> **ftimeOldestSyncValue** </ftimeOldestSync>

<ftimeOldestAdd> **ftimeOldestAddValue** </ftimeOldestAdd>

<ftimeOldestMod> **ftimeOldestModValue** </ftimeOldestMod>

<ftimeOldestDel> **ftimeOldestDelValue** </ftimeOldestDel>

<ftimeOldestUpdRefs> **ftimeOldestUpdRefsValue** </ftimeOldestUpdRefs>

</DS_REPL_QUEUE_STATISTICSW>

The structure returned by this attribute for the binary representation is DS_REPL_QUEUE_STATISTICSW_BLOB (section 2.2.5).

The information returned by reading this attribute is derived from the field replicationQueue of the variable dc specified in [MS-DRSR] section 5.30. dc.replicationQueue is used to serialize IDL_DRSReplicaSync, IDL_DRSReplicaAdd, IDL_DRSReplicaModify, IDL_DRSReplicaDel, and IDL_DRSUpdateRefs request processing [MS-DRSR] on the DC. msDS-ReplQueueStatistics returns the following information about the current state of this queue:

- **ftimeCurrentOpStartedValue** is the date and time that the current IDL_DRSReplicaSync, IDL_DRSReplicaAdd, IDL_DRSReplicaModify, IDL_DRSReplicaDel, or IDL_DRSUpdateRefs request left the queue and started running.
- **cNumPendingOpsValue** is the number of queued IDL_DRSReplicaSync, IDL_DRSReplicaAdd, IDL_DRSReplicaModify, IDL_DRSReplicaDel, or IDL_DRSUpdateRefs requests.
- **ftimeOldestSyncValue** is the date and time that the oldest queued IDL_DRSReplicaSync request entered the queue.
- **ftimeOldestAddValue** is the date and time that the oldest queued IDL_DRSReplicaAdd request entered the queue.
- **ftimeOldestModValue** is the date and time that the oldest queued IDL_DRSReplicaModify request entered the queue.
- **ftimeOldestDelValue** is the date and time that the oldest queued IDL_DRSReplicaDel request entered the queue.
- **ftimeOldestUpdRefsValue** is the date and time that the oldest queued IDL_DRSUpdateRefs request entered the queue.

cNumPendingOpsValue is an integer represented as a decimal string. The remaining values are represented as XML dateTime values in UTC, defined in [XMLSCHEMA2/2].

If a designated request does not exist, the corresponding portion of the msDS-ReplQueueStatistics response contains a zero filetime in the binary format, and the XML dateTime value "1601-01-01T00:00:00Z" in XML format. For instance, if there is no IDL_DRSUpdateRefs request in the replication queue, the msDS-ReplQueueStatistics XML response includes:

<ftimeOldestUpdRefs>1601-01-01T00:00:00Z</ftimeOldestUpdRefs>

3.1.1.3.2.31 msDS-TopQuotaUsage

Returns a multivalued set of strings specifying the top 10 quota users in all NC-replicas on this DC. The format of each value is as follows, where quota usage is measured in number of objects:

<MS_DS_TOP_QUOTA_USAGE>

<partitionDN> **DN of NC-replica** </partitionDN>

<ownerSID> **Security Identifier (SID) of quota user** </ownerSID>

<quotaUsed> **Amount of quota used by this quota user** </quotaUsed>

<tombstoneCount> **Number of tombstoned objects owned by this quota user** </tombstoneCount>

<liveCount> **Number of live (non-deleted) objects owned by this quota user** </liveCount >

</MS_DS_TOP_QUOTA_USAGE>

A client qualifies the attribute description for this attribute in an LDAP query with a "range qualifier" to specify a different range of quota users to return other than the top 10. The DC responds to this by returning the quota usage for the requested range of quota users. Following are examples of range qualifiers and what would be returned:

- An attribute specification of the form *msDS-TopQuotaUsage;Range=0-** will return the complete list of quota usage.
- An attribute specification of the form *msDS-TopQuotaUsage;Range=1-9* will return the second highest through the 10th highest quota usage.
- An attribute specification of the form *msDS-TopQuotaUsage;Range=2-2* will return the third highest quota usage.

The caller must have the RIGHT_DS_READ_PROPERTY access right on the Quotas container (see section 6.1.1.4.3). If the caller does not have this access right, the search operation will succeed but no results will be returned.

3.1.1.3.2.32 supportedConfigurableSettings

Returns a multivalued set of strings specifying the configurable settings supported by this DC. The setting strings returned are listed in section 3.1.1.3.4.7.

3.1.1.3.2.33 supportedExtension

Returns a multivalued set of OIDs specifying the extended LDAP operations that the DC supports. The definition of each OID is explained in section 3.1.1.3.4.2.

3.1.1.3.2.34 validFSMOs

Returns a set of DN s of objects representing the FSMO roles owned by this DC. Each object identifies a distinct FSMO role.

The valid types of FSMO role, and the object used to represent an instance of that type in the **validFSMOs** attribute, are as follows:

- Schema Master FSMO Role - the root of the schema NC
- Domain Naming FSMO Role - the **Partitions container** in the config NC
- Infrastructure Master FSMO Role - the Infrastructure container in a domain NC
- **Primary Domain Controller (PDC)** Emulator FSMO Role - the root of a domain NC
- RID Master FSMO Role - the RID Manager object of a domain NC, which is the object referenced by the rIDManagerReference attribute on the root of the domain NC

Because an AD LDS forest does not contain domain NCs, it does not contain instances of the Infrastructure Master, PDC Emulator, and RID Master FSMO roles, and the corresponding objects will not be present in the **validFSMOs** attribute of any DC running AD LDS.

A server indicates that it owns a given FSMO role F only if IsEffectiveRoleOwner(RoleObject(**nc**, **e**)) returns true, where the procedures IsEffectiveRoleOwner and RoleObject are defined in section 3.1.1.5.1.8. The parameters **nc** and **e** are defined as follows for each FSMO Role F:

- Schema Master FSMO
  - **nc**: Schema NC
  - **e**: SchemaMasterRole
- Domain Naming FSMO

- **nc**: Config NC
- **e**: DomainNamingMasterRole

- Infrastructure Master FSMO
  - **nc**: Default NC (AD DS)
  - **e**: InfrastructureMasterRole
- RID Master FSMO
  - **nc**: Default NC (AD DS)
  - **e**: RidAllocationMasterRole
- PDC Emulator FSMO
  - **nc**: Default NC (AD DS)
  - **e**: PdcEmulationMasterRole

3.1.1.3.2.35 dsaVersionString

Returns a string indicating the version of Active Directory running on the DC. For instance, when running Windows Server 2008 operating system Beta 2, the Active Directory version string is "6.0.5384.32 (winmain_beta2.060727-1500)".

This rootDSE attribute is readable by Domain Administrators (section 6.1.1.6.5) and Enterprise Administrators (section 6.1.1.6.10) only.

3.1.1.3.2.36 msDS-PortLDAP

Returns the integer TCP/UDP port number on which the DC is listening for LDAP requests. For AD DS, this always equals 389. For AD LDS, the port is configurable.

Note This rootDSE attribute is different from the schema attribute of the same name, msDS-PortLDAP.

3.1.1.3.2.37 msDS-PortSSL

Returns the integer TCP/UDP port number on which the DC is listening for TLS/SSL-protected LDAP requests. For AD DS, this always equals 636. For AD LDS, the port is configurable.

Note This rootDSE attribute is different from the schema attribute of the same name, msDS-PortSSL.

3.1.1.3.2.38 msDS-PrincipalName

Returns a string name of the security principal that has authenticated on the LDAP connection. If the client authenticated as a Windows security principal, the string contains either (1) the **NetBIOS domain name**, followed by a backslash ("\"), followed by the sAMAccountName of the security principal, or (2) the SID of the security principal, in SDDL SID string format ([MS-DTYP] section 2.4.2.1). If the client authenticated as an AD LDS security principal, the string contains the DN of the security principal. If the connection is not authenticated (only possible if the fLDAPBlockAnonOps heuristic in the dSHeuristics attribute is false; see section 6.1.1.2.4.1.2), the string is "NT AUTHORITY\ANONYMOUS LOGON".

Note This **rootDSE** attribute is different from the schema attribute of the same name, msDS-PrincipalName.

3.1.1.3.2.39 serviceAccountInfo

Returns a set of strings, each string containing a name-value pair encoded as **name**=**value**.

The serviceAccountInfo attribute contains information outside the state model. The possible name-value pairs are as follows:

replAuthenticationMode: The value is the value of the msDS-ReplAuthenticationMode attribute on the root of the config NC, or "**1**" if that attribute is not set. See section 6.1.1.1.2 for the effects of the msDS-ReplAuthenticationMode attribute.

accountType: If the service account is a domain account account, the value is "**domain**". Otherwise the service account is a local account, and the value is "**local**".

systemAccount: If the service account is a system account (meaning it has one of the SIDs SID "S-1-5-20" and "S-1-5-18") the value is "**true**"; otherwise the value is "**false**".

domainType: If the DC is running on a computer that is part of an Active Directory domain (always the case for an AD DS DC), the value is "**domainWithKerb**". If the DC is running on a computer that is part of an NT (pre–Active Directory) domain, the value is "**domainNoKerb**". Otherwise the DC is running on a computer that is not part of a domain, and the value is "**nonMember**".

serviceAcccountName: If the value of **replAuthenticationMode** is "**0**", the value is the SAM name of the DC's service account. Otherwise this name-value pair is not present.

machineDomainName: If **domainType** is "**domainWithKerb**" or "**domainNoKerb**" the value is the **NetBIOS** name of the domain. Otherwise the value is the NetBIOS name of the computer.

3.1.1.3.2.40 spnRegistrationResult

When running as AD DS, this value is 0. When running as AD LDS, if the DC was unable to register its **service principal names (SPNs)** ([MS-DRSR] section 2.2.2), this attribute returns the Windows error code associated with the failure. Otherwise, it returns zero.

Note When running as AD DS on Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, or Windows Server 2012 R2 operating system, this value is 21.

3.1.1.3.2.41 tokenGroups

Returns the SIDs contained in the **security context** as which the client has authenticated the LDAP connection. Refer to section 5.1.3 for details on LDAP Authorization. Refer to section 3.1.1.4.5.19 for details on the algorithm used to compute this attribute.

3.1.1.3.2.42 usnAtRifm

This attribute contains information outside the state model. If the DC is an RODC and was installed using the Install From Media feature, reading the usnAtRifm attribute returns the value of dc.usn (section 3.1.1.1.9) that was present in the Active Directory database on the installation media.

3.1.1.3.3 rootDSE Modify Operations

This section specifies the modifiable attributes on the rootDSE of Windows 2000 operating system, Windows Server 2003 operating system, Active Directory Application Mode (ADAM), Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system DCs (both AD DS and AD LDS).

rootDSE modify operations are used to trigger behaviors on a specific DC. For example, one such operation causes the DC to acquire the Schema Master FSMO. All of these rootDSE attributes are write-only; an LDAP request to read will be treated as if the attribute does not exist.

The following table specifies the set of modifiable rootDSE attributes included in each Windows or ADAM version.

Attribute name	Windows 2000	Windows 2000 operating system Service Pack 1 (SP 1)	Windows Server 2003	Windows Server 2003 SP 3	ADAM RTW	ADAM SP1	Windows Server 2008 AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 AD DS	Windows Server 2012 R2 AD LDS
becomeDomainMaster	X	X	X	X	X	X	X	X	X	X	X	X	X	X
becomeInfrastructureMaster	X	X	X	X			X		X		X		X	
becomePdc	X	X	X	X			X		X		X		X	
becomePdcWithCheckPoint	X	X	X	X			X		X		X		X	
becomeRidMaster	X	X	X	X			X		X		X		X	
becomeSchemaMaster	X	X	X	X	X	X	X	X	X	X	X	X	X	X
checkPhantoms	X	X	X	X			X		X		X		X	
doGarbageCollection	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Attribute name	Windows 2000	Windows 2000 operating system Service Pack 1 (SP 1)	Windows Server 2003	Windows Server 2003 SP 3	ADAMRTW	ADAMSP1	Windows Server 2008 ADDS	Windows Server 2008 ADLDS	Windows Server 2008 R2 ADDS	Windows Server 2008 R2 ADLDS	Windows Server 2012 ADDS	Windows Server 2012 ADLDS	Windows Server 2012 R2 ADDS	Windows Server 2012 R2 ADLDS
dumpDatabase	X	X	X	X	X	X	X	X	X	X	X	X	X	X
fixupInheritance	X	X	X	X	X	X	X	X	X	X	X	X	X	X
invalidateRidPool	X	X	X	X			X		X		X		X	
recalcHierarchy	X	X	X	X			X		X		X		X	
schemaUpdateNow	X	X	X	X	X	X	X	X	X	X	X	X	X	X
schemaUpgradeInProgress			X	X			X		X		X		X	
removeLingeringObject		X	X	X	X	X	X	X	X	X	X	X	X	X
doLinkCleanup			X	X	X	X	X	X	X	X	X	X	X	X
doOnlineDefrag			X	X	X	X	X	X	X	X	X	X	X	X
replicateSingleObject			X	X	X	X	X	X	X	X	X	X	X	X
updateCachedMemberships			X	X			X		X		X		X	
doGarbageCollectionPhantomsNow				X		X	X	X	X	X	X	X	X	X
invalidateGCConnec							X	X	X	X	X	X	X	X

Attribute name	Windows 2000	Windows 2000 operating system Service Pack 1 (SP 1)	Windows Server 2003	Windows Server 2003 SP 3	ADAMRTW	ADAMSP1	Windows Server 2008 ADDS	Windows Server 2008 ADLDS	Windows Server 2008 R2 ADDS	Windows Server 2008 R2 ADLDS	Windows Server 2012 ADDS	Windows Server 2012 ADLDS	Windows Server 2012 R2 ADDS	Windows Server 2012 R2 ADLDS
tion														
renewServerCertificate							X	X	X	X	X	X	X	X
rODCPurgeAccount							X		X		X		X	
runSamUpgradeTasks							X		X		X		X	
sqmRunOnce								X		X		X		X
runProtectAdminGroupsTask									X		X		X	
disableOptionalFeature									X	X	X	X	X	X
enableOptionalFeature									X	X	X	X	X	X
dumpReferences											X	X	X	X
dumpLinks													X	X
schemaUpdateIndicesNow													X	X
null													X	X

Each of these operations is executed by performing an LDAP Modify operation with a NULL DN for the object to be modified (indicating the rootDSE) and specifying the name of the operation as the attribute to be modified. In [RFC2849] terminology the rootDSE attribute to be modified is the "AttributeDescription" of the "mod-spec" associated with the "change-modify" record. In many of

the cases, the type of the modify (add or replace) and the values specified do not matter and are ignored. Whether the type and values matter, and what the client specifies if they do matter, will be indicated for each operation in the following sections. Examples are given as LDAP Data Interchange Format (LDIF) samples, described in [RFC2849]. In Windows, LDIF is implemented by the ldifde.exe command-line tool.

To perform many of these operations, the caller must be authenticated as a user that has a particular control access right or privilege; or, in some cases, as a user that is a member of a particular group. In each section that follows, the rights, privileges, or group membership, if any, that are required of the caller to perform a specific operation are specified. If the caller does not have the required rights, privileges, or group membership, the server returns the error *insufficientAccessRights* / *ERROR_ACCESS_DENIED*.

3.1.1.3.3.1 becomeDomainMaster

Performing this operation causes the DC to request a transfer of the Domain Naming FSMO to itself, per the FSMO role transfer procedure documented in [MS-DRSR] section 4.1.10.4.3 (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP_FSMO_REQ_ROLE). The requester must have the "Change-Domain-Master" control access right on the Partitions container in the config NC for this to succeed. This operation cannot be performed on an RODC; an RODC will return error *unwillingToPerform* / *ERROR_INVALID_PARAMETER*. The LDAP operation returns success after the transfer of the Domain Naming FSMO has completed successfully.

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter. The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: becomeDomainMaster
becomeDomainMaster: 1
-
```

3.1.1.3.3.2 becomeInfrastructureMaster

Performing this operation causes the DC to request a transfer of the Infrastructure Master FSMO to itself, per the FSMO role transfer procedure documented in [MS-DRSR] section 4.1.10.4.3 (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP_FSMO_REQ_ROLE). The requester must have the "Change-Infrastructure-Master" control access right on the Infrastructure container in the domain NC replica. This operation cannot be performed on an RODC; an RODC will return the error *unwillingToPerform* / *ERROR_INVALID_PARAMETER*. The LDAP operation returns success after the transfer of the Infrastructure Master FSMO has completed successfully.

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter. The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: becomeInfrastructureMaster
becomeInfrastructureMaster: 1
-
```

3.1.1.3.3.3 becomePdc

Performing this operation causes the DC to request a transfer of the PDC Emulator FSMO to itself, per the FSMO role transfer procedure documented in [MS-DRSR] section 4.1.10.4.3 (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP_FSMO_REQ_PDC). The requester must have the "Change-PDC" control access right on the root of the domain NC replica. This operation cannot be performed on an RODC; an RODC will return the error *unwillingToPerform* / *ERROR_INVALID_PARAMETER*. The LDAP operation returns success after the transfer of the PDC Emulator FSMO has completed successfully.

Prior to transferring the PDC FSMO to the DC, if the domain is in **mixed mode**, the DC attempts to synchronize with the DC that is currently the Owner of the PDC FSMO in such a way as to avoid causing a full synchronization by BDCs running Windows NT 4.0 operating system (see section 3.1.1.7). However, the FSMO role transfer will be performed even if this synchronization is unsuccessful.

In order to perform this operation, the requester must provide the domain's SID, in binary format (defined in [MS-DTYP] section 2.4.2), as the value of the modify operation. In LDIF, this would be performed as follows. Note that LDIF requires that binary values be base-64 encoded.

```
dn:
changetype: modify
add: becomePdc
becomePdc:: base-64 encoding of the domain SID in binary
-
```

3.1.1.3.3.4 becomePdcWithCheckPoint

This operation is the same as becomePdc except for the following. Prior to transferring the PDC FSMO, if the domain is in mixed mode, the DC attempts to synchronize with the DC that is the current the owner of the PDC FSMO. becomePdc transfers the PDC FSMO role even if this synchronization is unsuccessful, while becomePdcWithCheckPoint does not.

3.1.1.3.3.5 becomeRidMaster

Performing this operation causes the DC to request a transfer of the RID Master FSMO to itself, per the FSMO role transfer procedure documented in [MS-DRSR] section 4.1.10.4.3 (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP_FSMO_RID_REQ_ROLE). The requester must have the "Change-RID-Master" control access right on the RID Manager object, which is the object referenced by the rIDManagerReference attribute located on the root of the domain NC. The requester must also have **read permission** on the previously mentioned rIDManagerReference attribute. This operation cannot be performed on an RODC; an RODC returns the error *unwillingToPerform* / *ERROR_INVALID_PARAMETER*. The LDAP operation returns success after the transfer of the RID Master FSMO has completed successfully.

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter. The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: becomeRidMaster
becomeRidMaster: 1
-
```

3.1.1.3.3.6 becomeSchemaMaster

Performing this operation causes the DC to request a transfer of the Schema Master FSMO to itself, per the FSMO role transfer procedure documented in [MS-DRSR] section 4.1.10.4.3 (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP_FSMO_REQ_ROLE). The requester must have the "Change-Schema-Master" control access right on the root of the schema NC replica. This operation cannot be performed on an RODC; an RODC will return the error *unwillingToPerform* / *ERROR_INVALID_PARAMETER*. The LDAP operation returns success after the transfer of the Schema Master FSMO has completed successfully.

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter. The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: becomeSchemaMaster
becomeSchemaMaster: 1
-
```

3.1.1.3.3.7 checkPhantoms

This operation requests that the reference update task (see section 3.1.1.6.2) be immediately performed on the DC. During the operation, if the referential integrity on any of the objects is found to be incorrect and it cannot be corrected, then the operation returns an error and does not process any of the remaining objects. This task runs periodically; on a correctly functioning DC, there is no need to run it explicitly. The requester must have the "DS-Check-Stale-Phantoms" control access right on the nTDSDSA object for the DC.

No action is taken if the Recycle Bin optional feature is not enabled and the operation is performed against a DC that does not own the Infrastructure Master FSMO.

No action is taken if the operation is performed against a DC that is a global catalog.

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter.

The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: checkPhantoms
checkPhantoms: 1
-
```

3.1.1.3.3.8 doGarbageCollection

This operation requests that garbage collection be immediately performed on the DC. Tombstones and recycled-objects are subject to the requirement that they must be kept for at least the tombstone lifetime (see 3.1.1.6.2), but they may be kept longer. Deleted-objects are subject to the requirement that they must be kept for at least the deleted-object lifetime. Garbage collection

identifies tombstones and recycled-objects that have been kept for at least the tombstone lifetime and removes them. Additionally, garbage collection identifies deleted-objects that have been kept for at least the deleted-object lifetime and transforms them to recycled-objects. On a correctly functioning DC, there should not be a need to manually trigger garbage collection via this operation. The requester must have the "Do-Garbage-Collection" control access right on the DC's **DSA object**.

This operation is triggered by setting the doGarbageCollection attribute to "1".

The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: doGarbageCollection
doGarbageCollection: 1
-
```

3.1.1.3.3.9 dumpDatabase

This operation is triggered by setting the attribute to a space-separated list of attributes. The requester must be a member of the BUILTIN\Administrators group (section 6.1.1.4.12.2).

The following shows an LDIF sample that performs this operation for the description and sn attributes.

```
dn:
changetype: modify
add: dumpDatabase
dumpDatabase: description sn
-
```

The effects of dumpDatabase are outside the state model. An update of dumpDatabase causes the contents of the DC's database to be written to a text file on the DC's disk. All the attributes specified in the dumpDatabase value are included in the dump, except that certain security-sensitive attributes are omitted from the dump even if requested. The dump may include attributes that were not explicitly requested.

3.1.1.3.3.10 fixupInheritance

The fixupInheritance attribute permits administrative tools to request that the DC recompute inherited security permissions on objects to ensure that they conform to the security descriptor requirements (see section 6.1.3), in case the current state of the permissions on the object is erroneous. This operation is not necessary on a correctly functioning DC. The requester must have the "Recalculate-Security-Inheritance" control access right on the nTDSDSA object for the DC. The LDAP Operation returning success means the system accepts the request to perform security-descriptor propagation.

This operation is triggered by setting the fixupInheritance attribute to "1".

The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: fixupInheritance
fixupInheritance: 1
```

```
-
```

In Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, setting the fixupInheritance attribute to the special values "forceupdate" and "downgrade" has effects outside the state model.

In Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2, the fixupInheritance attribute can trigger security-descriptor propagation under an object, specified using an identifier outside the state model, rather than throughout the directory. This is performed by setting the fixupInheritance attribute to the string "dnt:" followed by an implementation-specific identifier representing the object. Consider the following example.

```
dn:
changetype: modify
add: fixupInheritance
fixupInheritance: dnt:54758
-
```

3.1.1.3.3.11 invalidateRidPool

This operation causes the DC to discard its current pool of RIDs, used for allocating security principals in the directory. The DC requests a fresh pool of RIDs from the DC that owns the RID Master FSMO, per the procedure documented in [MS-DRSR] section 4.1.10.4.3 (PerformExtendedOpRequestMsg, ulExtendedOp = EXOP_FSMO_REQ_RID_ALLOC). The LDAP Operations returns success when the RID pool has been invalidated. Obtaining a fresh pool of RIDs from the DC that owns the RID Master FSMO is an asynchronous operation.

The requester must have the "Change-RID-Master" control access right on the RID Manager object, which is the object referenced by the rIDManagerReference attribute located on the root of the domain NC. The requester must also have read permission on the previously mentioned rIDManagerReference attribute. This operation cannot be performed on an RODC; an RODC returns the error *unwillingToPerform / ERROR_INVALID_PARAMETER*.

In order to perform this operation, the requester provides the domain's SID, in binary format (defined in [MS-DTYP] section 2.4.2), as the value of the modify operation.

The following shows an LDIF sample that performs this operation. LDIF requires that binary values, like the domain SID, be base-64 encoded.

```
dn:
changetype: modify
add: invalidateRidPool
invalidateRidPool:: base-64 encoding of the binary-format domain SID
-
```

3.1.1.3.3.12 recalcHierarchy

The requester must have the "Recalculate-Hierarchy" control access right on the nTDSDSA object for the DC. The type of modification can be add or replace, and the values specified in the LDAP Modify operation do not matter. The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: recalcHierarchy
recalcHierarchy: 1
-
```

The effects of recalcHierarchy are outside the state model. An update of recalcHierarchy causes the hierarchy table used to support the MAPI address book to be recalculated immediately.

3.1.1.3.3.13 schemaUpdateNow

The requester must have the "Update-Schema-Cache" control access right on the nTDSDSA object for the DC or on the root of the **schema NC**. After the completion of this operation, the subschema exposed by the server reflects the current state of the schema as defined by the attributeSchema and classSchema objects in the schema NC.

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter. The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: schemaUpdateNow
schemaUpdateNow: 1
-
```

The other effects of schemaUpdateNow are outside the state model. An update of schemaUpdateNow causes the in-memory cache of the schema to be recalculated from the copy of the schema stored in the schema NC.

3.1.1.3.3.14 schemaUpgradeInProgress

This operation causes the **fschemaUpgradeInProgress** field of LDAPConnection instances in dc.LDAPConnections ([MS-DRSR] section 5.115) to be set. schemaUpgradeInProgress causes the DC to skip certain constraint validations when adding, updating, or removing directory objects. The skipped constraint validations are documented in the applicable constraint sections of this document. The requester must have the "Change-Schema-Master" control access right on the root of the schema NC-replica.

On the Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, when schemaUpgradeInProgress is set to 1 the **fschemaUpgradeInProgress** field is set to true on the LDAPConnection instance in dc.ldapConnections that corresponds to the LDAP connection on which the schemaUpgradeInProgress operation was performed. On these operating systems, when schemaUpgradeInProgress is set to zero the **fschemaUpgradeInProgress** field is set to false on the LDAPConnection instance in dc.ldapConnections that corresponds to the LDAP connection on which the schemaUpgradeInProgress operation was performed.

On the Windows Server 2003 operating system and Windows Server 2003 R2 operating system, when schemaUpgradeInProgress is set to 1 the **fschemaUpgradeInProgress** field is set to true in every LDAPConnection instance in dc.ldapConnections. On these operating systems, when schemaUpgradeInProgress is set to zero the **fschemaUpgradeInProgess** field is set to false on every LDAPConnection instance in dc.ldapConnections.

The type of modification can be add or replace. The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: schemaUpgradeInProgress
schemaUpgradeInProgress: 1
-
```

schemaUpgradeInProgress operation permits modifications to be performed that would otherwise violate constraints had schemaUpgradeInProgress not been set.

3.1.1.3.3.15 removeLingeringObject

This operation causes the DC to expunge a lingering object. A DC that was offline for longer than the value of the tombstone lifetime can contain objects that have been deleted on other DCs and for which tombstones no longer exist. The result is that when that DC is brought back online, any such objects can continue to exist in its NC replica even though the objects should have been deleted. Such objects are known as lingering objects.

Expunge is specified in section 3.1.1.1.6. Lingering object expunge can be performed on an object in a read-only NC. For more details on the lingering object expunge process, see IDL_DRSReplicaVerifyObjects and IDL_DRSGetObjectExistence in [MS-DRSR] sections 4.1.24 and 4.1.12.

The requester must have the "DS-Replication-Synchronize" control access right on the root of the NC replica that contains the lingering object.

The value specified for this operation contains (1) the DN of the DSA object of a DC holding a writable replica of the NC containing the lingering object, and (2) the DN of the lingering object. These are encoded in the value string as two DNs separated by a colon: "DSA Object DN:Lingering Object DN". Each DN specified may be either an [RFC2253]-style DN or one of the alternative DN formats described in section 3.1.1.3.1.2.4. If the value is not in the specified format, the server rejects the request with the error *operationsError* / *ERROR_DS_OBJ_NOT_FOUND*.

The DC performing the modify request first verifies that the lingering object specified in the request does not exist on the DC specified in the request. If this verification fails for any reason, the request returns the error *operationsError* / *ERROR_DS_GENERIC_ERROR*. If the verification succeeds, the DC expunges the lingering object specified in the request and then returns success.

The following shows an LDIF sample that performs this operation. The sample requests that the lingering object whose DN is "CN=TestObject, CN=Users, DC=Fabrikam, DC=com" be removed, and specifies that the server whose nTDSDSA object is "CN=NTDS Settings,CN=TESTDC-01,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=Fabrikam,DC=com" be used to verify the nonexistence of the lingering object.

```
dn:
changetype: modify
replace: removeLingeringObject
removeLingeringObject: CN=NTDS Settings,
CN=TESTDC-01,CN=Servers,CN=Default-First-Site-Name,
CN=Sites,CN=Configuration,DC=Fabrikam,DC=com:CN=TestObject,
CN=Users, DC=Fabrikam, DC=com
-
```

3.1.1.3.3.16 doLinkCleanup

This operation causes the DC to immediately begin performing any delayed link processing necessary to satisfy the requirements of delayed link processing, as specified in section 3.1.1.1.16. This processing runs automatically as needed to satisfy those requirements; on a correctly functioning DC, there is no need to explicitly request such processing. The requester must have the "Do-Garbage-Collection" control access right on the nTDSDSA object for the DC.

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter. The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: doLinkCleanup
doLinkCleanup: 1
-
```

3.1.1.3.3.17 doOnlineDefrag

This operation is triggered by setting the doOnlineDefrag attribute to a non-negative integer. The requester must have the "Do-Garbage-Collection" control access right on the nTDSDSA object for the DC. The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
replace: doOnlineDefrag
doOnlineDefrag: 60
-
```

The effects of doOnlineDefrag are outside the state model. An update of doOnlineDefrag causes an online defragmentation of the DC's directory database. If the doOnlineDefrag value is positive, it starts the defragmentation task, which runs until complete or until the specified number of seconds have elapsed. If the doOnlineDefrag value is zero, the defragmentation task is stopped if it is running.

3.1.1.3.3.18 replicateSingleObject

This operation causes the DC to request replication of a single object, specified in the modify request, from a source DC to the DC processing the request. The requester must have the "DS-Replication-Synchronize" control access right on the root of the NC that contains the object to be replicated.

The type of modification specified in the LDAP modify operation does not matter; however the value specified does matter. The value specified for the replicateSingleObject attribute in the modify request contains (1) the DN of the DSA object of the source DC, and (2) the DN of the object to be replicated. These are encoded in the value string as two DNs separated by a colon: "**DSA Object DN**:**Object To Be Replicated DN**". Each DN specified may be either an RFC 2253-style DN or one of the alternative DN formats described in section 3.1.1.3.1.2.4. If the value is not in the specified format, the server rejects the request with the error *operationsError* / *ERROR_DS_OBJ_NOT_FOUND*.

If the DC is an RODC, an additional colon may be added to the end of the value string, followed by the literal string "SECRETS_ONLY". The presence of this additional parameter indicates that the RODC should request replication of the object's secret attributes instead of the other attributes.

When this flag is specified, the "DS-Replication-Synchronize" control access right is not checked. Instead, the requester must possess the "Read-Only-Replication-Secret-Synchronization" control access right on the root of the NC containing the object whose secret attributes are to be replicated.

This operation is a synchronous operation. The LDAP response is returned by the server after the replication of the object from the source DC to the DC processing the request has completed. However, if the object to be replicated does not exist on the source DC, or if the object to be replicated has been deleted on the source DC, or if the object to be replicated does not have a parent object on the DC processing the request, an error is returned and the replication is not performed.

The following shows an LDIF sample that performs the replicateSingleObject operation. This sample requests that the object whose DN is "CN=TestObject, CN=Users, DC=Fabrikam, DC=com" be replicated from the DC whose nTDSDSA object is "CN=NTDS Settings,CN=TESTDC-01,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=Fabrikam,DC=com".

```
dn:
changetype: modify
replace: replicateSingleObject
replicateSingleObject: CN=NTDS Settings,
CN=TESTDC-01,CN=Servers,CN=Default-First-Site-Name,
CN=Sites,CN=Configuration,DC=Fabrikam,DC=com:CN=TestObject,
CN=Users, DC=Fabrikam, DC=com
-
```

3.1.1.3.3.19 updateCachedMemberships

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter. The requester must have the "Refresh-Group-Cache" control access right on the nTDSDSA object for the DC.

The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: updateCachedMemberships
updateCachedMemberships: 1
-
```

The effects of updateCachedMemberships are outside the state model. An update of updateCachedMemberships causes the DC to refresh its cache of universal group memberships from a GC server.

3.1.1.3.3.20 doGarbageCollectionPhantomsNow

This operation is triggered by setting the doGarbageCollectionPhantomsNow attribute to "1". The requester must have the "Do-Garbage-Collection" control access right on the nTDSDSA object for the DC.

The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: doGarbageCollectionPhantomsNow
```

```
doGarbageCollectionPhantomsNow: 1
-
```

The effects of doGarbageCollectionPhantomsNow are outside the state model. An update of doGarbageCollectionPhantomsNow causes a garbage-collector to run that reclaims storage used to implement referential integrity.

3.1.1.3.3.21 invalidateGCConnection

The type of modification to the invalidateGCConnection attribute and the values specified in the LDAP Modify operation do not matter. The requester must be a member of either the BUILTIN\Administrators group (section 6.1.1.4.12.2) or the BUILTIN\Server Operators group (section 6.1.1.4.12.18).

The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: invalidateGCConnection
invalidateGCConnection: 1
-
```

The effects of invalidateGCConnection are outside the state model. This operation causes the DC to rediscover the GC server that it uses in its implementation of referential integrity (section 3.1.1.1.6).

3.1.1.3.3.22 renewServerCertificate

The persistent state of a DC does not include the certificates that are necessary to authenticate the DC when a client makes an LDAPS (LDAP over SSL/TLS) connection. A DC obtains the certificates it needs by querying the operating system for them at startup. This operation provides a means for the requester to request that the DC repeat the query to the operating system for the certificates—for example, if the available certificates have changed since startup. The requester must have the "Reload-SSL-Certificate" control access right on the nTDSDSA object for the DC.

An LDAP Modify of the renewServerCertificate attribute causes the DC to query the operating system for certificates. When the operation returns, the DC has performed the query and the certificates it found are available for use in LDAPS connections.

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter.

The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: renewServerCertificate
renewServerCertificate: 1
-
```

3.1.1.3.3.23 rODCPurgeAccount

An LDAP Modify of the rODCPurgeAccount attribute causes the RODC to purge cached secret attributes of a specified security principal. The requester must have the "Read-Only-Replication-Secret-Synchronization" control access right on the root of the default NC. The Modify request must be directed to an RODC that hosts an NC replica that contains the specified RODC object. If the RODC to which the operation is directed does not host such an NC, then the error *operationsError / ERROR_DS_CANT_FIND_EXPECTED_NC* is returned. If the operation is sent to a DC that is not an RODC, then the error *operationsError / ERROR_DS_GENERIC_ERROR* is returned.

The value specified for the rODCPurgeAccount attribute in the LDAP modify request must be the **DN** of the object whose secret attributes are to be purged. The DN specified may be either an [RFC2253]-style DN or one of the alternative DN formats described in section 3.1.1.3.1.2.4. If the value is not in the specified format or the object does not exist, the server rejects the request with the error *operationsError / ERROR_DS_OBJ_NOT_FOUND*. The server returns success upon successfully purging the secret attributes of the specified security principal.

The following shows an LDIF sample that performs this operation. This sample purges the cached secret attributes of the user whose DN is "CN=TestUser, CN=Users, DC=Fabrikam, DC=com" from the RODC to which this operation is sent.

```
dn:
changetype: modify
replace: rODCPurgeAccount
rODCPurgeAccount: CN=TestUser, CN=Users, DC=Fabrikam, DC=com
-
```

3.1.1.3.3.24 runSamUpgradeTasks

An LDAP Modify of the runSamUpgradeTasks attribute causes the default groups and memberships (as specified in [MS-SAMR] section 3.1.4.2) to be created in the domain if they are not already created. This operation is useful in a domain with different versions of domain controllers where the default groups and memberships are not yet created.

If a partial set of these modifications has already been performed in the domain through this task, the Modify operation of this attribute MUST cause the rest of the operations to be performed. If all such modifications have already been performed, the Modify operation of this attribute MUST NOT make any changes in the domain.

The requester MUST be a member of the "Domain Admins" group in the domain to perform this operation.

The DC, on receiving this request, MUST verify that the otherWellKnownObjects attribute on the object "CN=Server, CN=System, DC=<domain>" on the DC with the PDC role contains "B:32:6ACDD74F3F314AE396F62BBE6B2DB961:X", where <domain> is the domain NC DN, and X is the DN of the nTDSDSA object of the DC receiving the request. If this condition is not satisfied, the LDAP Modify returns *operationsError / ERROR_DS_GENERIC_ERROR*.

If these conditions are satisfied, the default groups and memberships (as specified in [MS-SAMR] section 3.1.4.2) are created in the domain.

The type of modification and values specified in the LDAP Modify operation do not matter. The following shows an LDIF sample that performs this operation. This sample triggers the default groups and memberships created on the target domain.

```
dn:
changetype: modify
add: runSamUpgradeTasks
runSamUpgradeTasks: 1
-
```

3.1.1.3.3.25 sqmRunOnce

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter. The requester must have the SE_DEBUG_PRIVILEGE.

The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: sqmRunOnce
sqmRunOnce: 1
-
```

The effects of sqmRunOnce are outside the state model. An update of sqmRunOnce causes the DC to report statistical data on the types and numbers of operations that the DC has performed using an implementation-defined reporting mechanism.

3.1.1.3.3.26 runProtectAdminGroupsTask

The type of modification made to the runProtectAdminGroupsTask attribute and the values specified in the LDAP Modify operation have no significance. If the DC is the PDC **FSMO role owner**, an LDAP Modify of the runProtectAdminGroupsTask attribute causes the DC to run the AdminSDHolder protection operation (section 3.1.1.6.1). Otherwise, the Modify request does not have any effect. The requester must have the "Run-Protect-Admin-Groups-Task" control access right on the domain root of the DC. The LDAP server returns success after the AdminSDHolder operation has completed.

An LDIF sample that performs this operation is shown as follows.

```
dn:
changetype: modify
add: runProtectAdminGroupsTask
runProtectAdminGroupsTask: 1
-
```

3.1.1.3.3.27 disableOptionalFeature

This operation requests that an optional feature (as described in section 3.1.1.9) be disabled for some scope. The requester must have the correct "Manage-Optional-Features" control access on the object representing the scope.

This operation is triggered by setting the disableOptionalFeature attribute to a value that contains the DN of the object that represents the scope, followed by the colon (:) character, followed by the GUID of the optional feature to be enabled, expressed as a GUIDString.

If the server does not recognize the GUID as identifying a known feature, the server will return the error *operationsError* / *ERROR_INVALID_PARAMETER*.

If the DN represents an existing object but the object does not represent a scope, the server will return the error *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED*. If the DN does not represent an existing object, the server will return the error *operationsError* / *ERROR_INVALID_PARAMETER*.

If the feature is not marked as being valid for the specified scope, the server will return the error *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED*.

If the specified scope is forest-wide, and this operation is not performed against the DC that holds the Domain Naming Master role, the server will return the error *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED*.

If the feature is not marked as being able to be disabled, the server will return the error *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED*.

If the specified optional feature is not already enabled in the specified scope, the server will return the error *noSuchAttribute* / *ERROR_DS_CANT_REM_MISSING_ATT_VAL*.

The LDAP server returns success when the specified optional feature has been successfully disabled.

An LDIF sample that performs this operation is shown as follows.

```
dn:
changetype: modify
add: disableOptionalFeature
disableOptionalFeature: cn=Partitions,cn=Configuration,DC=Contoso,DC=Com:766DDCD8-ACD0-445E-
F3B9-A7F9B6744F2A
-
```

3.1.1.3.3.28 enableOptionalFeature

This operation requests that an optional feature (as described in section 3.1.1.9) be enabled for some scope. The requester must have the "Manage-Optional-Features" control access right on the object representing the scope.

This operation is triggered by setting the enableOptionalFeature attribute to a value that contains the DN of the object that represents the scope, followed by the ':' character, followed by the GUID of the optional feature to be enabled, expressed as a GUIDString.

If the server does not recognize the GUID as identifying a known feature, the server will return the error *operationsError* / *ERROR_INVALID_PARAMETER*.

If the DN represents an existing object but the object does not represent a scope, the server will return the error *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED*. If the DN does not represent an existing object, the server will return the error *operationsError* / *ERROR_INVALID_PARAMETER*.

If the feature is not marked as being valid for the specified scope, the server will return the error *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED*.

If the specified scope is forest-wide and this operation is not performed against the DC that holds the Domain Naming Master role, the server will return the error *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED*.

If the specified optional feature is already enabled in the specified scope, the server will return the error *attributeOrValueExists* / *ERROR_DS_ATT_VAL_ALREADY_EXISTS*.

The LDAP server returns success when the specified optional feature has been successfully enabled.

An LDIF sample that performs this operation is shown as follows.

```
dn:
changetype: modify
add: enableOptionalFeature
enableOptionalFeature: cn=Partitions,cn=Configuration,DC=Contoso,DC=Com:766DDCD8-ACD0-445E-
F3B9-A7F9B6744F2A
-
```

3.1.1.3.3.29 dumpReferences

This operation is triggered by setting the attribute to the DN of an existing object. The requester must be a member of the BUILTIN\Administrators group (section 6.1.1.4.12.2).

The following shows an LDIF sample that performs this operation for the object whose DN is "CN=TestObject,CN=Users,DC=Fabrikam,DC=com":

```
changetype: modify
add: dumpReferences
dumpReferences: CN=TestObject,CN=Users,DC=Fabrikam,DC=com
-
```

The effects of dumpReferences are outside the state model. An update of dumpReferences causes all attributes that reference the given DN and all objects containing those attributes to be written to a text file on the DC's disk.

3.1.1.3.3.30 dumpLinks

The type of modification made to the dumpLinks attribute and the values specified in the LDAP Modify operation have no significance. The requester must be a member of the BUILTIN\Administrators group (section 6.1.1.4.12.2).

The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: dumpLinks
dumpLinks: 1
-
```

The effects of dumpLinks are outside the state model. An update of dumpLinks causes the portion of the contents of the DC's database relating to link values to be written to a text file on the DC's disk.

3.1.1.3.3.31 schemaUpdateIndicesNow

The requester must have the "Update-Schema-Cache" control access right on the nTDSDSA object for the DC or on the root of the schema NC. This operation is supported only when the fDisableAutoIndexingOnSchemaUpdate heuristic (section 6.1.1.2.4.1.2) is "2". If fDisableAutoIndexingOnSchemaUpdate is not "2", the operation fails with an error. After the completion of this operation, the subschema exposed by the server reflects the current state of the schema as defined by the attributeSchema and classSchema objects in the schema NC.

The type of modification can be add or replace, and the values specified in the LDAP modify operation do not matter. The following shows an LDIF sample that performs this operation.

```
dn:
changetype: modify
add: schemaUpdateIndicesNow
schemaUpdateIndicesNow: 1
-
```

The other effects of schemaUpdateIndicesNow are outside the state model. An update of schemaUpdateIndicesNow causes the DC to verify its data indices. See section 3.1.1.3.4.1.32.1 for a note on indices.

3.1.1.3.3.32 null

The type of modification made to the null attribute and the values specified in the LDAP Modify operation have no significance. Writing to this attribute has no effect.

3.1.1.3.4 LDAP Extensions

This section describes the extensions to LDAP that are supported by Active Directory DCs in Windows 2000 operating system, Windows Server 2003 operating system, Active Directory Application Mode (ADAM), Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system. These extensions are:

- LDAP extended controls
- LDAP extended operations
- LDAP capabilities
- Matching rules
- SASL mechanisms
- Policies
- Configurable settings
- IP Deny list

3.1.1.3.4.1 LDAP Extended Controls

LDAP extended controls are an extensibility mechanism in version 3 of LDAP, as discussed in [RFC2251] section 4.1.12. The following sections describe the LDAP extended controls implemented by DCs in Windows 2000 operating system, Windows Server 2003 operating system, Active Directory Application Mode (ADAM), Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system (both AD DS and AD LDS).

The LDAP extended controls supported by a DC are exposed as OIDs in the supportedControl attribute of the rootDSE. Each OID corresponds to a human-readable name, as shown in the following table.

Extended control name	OID
LDAP_PAGED_RESULT_OID_STRING	1.2.840.113556.1.4.319
LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID	1.2.840.113556.1.4.521
LDAP_SERVER_DIRSYNC_OID	1.2.840.113556.1.4.841
LDAP_SERVER_DOMAIN_SCOPE_OID	1.2.840.113556.1.4.1339
LDAP_SERVER_EXTENDED_DN_OID	1.2.840.113556.1.4.529
LDAP_SERVER_GET_STATS_OID	1.2.840.113556.1.4.970
LDAP_SERVER_LAZY_COMMIT_OID	1.2.840.113556.1.4.619
LDAP_SERVER_PERMISSIVE_MODIFY_OID	1.2.840.113556.1.4.1413
LDAP_SERVER_NOTIFICATION_OID	1.2.840.113556.1.4.528
LDAP_SERVER_RESP_SORT_OID	1.2.840.113556.1.4.474
LDAP_SERVER_SD_FLAGS_OID	1.2.840.113556.1.4.801
LDAP_SERVER_SEARCH_OPTIONS_OID	1.2.840.113556.1.4.1340
LDAP_SERVER_SORT_OID	1.2.840.113556.1.4.473
LDAP_SERVER_SHOW_DELETED_OID	1.2.840.113556.1.4.417
LDAP_SERVER_TREE_DELETE_OID	1.2.840.113556.1.4.805
LDAP_SERVER_VERIFY_NAME_OID	1.2.840.113556.1.4.1338
LDAP_CONTROL_VLVREQUEST	2.16.840.1.113730.3.4.9
LDAP_CONTROL_VLVRESPONSE	2.16.840.1.113730.3.4.10
LDAP_SERVER_ASQ_OID	1.2.840.113556.1.4.1504
LDAP_SERVER_QUOTA_CONTROL_OID	1.2.840.113556.1.4.1852
LDAP_SERVER_RANGE_OPTION_OID	1.2.840.113556.1.4.802
LDAP_SERVER_SHUTDOWN_NOTIFY_OID	1.2.840.113556.1.4.1907
LDAP_SERVER_FORCE_UPDATE_OID	1.2.840.113556.1.4.1974
LDAP_SERVER_RANGE_RETRIEVAL_NOERR_OID	1.2.840.113556.1.4.1948
LDAP_SERVER_RODC_DCPROMO_OID	1.2.840.113556.1.4.1341
LDAP_SERVER_DN_INPUT_OID	1.2.840.113556.1.4.2026
LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID	1.2.840.113556.1.4.2065
LDAP_SERVER_SHOW_RECYCLED_OID	1.2.840.113556.1.4.2064
LDAP_SERVER_POLICY_HINTS_DEPRECATED_OID	1.2.840.113556.1.4.2066
LDAP_SERVER_DIRSYNC_EX_OID	1.2.840.113556.1.4.2090

Extended control name	OID
LDAP_SERVER_UPDATE_STATS_OID	1.2.840.113556.1.4.2205
LDAP_SERVER_TREE_DELETE_EX_OID	1.2.840.113556.1.4.2204
LDAP_SERVER_SEARCH_HINTS_OID	1.2.840.113556.1.4.2206
LDAP_SERVER_EXPECTED_ENTRY_COUNT_OID	1.2.840.113556.1.4.2211
LDAP_SERVER_POLICY_HINTS_OID	1.2.840.113556.1.4.2239
LDAP_SERVER_SET_OWNER_OID	1.2.840.113556.1.4.2255
LDAP_SERVER_BYPASS_QUOTA_OID	1.2.840.113556.1.4.2256

The following table lists the set of LDAP extended controls supported in each Windows Server operating system or ADAM version.

Extended control name	Windows 2000	Windows Server 2003	Windows Server 2003 operating system with Service Pack 1 (SP1)	ADAM RTW	ADAM SP1	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
LDAP_PAGED_RESULT_OID_STRING	X	X	X	X	X	X	X	X	X
LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_DIRSYNC_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_DOMAIN_SCOPE_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_EXTENDED_DN_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_GET_STATS_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_LAZY_COMMIT_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_PERMISSIVE_MODIFY_OID	X	X	X	X	X	X	X	X	X

Extended control name	Windows 2000	Windows Server 2003	Windows Server 2003 operating system with Service Pack 1 (SP1)	ADAM RTW	ADAM SP1	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
LDAP_SERVER_NOTIFICATION_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_RANGE_OPTION_OID*	X	X	X	X	X	X	X	X	X
LDAP_SERVER_RESP_SORT_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_SD_FLAGS_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_SEARCH_OPTIONS_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_SORT_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_SHOW_DELETED_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_TREE_DELETE_OID	X	X	X	X	X	X	X	X	X
LDAP_SERVER_VERIFY_NAME_OID	X	X	X	X	X	X	X	X	X
LDAP_CONTROL_VLVREQUEST		X	X	X	X	X	X	X	X
LDAP_CONTROL_VLVRESPONSE		X	X	X	X	X	X	X	X
LDAP_SERVER_ASQ_OID		X	X	X	X	X	X	X	X
LDAP_SERVER_QUOTA_CONTROL_OID		X	X	X	X	X	X	X	X
LDAP_SERVER_SHUTDOWN_NOTIFY_OID**			X		X	X	X	X	X
LDAP_SERVER_FORCE_UPDATE_OID						X	X	X	X

Extended control name	Windows 2000	Windows Server 2003	Windows Server 2003 operating system with Service Pack 1 (SP1)	ADAM RTW	ADAM SP1	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
LDAP_SERVER_RANGE_RETRIEVAL_NOERR_OID					X	X	X	X	X
LDAP_SERVER_RODC_DCPROMO_OID						X	X	X	X
LDAP_SERVER_DN_INPUT_OID						X	X	X	X
LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID							X	X	X
LDAP_SERVER_SHOW_RECYCLED_OID							X	X	X
LDAP_SERVER_POLICY_HINTS_DEPRECATED_OID							X	X	X
LDAP_SERVER_DIRSYNC_EX_OID								X	X
LDAP_SERVER_UPDATE_STATS_OID								X	X
LDAP_SERVER_TREE_DELETE_EX_OID								X	X
LDAP_SERVER_SEARCH_HINTS_OID								X	X
LDAP_SERVER_EXPECTED_ENTRY_COUNT_OID								X	X
LDAP_SERVER_POLICY_HINTS_OID								X	X
LDAP_SERVER_SET_OWNER_OID									X
LDAP_SERVER_BYPASS_QUOTA_OID									X

* This OID does not identify an LDAP extended control. Its presence in the supportedControl attribute indicates that the DC is capable of range retrieval (see section 3.1.1.3.1.3.3) of LDAP multivalued attributes. However, its absence does not indicate lack of support for range retrieval. This OID is not present in the supportedControl attribute of Windows 2000 DCs, but those DCs do support range retrieval.

** Although exposed on the supportedControl attribute of Windows Server 2003 SP1, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 DCs, this control is only functional on DCs running the Small Business Server version of that operating system.

A client sends a control to the DC by attaching a Control structure (defined in [RFC2251] section 4.1.12) to an LDAP operation. The client sets the controlType field to the control's OID and the controlValue field as specified in the discussion for the control that follows. If the controlValue field contains data that is not in conformance with the specification of the control, including the case where the controlValue field contains data and the specification of the control states that the controlValue field is omitted, then if the control is marked critical the server returns the error *unavailableCriticalExtension* / *ERROR_INVALID_PARAMETER*. If the controlValue field is incorrect but the control is not marked critical, the server ignores the control.

A control sent by the client to a DC is known as a request control. In some cases, the server includes a corresponding Control structure attached to the response for the LDAP operation. These controls, known as response controls, are discussed below in conjunction with the request control that causes that response control to be returned.

A brief description of each LDAP control is given in the following table. Additionally, each control is discussed in more detail in the sections that follow. References to ASN.1 and BER encoding in the following section are references to [ITUX680] and [ITUX690], respectively.

Extended control name	Description
LDAP_PAGED_RESULT_OID_STRING	Splits the results of an LDAP search across multiple result sets.
LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID	Used with an LDAP Modify DN operation to move an object from one domain to another domain.
LDAP_SERVER_DIRSYNC_OID	Used with an LDAP search operation to retrieve the changes made to objects since a previous LDAP_SERVER_DIRSYNC_OID search was performed.
LDAP_SERVER_DOMAIN_SCOPE_OID	Instructs the DC not to generate LDAP continuation references in response to a search operation.
LDAP_SERVER_EXTENDED_DN_OID	Used to request than an LDAP search operation return DNs in an extended format containing the values of the objectGUID and objectSid attributes.
LDAP_SERVER_GET_STATS_OID	Used with an LDAP search request to instruct the DC to return statistical data related to how the search was performed.
LDAP_SERVER_LAZY_COMMIT_OID	Instructs the DC that it MAY sacrifice durability guarantees on updates to improve performance.
LDAP_SERVER_PERMISSIVE_MODIFY_OID	Instructs the DC that an LDAP modify should succeed even if it attempts to add a value already present on the attribute or remove a value not present on the

Extended control name	Description
	attribute.
LDAP_SERVER_NOTIFICATION_OID	Used with an LDAP search operation to register the client to be notified when changes are made to an object in the directory.
LDAP_SERVER_SD_FLAGS_OID	Instructs the DC which portions of a Windows security descriptor to retrieve during an LDAP search operation.
LDAP_SERVER_SEARCH_OPTIONS_OID	Used to pass flags to the DC to control search behaviors; specifically, to prevent LDAP continuation references from being generated and to search all NC replicas that are subordinate to the search base, even if the search base is not instantiated on the DC.
LDAP_SERVER_SORT_OID and LDAP_SERVER_RESP_SORT_OID	Request and response controls, respectively, for instructing the DC to sort the search results.
LDAP_SERVER_SHOW_DELETED_OID	Used with an LDAP operation to specify that tombstones and deleted-objects are visible to the operation.
LDAP_SERVER_TREE_DELETE_OID	Used with an LDAP delete operation to cause the server to recursively delete the entire subtree of objects located under the object specified in the search request (including the specified object).
LDAP_SERVER_VERIFY_NAME_OID	Permits the client to specify which GC the DC should use when processing an add or modify request to verify the existence of any objects pointed to by DN attribute values.
LDAP_CONTROL_VLVREQUEST and LDAP_CONTROL_VLVRESPONSE	Request and response control, respectively, used with an LDAP search operation to retrieve a "sliding window" subset of the objects that satisfy the search request.
LDAP_SERVER_ASQ_OID	Used to specify that an LDAP search operation should not be performed against the object specified as the base in the search, but rather against the set of objects named by a specified attribute of Object(DS-DN) syntax on the base object.
LDAP_SERVER_QUOTA_CONTROL_OID	Used with an LDAP search operation to retrieve the quota of a user.
LDAP_SERVER_RANGE_OPTION_OID	Indicates that the server is capable of range retrieval (see section 3.1.1.3.1.3.3).
LDAP_SERVER_SHUTDOWN_NOTIFY_OID	Used with an LDAP search operation to cause the client to be notified when the DC is shutting down.
LDAP_SERVER_FORCE_UPDATE_OID	When attached to an LDAP update operation, causes the DC to perform the update even if that update would not affect the state of the DC.
LDAP_SERVER_RANGE_RETRIEVAL_NOERR_OID	Instructs the DC that, when performing a search

Extended control name	Description
	using range retrieval (see section 3.1.1.3.1.3.3) on an attribute whose values are forward link values or back link values and the value of low is greater than or equal to the number of values in the attribute, no error should be returned.
LDAP_SERVER_RODC_DCPROMO_OID	This control is used as part of the process of promoting a computer to be an RODC.
LDAP_SERVER_DN_INPUT_OID	This control is used to specify the DN of an object during an LDAP operation. Currently this control is used only while retrieving the constructed attribute msDS-IsUserCachableAtRodc (see section 3.1.1.3.4.1.24).
LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID	Used with an LDAP search operation to specify that link attributes that refer to deleted-objects are visible to the search operation. If used in conjunction with LDAP_SERVER_SHOW_DELETED_OID or LDAP_SERVER_SHOW_RECYCLED_OID, link attributes that are stored on deleted-objects are also visible to the search operation. This applies both to the search filter and the set of attributes returned by the search operation.
LDAP_SERVER_SHOW_RECYCLED_OID	Used with an LDAP operation to specify that tombstones, deleted-objects, and recycled-objects are visible to the operation.
LDAP_SERVER_POLICY_HINTS_DEPRECATED_OID	The LDAP_SERVER_POLICY_HINTS_DEPRECATED_OID control has the exact semantics and behaviors as LDAP_SERVER_POLICY_HINTS_OID (section 3.1.1.3.4.1.27); this control MAY be used by clients when the server does not support LDAP_SERVER_POLICY_HINTS_OID. Clients SHOULD use LDAP_SERVER_POLICY_HINTS_OID when it is supported by the server.
LDAP_SERVER_DIRSYNC_EX_OID	Used with an LDAP search operation to retrieve the changes made to objects since a previous LDAP_SERVER_DIRSYNC_EX_OID search was performed.
LDAP_SERVER_UPDATE_STATS_OID	The LDAP_SERVER_UPDATE_STATS_OID control indicates that the requester requires statistics from the DC.
LDAP_SERVER_TREE_DELETE_EX_OID	Used with an LDAP delete operation to cause the server to recursively delete the entire subtree of objects, up to a specified number of objects, located under the object specified in the search request (including the specified object).
LDAP_SERVER_SEARCH_HINTS_OID	Provides hints to the DC during LDAP search operations.
LDAP_SERVER_EXPECTED_ENTRY_COUNT_OID	Monitors the result of an LDAP search operation and

Extended control name	Description
	potentially modifies the return code.
LDAP_SERVER_POLICY_HINTS_OID	Used with an LDAP operation to enforce password history policies during password set.
LDAP_SERVER_SET_OWNER_OID	Used with an LDAP add operation to set the owner of the object to a SID other than that of the requester.
LDAP_SERVER_BYPASS_QUOTA_OID	Used with an LDAP add operation to specify that quota limits do not apply for the add operation.

3.1.1.3.4.1.1 LDAP_PAGED_RESULT_OID_STRING

This control, which is used as both a request control and a response control, is documented in [RFC2696].

DCs limit the number of objects that can be returned in a single search operation to the value specified by the MaxPageSize policy defined in section 3.1.1.3.4.6. The use of the LDAP_PAGED_RESULT_OID_STRING control permits clients to perform searches that return more objects than this limit by splitting the search into multiple searches, each of which returns no more objects than this limit.

3.1.1.3.4.1.2 LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID

The LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID control is used with an LDAP Modify DN operation to instruct the DC to move an object from one domain to another (see the Modify DN operation in section 3.1.1.5). This control is used by the client when moving an object from one domain to another. The client sends the LDAP Modify DN operation to which this control is attached to a DC in the domain containing the object to be moved. If the client does not specify the LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID control in the LDAP Modify DN request, then the server interprets the update as an intradomain Modify DN operation.

When operating as AD LDS, a DC rejects this control with the error *operationsError* / *<unrestricted>*.

When sending this control to the DC, the **controlValue** field is set to a UTF-8 string containing the **fully qualified domain name** of a DC in the domain to which the object is to be moved. The string is not BER-encoded. Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.3 LDAP_SERVER_DIRSYNC_OID

The LDAP_SERVER_DIRSYNC_OID control is used with an LDAP search operation to retrieve the changes made to objects since a previous search with an LDAP_SERVER_DIRSYNC_OID control was performed. The LDAP_SERVER_DIRSYNC_OID control can only be used to monitor for changes across an entire NC replica, not a subtree within an NC replica.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
    Flags       INTEGER
    MaxBytes    INTEGER
    Cookie      OCTET STRING
```

}

The **Flags** value has the following format presented in big-endian byte order. X denotes unused bits set to 0 by the client and ignored by the server.

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

The **Flags** value is a combination of zero or more bit flags from the following table, and is used to specify additional behaviors for the LDAP_SERVER_DIRSYNC_OID control.

Bit flag name and value	Description
LDAP_DIRSYNC_OBJECT_SECURITY (OS) 0x00000001	Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system: If this flag is present, the client can only view objects and attributes that are otherwise accessible to the client. If this flag is not present, the server checks if the client has access rights to read the changes in the NC. Windows 2000 operating system: Not supported.
LDAP_DIRSYNC_ANCESTORS_FIRST_ORDER (AFO) 0x00000800	The server returns parent objects before child objects.
LDAP_DIRSYNC_PUBLIC_DATA_ONLY (PDO) 0x00002000	Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2: This flag can optionally be passed to the DC, but it has no effect. Windows 2000: Not supported.
LDAP_DIRSYNC_INCREMENTAL_VALUES (IV) 0x80000000	Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2: If this flag is not present, all of the values, up to a server-specified limit, in a multivalued attribute are returned when any value changes. If this flag is present, only the changed values are returned, provided the attribute is a forward link value. Windows 2000: Not supported.

MaxBytes specifies the maximum number of bytes to return in the reply message.

The minimum value for **MaxBytes** is 0x100000. When a lower value is specified, the value is ignored and the maximum number of bytes in the reply message is 0x100000.

The maximum value for **MaxBytes** is determined by the size, in bytes, of a response with the maximum number of objects that can be returned in a single search as specified by the

MaxPageSize policy, section 3.1.1.3.4.6. When a higher value is specified, the value is ignored and the maximum number of bytes in the reply message is the size, in bytes, of a response with the MaxPageSize number of objects.

Cookie is an opaque value that was returned by the DC on a previous search request that included the LDAP_SERVER_DIRSYNC_OID control. The contents of **Cookie** are defined by the server and cannot be interpreted by the client. A search request with the LDAP_SERVER_DIRSYNC_OID control attached will return the changes made to objects since the point in time when the previous search request, which returned the value of **Cookie** that is being used in the current search request, took place. If there was no previous LDAP_SERVER_DIRSYNC_OID search request, **Cookie** is NULL, in which case the search will return all objects that satisfy the search request, along with a value of **Cookie** to use for the next LDAP_SERVER_DIRSYNC_OID search request.

If the base of the search is not the root of an NC, and the LDAP_DIRSYNC_OBJECT_SECURITY bit in the **Flags** field is not set, the server will return the error *insufficientAccessRights* / *ERROR_DS_DRA_ACCESS_DENIED*. If the LDAP_DIRSYNC_OBJECT_SECURITY bit in the **Flags** field is set, the server will return the error *unwillingToPerform* / *<unrestricted>*. If the search scope is not subtree scope, the server will treat the search as if subtree scope was specified.

Any valid LDAP search filter can be specified.

Any attributes can be requested in the search. Only those objects for which these attributes have been created or modified since the time represented by **Cookie** will be considered for inclusion in the search.

If the list of requested attributes contains an asterisk (*) plus some attribute, then the asterisk is ignored. That is, the list is effectively equal to the list with only the attributes explicitly requested.

The search results MUST always contain the objectGUID and instanceType attributes of each object, even if those attributes were not specified in the search request.

When the server receives a search request with the LDAP_SERVER_DIRSYNC_OID control attached to it, it includes a response control in the search response. The **controlType** field of the returned Control structure is set to the OID of the LDAP_SERVER_DIRSYNC_OID control, and the controlValue is the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
    MoreResults     INTEGER
    unused          INTEGER
    CookieServer    OCTET STRING
}
```

The structure of the controlValue in the response control is the same as the structure of the controlValue in the request control, but the fields are interpreted differently. **MoreResults** is nonzero if there are more changes to retrieve, **unused** is not used, and **CookieServer** is the value to be used for **Cookie** in the next LDAP_SERVER_DIRSYNC_OID control sent in a search request to the server.

Further details about how this control is processed are described in the pseudocode for the ProcessDirSyncSearchRequest procedure in [MS-DRSR] section 5.114.3.

3.1.1.3.4.1.4 LDAP_SERVER_DOMAIN_SCOPE_OID

The LDAP_SERVER_DOMAIN_SCOPE_OID control is used to instruct the DC not to generate any LDAP continuation references when performing an LDAP operation. The effect of this is to limit any

search using it to the single NC replica in which the object that serves as the root of the search is located.

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.5 LDAP_SERVER_EXTENDED_DN_OID

The LDAP_SERVER_EXTENDED_DN_OID control is used with an LDAP search request to cause the DC to return extended DNs. The extended form of an object's DN includes a string representation of the object's objectGUID attribute; for objects that have an objectSid attribute, the extended form also includes a string representation of that attribute. The DC uses this extended DN for all DNs in the LDAP search response. Attributes with Object(OR-Name) syntax are not affected by this control, because in those cases, the DC always uses the DN form as specified in [RFC2253].

The extended DN format is as follows:

<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, and **dn** is the object's RFC 2253 DN. For objects that do not have an objectSid attribute, the format is instead as follows:

<GUID=**guid_value**>;**dn**

When sending this control to a Windows 2000 operating system DC, the controlValue field is omitted. When sending this control to a Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, or Windows Server 2012 R2 operating system DC, the controlValue field is either omitted or is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    Flag    INTEGER
}
```

If the controlValue field is omitted, the value of Flag is treated as 0.

If the value of **Flag** is 0, the DC returns the values of the objectGUID and objectSid attributes as a hexadecimal representation of their binary format.

If the value of **Flag** is 1, the DC returns the GUID in dashed-string format ([RFC4122] section 3) and the SID in SDDL SID string format ([MS-DTYP] section 2.4.2.1). The returned SDDL SID string begins with "S-".

If the value of Flag is neither 0 nor 1, then it does not conform with the specification of this control and the server behaves as described in section 3.1.1.3.4.1.

For example, setting **Flag** to 0 (or omitting the controlValue field) might return the following extended DN:

<GUID=b3d4bfbd3c45ee4298e27b4a698a61b8>;<SID=010500000000000515000000061eb5b8c50ef705befda808bf4010000>;CN=Administrator, CN=Users,DC=Fabrikam,DC=com

While setting **Flag** to 1 would return the same object's extended DN in the following form:

<GUID=bdbfd4b3-453c-42ee-98e2-7b4a698a61b8>;<SID=S-1-5-21-2354834273-1534127952-2340477679-500>;CN=Administrator, CN=Users,DC=Fabrikam,DC=com

Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.6 LDAP_SERVER_GET_STATS_OID

The LDAP_SERVER_GET_STATS_OID control is used with an LDAP search operation.

When sending this control to a DC running Windows 2000 operating system, the client omits the controlValue field. When sending this control to a DC running Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, or Windows Server 2012 R2 operating system, the client either omits the controlValue field or sets the controlValue field to one of the 32-bit unsigned integer values in the following table. The values are not BER-encoded.

Value name	Value	Description
SO_NORMAL	0	Perform the search as if no LDAP_SERVER_GET_STATS_OID control was included in the search request.
SO_STATS	1	Perform the search and return data related to the resources consumed performing the search, as well as the actual search results.
SO_ONLY_OPTIMIZE	2	Return data related to how the search would be performed, but do not actually return the search results.
SO_EXTENDED_FMT	4	Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2: Returns the data in an alternative format documented later in this section. Windows 2000, Windows Server 2003, and Active Directory Application Mode (ADAM): Not supported.

Omitting the controlValue field is equivalent to specifying the SO_STATS value.

When the server receives a search request with the LDAP_SERVER_GET_STATS_OID control attached to it, it includes a response control in the search response. The controlType field of the returned Control structure is set to the OID of the LDAP_SERVER_GET_STATS_OID control. The controlValue field is included in the returned Control structure.

The response to this control contains information outside the state model. This control instructs the server to return internal data related to how the LDAP search was performed.

For Windows 2000 DCs, the returned controlValue is the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    threadCountTag            INTEGER
    threadCount               INTEGER
    coreTimeTag               INTEGER
    coreTime                  INTEGER
    callTimeTag               INTEGER
    callTime                  INTEGER
    searchSubOperationsTag    INTEGER
    searchSubOperations       INTEGER
}
```

where **threadCountTag**, **coreTimeTag**, **callTimeTag**, and **searchSubOperationsTag** are equal to 1, 2, 3, and 4, respectively. **threadCount** is the number of threads that were processing LDAP requests on the DC at the time the search operation was performed, **coreTime** is the time, in milliseconds, that the core logic in the DC spent processing the request, **callTime** is the overall time, in milliseconds, that the DC spent processing the request, and **searchSubOperations** is the number of individual operations that the DC performed in processing the request.

If the client does not have the SE_DEBUG_PRIVILEGE, a Windows 2000 DC MUST return the value 0 for the **suboperations** field of this structure.

For Windows Server 2003 and ADAM DCs, the controlValue of the response control is the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
    threadCountTag        INTEGER
    threadCount           INTEGER
    callTimeTag           INTEGER
    callTime              INTEGER
    entriesReturnedTag    INTEGER
    entriesReturned       INTEGER
    entriesVisitedTag     INTEGER
    entriesVisited        INTEGER
    filterTag             INTEGER
    filter                OCTET STRING
    indexTag              INTEGER
    index                 OCTET STRING
}
```

In this structure, **threadCountTag**, **threadCount**, **callTimeTag**, and **callTime** are as in the Windows 2000 structure. **entriesReturnedTag**, **entriesVisitedTag**, **filterTag**, and **indexTag** are 5, 6, 7, and 8, respectively. **entriesReturned** is the number of objects returned in the search result. **entriesVisited** is the number of objects that the DC considered for inclusion in the search result. **filter** is a UTF-8 string that represents the optimized form of the search filter that is used by the DC to perform a search. **index** is a string, defined by the system default code page, that indicates which database indexes were used by the DC to perform the search.

If the client does not have the SE_DEBUG_PRIVILEGE, a Windows Server 2003 or ADAM DC MUST return the value 0 for the **entriesReturned** and **entriesVisited** fields of this structure. The server MUST return NULL for the **filter** and **index** fields of this structure.

For Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 DCs, the controlValue of the response control is the BER encoding of the following ASN.1 structure if the SO_EXTENDED_FMT flag is not specified.

```
SEQUENCE {
    threadCountTag        INTEGER
    threadCount           INTEGER
    callTimeTag           INTEGER
    callTime              INTEGER
    entriesReturnedTag    INTEGER
    entriesReturned       INTEGER
    entriesVisitedTag     INTEGER
    entriesVisited        INTEGER
    filterTag             INTEGER
    filter                OCTET STRING
```

```
    indexTag              INTEGER
    index                 OCTET STRING
    pagesReferencedTag    INTEGER
    pagesReferenced       INTEGER
    pagesReadTag          INTEGER
    pagesRead             INTEGER
    pagesPrereadTag       INTEGER
    pagesPreread          INTEGER
    pagesDirtiedTag       INTEGER
    pagesDirtied          INTEGER
    pagesRedirtiedTag     INTEGER
    pagesRedirtied        INTEGER
    logRecordCountTag     INTEGER
    logRecordCount        INTEGER
    logRecordBytesTag     INTEGER
    logRecordBytes        INTEGER
}
```

In this structure, fields with the same name as fields in the Windows Server 2003 structure are as in the Windows Server 2003 structure. **pagesReferencedTag**, **pagesReadTag**, **pagesPrereadTag**, **pagesDirtiedTag**, **pagesRedirtiedTag**, **logRecordCountTag**, and **logRecordCountBytesTag** are 9, 10, 11, 12, 13, 14, and 15, respectively. **pagesReferenced** is the number of database pages referenced by the DC in processing the search. **pagesRead** is the number of database pages read from disk, and **pagesPreread** is the number of database pages preread from disk by the DC in processing the search. **pagesDirtied** is the number of clean database pages modified by the DC in processing the search, while **pagesRedirtied** is the number of previously modified database pages that were modified by the DC in processing the search. **logRecordCount** and **logRecordBytes** are the number and size in bytes, respectively, of database log records generated by the DC in processing the search.

For Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 DCs, if the SO_EXTENDED_FMT flag is specified, an alternative format is used for the controlValue of the response control instead of the format shown previously. Unlike the previous formats in which each statistic is assigned a fixed position within the structure, in the alternative format the ordering of the statistics can change. Rather than relying on position, each statistic has an associated human-readable string that specifies what that statistic is. Additionally, the use of these associated strings alleviates the need to hard-code the positional information into the client-side parser of the response control, permitting the DC to be updated to return addition statistics without necessitating a corresponding client-side change.

When using the alternative format, the controlValue of the response control is the BER encoding of the following ASN.1 structure.

```
SEQUENCE OF SEQUENCE  {
    statisticName         OCTET STRING
    CHOICE {
        [0] intStatistic       INTEGER
        [1] stringStatistic    OCTET STRING
    }
}
```

Effectively, this is an array of statistics, in which each statistic has a human-readable name (the **statisticName** field) and a value. If it is an integer-valued statistic, the value is stored in the **intStatistic** field. If it is a string-valued statistic, the value is stored in the **stringStatistic** field.

When the SO_EXTENDED_FMT flag is specified, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 DCs return the same statistics as if the flag was not specified. The only difference is the format used to return the statistics. The wording of the **statisticName** field is implementation-defined. Currently, the wording as it maps to each statistic as specified in the non-SO_EXTENDED_FMT version of the structure is as follows.

```
threadCount           "Thread count"
callTime              "Call time (in ms)"
entriesReturned       "Entries Returned"
entriesVisited        "Entries Visited"
filter                "Used Filter"
index                 "Used Indexes"
pagesReferenced       "Pages Referenced"
pagesRead             "Pages Read From Disk"
pagesPreread          "Pages Pre-read From Disk"
pagesDirtied          "Clean Pages Modified"
pagesRedirtied        "Dirty Pages Modified"
logRecordCount        "Log Records Generated"
logRecordBytes        "Log Record Bytes Generated"
```

If the client does not have the SE_DEBUG_PRIVILEGE, a Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, or Windows Server 2012 R2 DC MUST return the value 0 for the **entriesReturned**, **entriesVisited**, **pagesReferenced**, **pagesRead**, **pagesPreread**, **pagesDirtied**, **pagesRedirtied**, **logRecordCount**, and **ogRecordBytes** fields, regardless of the format in which the data is returned. The server MUST return NULL for the **filter** and **index** fields, regardless of the format in which the data is returned.

3.1.1.3.4.1.7 LDAP_SERVER_LAZY_COMMIT_OID

The LDAP_SERVER_LAZY_COMMIT_OID control is used to modify the behavior of any LDAP operation. The presence of this control instructs the DC that it may sacrifice durability guarantees on updates to improve performance.

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.8 LDAP_SERVER_PERMISSIVE_MODIFY_OID

The LDAP_SERVER_PERMISSIVE_MODIFY_OID control is used to modify the behavior of an LDAP modify operation. An LDAP modify operation normally returns an error if it attempts to add an attribute that already exists on an object to that object (or, in the case of multivalued attributes, it attempts to add a value that is already present in the attribute). An LDAP modify operation will also normally fail if it attempts to delete an attribute that does not exist on the specified object or that does not contain the value specified in the deletion request. With this control, adding a value to an attribute that already exists and already contains the value to be added will cause the server to return *success* even though no modification was actually performed by the server. Similarly, deletion of an attribute that does not exist or does not contain the specified value will return *success*.

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.9 LDAP_SERVER_NOTIFICATION_OID

The LDAP_SERVER_NOTIFICATION_OID control is used with an LDAP search operation to register the client that is to be notified when changes are made to an object in the directory.

Notifications are asynchronous operations. When the DC receives a search request with this control attached, it does not immediately send a response to the request. Instead, when an object is modified, if that object falls within the scope of the search request to which the LDAP_SERVER_NOTIFICATION_OID control was attached, the DC sends a SearchEntry response that contains the modified object to the client, using the messageID from the original search request (SearchEntry and messageID are defined in [RFC2251] section 4.1.1). The SearchEntry response will contain those attributes of the object that were requested in the original request. These attributes are not necessarily the attributes that were modified. A client indicates that it no longer requires notifications by sending an LDAP abandon operation, specifying the messageID of the original search request.

LDAP search requests that include this control are subject to the following restrictions:

- The only filter permitted in the search request is "(objectclass = *)". The server will return the error *unwillingToPerform* / *<unrestricted>* if this is not the case.
- Base, one-level, and subtree search scopes are permitted. For Windows 2000 operating system DCs, if the base DN specified in a subtree search is not the root of an NC, the server returns the error *unwillingToPerform* / *<unrestricted>*. Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system DCs do not have this restriction.

When sending this control to the DC, the **controlValue** field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its eventual responses.

3.1.1.3.4.1.10 LDAP_SERVER_RANGE_OPTION_OID

LDAP_SERVER_RANGE_OPTION_OID, unlike the other controls discussed in this section, does not actually designate an LDAP extended control. Nonetheless, it is included in this discussion because its OID is found in the supportedControl attribute of the DC's rootDSE. The presence of this OID indicates that the DC supports range retrieval of multivalued attributes. Range retrieval is a mechanism that permits attributes that have too many values to be retrieved in a single LDAP search request to be retrieved via multiple LDAP search requests. Range retrieval is documented in section 3.1.1.3.1.3.3.

Note that although this OID is not present in the supportedControl attribute of Windows 2000 operating system DCs, such DCs nonetheless support range retrieval.

3.1.1.3.4.1.11 LDAP_SERVER_SD_FLAGS_OID

The LDAP_SERVER_SD_FLAGS_OID control is used with an LDAP Search request to control the portion of a Windows Security Descriptor to retrieve. The DC returns only the specified portion of the security descriptors. It is also used with LDAP Add and Modify requests to control the portion of a Windows security descriptor to modify. The DC modifies only the specified portion of the security descriptor.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
    Flags    INTEGER
}
```

The **Flags** value has the following format presented in big-endian byte order. X denotes unused bits that SHOULD be set to 0 by the client and that MUST be ignored by the server.

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

The **Flags** value is a combination of zero or more bit flags from the following table.

Bit flag name and value	Portion of security descriptor to retrieve/update
OWNER_SECURITY_INFORMATION (OSI) 0x1	Owner identifier of the object.
GROUP_SECURITY_INFORMATION (GSI) 0x2	**Primary group** identifier.
DACL_SECURITY_INFORMATION (DSI) 0x4	Discretionary access control list (DACL) of the object.
SACL_SECURITY_INFORMATION (SSI) 0x8	**System access control list (SACL)** of the object.

Specifying **Flags** with no bits set, or not using the LDAP_SERVER_SD_FLAGS_OID control, is equivalent to setting **Flags** to (OWNER_SECURITY_INFORMATION | GROUP_SECURITY_INFORMATION | DACL_SECURITY_INFORMATION | SACL_SECURITY_INFORMATION). Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.12 LDAP_SERVER_SEARCH_OPTIONS_OID

The LDAP_SERVER_SEARCH_OPTIONS_OID control is used with an LDAP Search request to control various behaviors.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
    Flags    INTEGER
}
```

The **Flags** value has the following format presented in big-endian byte order. X denotes unused bits that SHOULD be set to 0 by the client and that MUST be ignored by the server.

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

The **Flags** value is a combination of zero or more bit flags from the following table.

Bit flag name and value	Description
SERVER_SEARCH_FLAG_DOMAIN_SCOPE (SSFDS) 1	Prevents continuation references from being generated when the search results are returned. This performs the same function as the LDAP_SERVER_DOMAIN_SCOPE_OID control.
SERVER_SEARCH_FLAG_PHANTOM_ROOT (SSFPR) 2	For AD DS, instructs the server to search all NC replicas except application NC replicas that are subordinate to the search base, even if the search base is not instantiated on the server. For AD LDS, the behavior is the same except that it also includes application NC replicas in the search. For AD DS and AD LDS, this will cause the search to be executed over all NC replicas (except for application NCs on AD DS DCs) held on the DC that are subordinate to the search base. This enables search bases such as the empty string, which would cause the server to search all of the NC replicas (except for application NCs on AD DS DCs) that it holds.

Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.13 LDAP_SERVER_SORT_OID and LDAP_SERVER_RESP_SORT_OID

This request control and its corresponding response control, LDAP_SERVER_RESP_SORT_OID, are documented in [RFC2891].

DCs only support sorting on a single attribute at a time. Therefore, the client constructs a SortKeyList that contains only one sequence. DCs running Windows 2000 operating system do not support ordering rules when sorting, so the client omits the orderingRule field of the SortKeyList when sending this control to a DC running Windows 2000; sorting uses the *English: United States* sort order. Starting with Windows Server 2003 operating system, DCs support ordering rules for the sort orders specified in the following table; if no ordering rule is specified, the DC uses the *English: United States* sort order. Section 6.5 specifies, by reference to [MS-UCODEREF], the effect of each sort order. Section 2.2.1 specifies the mapping between the sort orders that follow and the LCIDs used in section 6.5.

Ordering rule OID	Sort order
1.2.840.113556.1.4.1461	*Afrikaans*
1.2.840.113556.1.4.1462	*Albanian*
1.2.840.113556.1.4.1463	*Arabic: Saudi Arabia*
1.2.840.113556.1.4.1464	*Arabic: Iraq*
1.2.840.113556.1.4.1465	*Arabic: Egypt*
1.2.840.113556.1.4.1466	*Arabic: Libya*
1.2.840.113556.1.4.1467	*Arabic: Algeria*
1.2.840.113556.1.4.1468	*Arabic: Morocco*
1.2.840.113556.1.4.1469	*Arabic: Tunisia*
1.2.840.113556.1.4.1470	*Arabic: Oman*
1.2.840.113556.1.4.1471	*Arabic: Yemen*
1.2.840.113556.1.4.1472	*Arabic: Syria*
1.2.840.113556.1.4.1473	*Arabic: Jordan*
1.2.840.113556.1.4.1474	*Arabic: Lebanon*
1.2.840.113556.1.4.1475	*Arabic: Kuwait*
1.2.840.113556.1.4.1476	*Arabic: UAE*
1.2.840.113556.1.4.1477	*Arabic: Bahrain*
1.2.840.113556.1.4.1478	*Arabic: Qatar*
1.2.840.113556.1.4.1479	*Armenian*
1.2.840.113556.1.4.1480	*Assamese*
1.2.840.113556.1.4.1481	*Azeri: Latin*
1.2.840.113556.1.4.1482	*Azeri: Cyrillic*
1.2.840.113556.1.4.1483	*Basque*
1.2.840.113556.1.4.1484	*Belarussian*
1.2.840.113556.1.4.1485	*Bengali*
1.2.840.113556.1.4.1486	*Bulgarian*
1.2.840.113556.1.4.1487	*Burmese*
1.2.840.113556.1.4.1488	*Catalan*
1.2.840.113556.1.4.1489	*Chinese: Taiwan*
1.2.840.113556.1.4.1490	*Chinese: PRC*

Ordering rule OID	Sort order
1.2.840.113556.1.4.1491	*Chinese: Hong Kong SAR*
1.2.840.113556.1.4.1492	*Chinese: Singapore*
1.2.840.113556.1.4.1493	*Chinese: Macau SAR*
1.2.840.113556.1.4.1494	*Croatian*
1.2.840.113556.1.4.1495	*Czech*
1.2.840.113556.1.4.1496	*Danish*
1.2.840.113556.1.4.1497	*Dutch*
1.2.840.113556.1.4.1498	*Dutch:Belgium*
1.2.840.113556.1.4.1499	*English: United States*
1.2.840.113556.1.4.1500	*English: United Kingdom*
1.2.840.113556.1.4.1665	*English: Australia*
1.2.840.113556.1.4.1666	*English: Canada*
1.2.840.113556.1.4.1667	*English: New Zealand*
1.2.840.113556.1.4.1668	*English: Ireland*
1.2.840.113556.1.4.1505	*English: South Africa*
1.2.840.113556.1.4.1506	*English: Jamaica*
1.2.840.113556.1.4.1507	*English: Caribbean*
1.2.840.113556.1.4.1508	*English: Belize*
1.2.840.113556.1.4.1509	*English:Trinidad*
1.2.840.113556.1.4.1510	*English: Zimbabwe*
1.2.840.113556.1.4.1511	*English: Philippines*
1.2.840.113556.1.4.1512	*Estonian*
1.2.840.113556.1.4.1513	*Faeroese*
1.2.840.113556.1.4.1514	*Persian*
1.2.840.113556.1.4.1515	*Finnish*
1.2.840.113556.1.4.1516	*French: France*
1.2.840.113556.1.4.1517	*French: Belgium*
1.2.840.113556.1.4.1518	*French: Canada*
1.2.840.113556.1.4.1519	*French: Switzerland*
1.2.840.113556.1.4.1520	*French: Luxembourg*

Ordering rule OID	Sort order
1.2.840.113556.1.4.1521	*French: Monaco*
1.2.840.113556.1.4.1522	*Georgian*
1.2.840.113556.1.4.1523	*German: Germany*
1.2.840.113556.1.4.1524	*German: Switzerland*
1.2.840.113556.1.4.1525	*German: Austria*
1.2.840.113556.1.4.1526	*German: Luxembourg*
1.2.840.113556.1.4.1527	*German: Liechtenstein*
1.2.840.113556.1.4.1528	*Greek*
1.2.840.113556.1.4.1529	*Gujarati*
1.2.840.113556.1.4.1530	*Hebrew*
1.2.840.113556.1.4.1531	*Hindi*
1.2.840.113556.1.4.1532	*Hungarian*
1.2.840.113556.1.4.1533	*Icelandic*
1.2.840.113556.1.4.1534	*Indonesian*
1.2.840.113556.1.4.1535	*Inukitut*
1.2.840.113556.1.4.1536	*Italian:Italy*
1.2.840.113556.1.4.1537	*Italian:Switzerland*
1.2.840.113556.1.4.1538	*Japanese*
1.2.840.113556.1.4.1539	*Kannada*
1.2.840.113556.1.4.1540	*Kashmiri Arabic*
1.2.840.113556.1.4.1541	*Kashmiri*
1.2.840.113556.1.4.1542	*Kazakh*
1.2.840.113556.1.4.1543	*Khmer*
1.2.840.113556.1.4.1544	*Kirghiz*
1.2.840.113556.1.4.1545	*Konkani*
1.2.840.113556.1.4.1546	*Korean*
1.2.840.113556.1.4.1547	*Korean:Johab*
1.2.840.113556.1.4.1548	*Latvian*
1.2.840.113556.1.4.1549	*Lithuanian*
1.2.840.113556.1.4.1550	*Macedonian FYROM*

Ordering rule OID	Sort order
1.2.840.113556.1.4.1551	*Malaysian*
1.2.840.113556.1.4.1552	*Malay Brunei Darussalam*
1.2.840.113556.1.4.1553	*Malayalam*
1.2.840.113556.1.4.1554	*Maltese*
1.2.840.113556.1.4.1555	*Manipuri*
1.2.840.113556.1.4.1556	*Marathi*
1.2.840.113556.1.4.1557	*Nepali:Nepal*
1.2.840.113556.1.4.1558	*Norwegian:Bokmal*
1.2.840.113556.1.4.1559	*Norwegian:Nynorsk*
1.2.840.113556.1.4.1560	*Odia*
1.2.840.113556.1.4.1561	*Polish*
1.2.840.113556.1.4.1562	*Portuguese:Brazil*
1.2.840.113556.1.4.1563	*Portuguese:Portugal*
1.2.840.113556.1.4.1564	*Punjabi*
1.2.840.113556.1.4.1565	*Romanian*
1.2.840.113556.1.4.1566	*Russian*
1.2.840.113556.1.4.1567	*Sanskrit*
1.2.840.113556.1.4.1568	*Serbian:Cyrillic*
1.2.840.113556.1.4.1569	*Serbian:Latin*
1.2.840.113556.1.4.1570	*Sindhi:India*
1.2.840.113556.1.4.1571	*Slovak*
1.2.840.113556.1.4.1572	*Slovenian*
1.2.840.113556.1.4.1573	*Spanish: SpainTraditional Sort*
1.2.840.113556.1.4.1574	*Spanish: Mexico*
1.2.840.113556.1.4.1575	*Spanish: SpainModern Sort*
1.2.840.113556.1.4.1576	*Spanish: Guatemala*
1.2.840.113556.1.4.1577	*Spanish: Costa Rica*
1.2.840.113556.1.4.1578	*Spanish: Panama*
1.2.840.113556.1.4.1579	*Spanish: Dominican Republic*
1.2.840.113556.1.4.1580	*Spanish: Venezuela*

Ordering rule OID	Sort order
1.2.840.113556.1.4.1581	*Spanish: Colombia*
1.2.840.113556.1.4.1582	*Spanish: Peru*
1.2.840.113556.1.4.1583	*Spanish: Argentina*
1.2.840.113556.1.4.1584	*Spanish: Ecuador*
1.2.840.113556.1.4.1585	*Spanish: Chile*
1.2.840.113556.1.4.1586	*Spanish: Uruguay*
1.2.840.113556.1.4.1587	*Spanish: Paraguay*
1.2.840.113556.1.4.1588	*Spanish: Bolivia*
1.2.840.113556.1.4.1589	*Spanish: El Salvador*
1.2.840.113556.1.4.1590	*Spanish: Honduras*
1.2.840.113556.1.4.1591	*Spanish: Nicaragua*
1.2.840.113556.1.4.1592	*Spanish: Puerto Rico*
1.2.840.113556.1.4.1593	*Swahili: Kenya*
1.2.840.113556.1.4.1594	*Swedish*
1.2.840.113556.1.4.1595	*Swedish: Finland*
1.2.840.113556.1.4.1596	*Tamil*
1.2.840.113556.1.4.1597	*Tatar: Tatarstan*
1.2.840.113556.1.4.1598	*Telugu*
1.2.840.113556.1.4.1599	*Thai*
1.2.840.113556.1.4.1600	*Turkish*
1.2.840.113556.1.4.1601	*Ukrainian*
1.2.840.113556.1.4.1602	*Urdu: Pakistan*
1.2.840.113556.1.4.1603	*Urdu: India*
1.2.840.113556.1.4.1604	*Uzbek: Latin*
1.2.840.113556.1.4.1605	*Uzbek: Cyrillic*
1.2.840.113556.1.4.1606	*Vietnamese*
1.2.840.113556.1.4.1607	*Japanese: XJIS*
1.2.840.113556.1.4.1608	*Japanese: Unicode*
1.2.840.113556.1.4.1609	*Chinese: Big5*
1.2.840.113556.1.4.1610	*Chinese: PRCP*

Ordering rule OID	Sort order
1.2.840.113556.1.4.1611	*Chinese: Unicode*
1.2.840.113556.1.4.1612	*Chinese: PRC*
1.2.840.113556.1.4.1613	*Chinese: BOPOMOFO*
1.2.840.113556.1.4.1614	*Korean: KSC*
1.2.840.113556.1.4.1615	*Korean: Unicode*
1.2.840.113556.1.4.1616	*German Phone Book*
1.2.840.113556.1.4.1617	*Hungarian: Default*
1.2.840.113556.1.4.1618	*Hungarian: Technical*
1.2.840.113556.1.4.1619	*Georgian: Traditional*
1.2.840.113556.1.4.1620	*Georgian: Modern*

Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system support an additional sort behavior called "phonetic display name sort". This behavior is triggered by specifying "msDS-PhoneticDisplayName;extended" as the attributeType in the SortKeyList ([RFC2891] section 1.1). When this option is present, the DC checks that the LDAP request satisfies the following requirements:

- The operation is an LDAP search request.
- The orderingRule field specifies the Japanese sort order (namely, "1.2.840.113556.1.4.1538").
- The LDAP_CONTROL_VLVREQUEST control is attached to the search.
- The search request has been sent to a global catalog port (port 3268 or 3269).
- The scope of the search request is wholeSubtree.
- The base object of the search request specifies the DN "".
- The filter is set to (&(showInAddressBook=X)(displayName=*)), where X is a distinguished name and there exists an object O such that O!objectClass = addressBookContainer and O!distinguishedName = X.

If one or more of these criteria are not satisfied, the server returns the error *unwillingToPerform* / *<unrestricted>*.

If all of these criteria are satisfied, the DC performs a phonetic display name sort. In this sort, the search results are sorted on the msDS-PhoneticDisplayName attribute, using the Japanese sort order, in the normal fashion, except that if an object O does not have a value for the msDS-PhoneticDisplayName attribute but does have a value V for the displayName attribute, the server treats V as the value of O!msDS-PhoneticDisplayName for the purposes of the sort.

For example, consider an unsorted search result set consisting of four objects, as shown in the following table. Note that object #2 does not have a value for msDS-PhoneticDisplayName.

Object #	msDS-PhoneticDisplayName value	displayName value
1	A	C
2		D
3	B	E
4	F	C

Assuming for the purpose of this example that the letters A...Z sort in the order {A, ..., Z}, the results of performing a phonetic display name sort on the preceding data is the following.

Object #	msDS-PhoneticDisplayName value	displayName value
1	A	C
3	B	E
2		D
4	F	C

In particular, object #2 was placed before object #4 because the sort treated it as if it had the value "D" for its msDS-PhoneticDisplayName attribute.

3.1.1.3.4.1.14 LDAP_SERVER_SHOW_DELETED_OID

The LDAP_SERVER_SHOW_DELETED_OID control is used with an LDAP operation to specify that tombstones and deleted-objects should be visible to the operation. For example, when the control is used with an LDAP search operation, the search results include any tombstones or deleted-objects that match the search filter.

The following table compares the behavior of the two similar controls LDAP_SERVER_SHOW_DELETED_OID and LDAP_SERVER_SHOW_RECYCLED_OID (section 3.1.1.3.4.1.26).

Extended control name	Deleted-objects	Tombstones	Recycled-objects
LDAP_SERVER_SHOW_DELETED_OID	Visible	Visible	Not Visible
LDAP_SERVER_SHOW_RECYCLED_OID	Visible	Visible	Visible

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.15 LDAP_SERVER_TREE_DELETE_OID

The LDAP_SERVER_TREE_DELETE_OID control is used with an LDAP delete operation to cause the server to recursively delete the entire subtree of objects located underneath the object specified in the delete operation. The object specified in the delete operation is also deleted.

The server deletes between 1 and 16,384 objects. If the server does not delete the entire tree in a single LDAP delete request, it MUST NOT delete the root of the tree (the object specified in the delete operation), and MUST return the error code *adminLimitExceeded* / *ERROR_DS_TREE_DELETE_NOT_FINISHED*.

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.16 LDAP_SERVER_VERIFY_NAME_OID

The LDAP_SERVER_VERIFY_NAME_OID control is used with LDAP Add and Modify requests to identify the global catalog server (GC server) that is used to verify the existence of any objects pointed to by DN attribute values (as specified in section 3.1.1.1.6). If the DC needs to call a GC server while processing the Add or Modify request, it calls the GC server specified in this control. If this control is not used, the DC is free to call any GC server in the forest.

When sending this control to the DC, the **controlValue** field is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    Flags          INTEGER
    ServerName     OCTET STRING
}
```

where **Flags** is ignored and **ServerName** is a UTF-16 encoded Unicode string containing the FQDN of the GC server to contact for verification. Sending this control to the DC does not cause the server to include any controls in its response.

If the LDAP Add or Modify request needs to call a GC server and the server designated by this control in the request is not available or is not a GC server, the Add or Modify request fails with the error *unavailable* / <*unrestricted*>.

3.1.1.3.4.1.17 LDAP_CONTROL_VLVREQUEST and LDAP_CONTROL_VLVRESPONSE

The LDAP_CONTROL_VLVREQUEST control is used with an LDAP search operation to retrieve a subset of the objects that satisfy the search request. This control permits the client to specify a particular object (the "target object") in a sorted set of search results, and to request that the server return a specified number of objects before and after the target object, in addition to the target object itself. "Before" and "after" the target object are relative to the sort order of the search result set. The server will not return objects whose attribute value, used as the sort key, is absent. This control can only be used if the LDAP_SERVER_SORT_OID (section 3.1.1.3.4.1.13) control is also specified.

When sending this control to the DC, the **controlValue** field is set to the BER encoding of the following ASN.1 structure (maxInt is defined in [RFC2251] section 4.1.1):

```
SEQUENCE {
  beforeCount        INTEGER (0..maxInt),
  afterCount         INTEGER (0..maxInt),
  CHOICE {
    byoffset           [0] SEQUENCE {
                           offset       INTEGER (0 .. maxInt),
                           contentCount INTEGER (0 .. maxInt)
                           },
    greaterThanOrEqual [1] AssertionValue
  },
  contextID          OCTET STRING OPTIONAL
}
```

where **beforeCount** indicates how many objects before the target object are to be included in the search results, and **afterCount** indicates how many objects after the target object are to be included in the search results.

byoffset and **greaterThanOrEqual** provide two mutually exclusive ways of specifying the target object. These will now be discussed in turn.

First, the target object can be specified by its position relative to the first object in the sorted set of objects that satisfy the search request, in which case the **byoffset** choice is used. In this case, **contentCount** contains the client's estimation of the total number of objects that satisfy the search criteria. If the client specifies 0 for **contentCount**, it is as if the client had specified a number identical to the server's estimate of the total number of objects that satisfy the search criteria—the quantity **serverContentCount** below. **offset** is used with **contentCount** to specify the position (relative to the first object in the sorted set of search results) of the object to use as the target object according to the following formula:

p = **serverContentCount** * (**offset** / **contentCount**)

where **serverContentCount** is the DC's estimate of the total number of objects that satisfy the search criteria. The object located at position **p** in the sorted list of search results is used as the target object.

A value of **offset** equal to 1 means that the target object is the first object in the search result set, while a value of **offset** equal to **contentCount** means the target object is the last object in the search result set. The **offset** value cannot equal 0 unless **contentCount** also equals 0. If the client specified 0 for **contentCount**, then **p** = **offset** in the preceding formula, so the target object is **offset**-1 objects beyond the first object in the search result set, unless both **offset** and **contentCount** are equal to 0, in which case the previous rule applies.

The second means of specifying the target object is by the **greaterThanOrEqual** choice, instead of the **byoffset** choice. In this case, **greaterThanOrEqual** is an AssertionValue as defined in [RFC2251] section 4.1.7. The target object is the first object in the sorted result set for which the value of the attribute on which it is sorted (that is, the attribute specified by **attributeType** in the LDAP_SERVER_SORT_OID control) is greater than or equal to the value specified by **greaterThanOrEqual**. However, if the sort order is reversed (by specifying that the **reverseOrder** field of the LDAP_SERVER_SORT_OID control is true), then the target object is the first object for which the sort attribute value is less than or equal to the **greaterThanOrEqual** value.

If the **contextID** field is present, it is the opaque value returned by the DC as the **contextIDServer** field of the LDAP_CONTROL_VLVRESPONSE control that was returned with the search response to the previous search over the same "list" as this search. A "list" is a sorted set of search results, defined by a search request value sent to a particular DC over a particular LDAP connection. The client omits this field if this is the first search request that included the LDAP_CONTROL_VLVREQUEST control for the "list", or if the client did not retain the **contextIDServer** field of the previous LDAP_CONTROL_VLVRESPONSE for the "list". The presence or absence of the **contextID** field in the request only affects performance. The **contextID** is valid only on the DC that returned it. If an invalid **contextID** is present, then the LDAP_CONTROL_VLVREQUEST control is ignored.

When the server receives a search request with the LDAP_CONTROL_VLVREQUEST control attached to it, it includes a response control in the search response. The **controlType** field of the returned Control structure is set to the OID of the LDAP_CONTROL_VLVRESPONSE control, and the controlValue is the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
```

```
    targetPosition        INTEGER (0 .. maxInt),
    contentCount          INTEGER (0 .. maxInt),
    virtualListViewResult ENUMERATED {
                              success                   (0),
                              operationsError           (1),
                              unwillingToPerform        (53),
                              insufficientAccessRights  (50),
                              busy                      (51),
                              timeLimitExceeded         (3),
                              adminLimitExceeded        (11),
                              sortControlMissing        (60),
                              offsetRangeError          (61),
                              other                     (80)
                              },
    contextIDServer       OCTET STRING OPTIONAL
}
```

where **targetPosition** is the position of the target object relative to the beginning of the sorted set of search results, **contentCount** is the server's estimate of the total number of objects that satisfy the search request, **contextIDServer** is the opaque value described in the specification of the **contextID** field earlier in this section, and **virtualListViewResult** is an LDAP error code that indicates the success or failure of the DC in processing the LDAP_CONTROL_VLVREQUEST control. These codes have the same meanings as defined for LDAP in [RFC2251], but they pertain specifically to the processing of the control. Error codes *sortControlMissing* and *offsetRangeError* are not defined in [RFC2251]. In the Active Directory implementation of virtual list view (VLV), **virtualListViewResult** is set to error code *sortControlMissing* if the LDAP_SERVER_SORT_OID control is not specified in conjunction with the LDAP_CONTROL_VLVREQUEST control. It is set to error code *offsetRangeError* if **contentCount** is not equal to 0 but **offset** is equal to 0.

The Active Directory implementation of VLV is based on that described in [VLVDRAFT]. Although implementers may consult that document as an informative reference, the preceding description documents the protocol as implemented by Active Directory. No claim is made with regard to Active Directory's conformance or nonconformance with the protocol as specified in [VLVDRAFT].

3.1.1.3.4.1.18 LDAP_SERVER_ASQ_OID

The LDAP_SERVER_ASQ_OID control is used with an LDAP search operation. When this control is used, the search is not performed against the object specified in the search, or the objects located underneath that object, but rather against the set of objects named by an attribute of Object(DS-DN) syntax that is located on the object specified by the base DN of the search request. The specific attribute to use to scope the search is named in the control. Only searches of base object scope can be used with the LDAP_SERVER_ASQ_OID control.

For example, suppose there is an object o and a multivalued attribute A of Object(DS-DN) syntax such that o.A contains the DNs of objects o1, o2, and o3. An LDAP base-scope search operation that targets object o, with the LDAP_SERVER_ASQ_OID control attached and specifying the A attribute, will cause the server to perform the search not against object o but against objects o1, o2, and o3.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    sourceAttribute    OCTET STRING
}
```

where **sourceAttribute** is a UTF-8 string that specifies the LDAP display name of the attribute to use to scope the search (for example, attribute A in the previous example).

When the server receives a search request with the LDAP_SERVER_ASQ_OID control attached to it, it includes a response control in the search response. The controlType field of the returned Control structure is set to the OID of the LDAP_SERVER_ASQ_OID control, and the controlValue is the BER encoding of the following ASN.1 structure:

```
SEQUENCE {
    searchResults    ENUMERATED {
                         success                   (0),
                         invalidAttributeSyntax    (21),
                         unwillingToPerform        (53),
                         affectsMultipleDSAs       (71)
                         },
}
```

where the meaning of **searchResults** is as indicated in the following table.

searchResult name	searchResult value	Description
success	0	Search results are returned for all objects referenced by sourceAttribute.
invalidAttributeSyntax	21	sourceAttribute is not of Object(DS-DN) syntax.
unwillingToPerform	53	The search scope was not set to base object scope.
affectsMultipleDSAs	71	Partial results were returned, but not all the objects were available on the DC.

The search results consist of each object that is specified by the **sourceAttribute** attribute, and that matches the search filter returned as a SearchResultEntry (defined in [RFC2251] section 4.5.2) containing the attributes specified in the attribute list of the search request. If any of the objects specified by **sourceAttribute** are not available on the DC, the search results include all of the objects that are available on the DC, and the **searchResults** return value is set to the *affectsMultipleDSAs* error code to indicate that some data that might be otherwise available is not present in the results.

3.1.1.3.4.1.19 LDAP_SERVER_QUOTA_CONTROL_OID

This control is used with an LDAP search operation to retrieve the quota of a user. When used with an LDAP search operation that queries the constructed attributes msDS-QuotaEffective and msDS-QuotaUsed on the msDS-QuotaContainer object, the server will return the quota of the user who is specified by the control, rather than the quota of the user whom the connection is authenticated as.

If the caller attempts to retrieve the quota of a user other than the user whom the caller is authenticated as, and the caller does not have the RIGHT_DS_READ_PROPERTY right on the Quotas container (described in section 6.1.1.4.3), the server returns an empty result set.

If the caller attempts to retrieve the quota of the user whom the caller is authenticated as, and the caller has neither the RIGHT_DS_READ_PROPERTY right on the Quotas container (described in section 6.1.1.4.3) nor the DS-Query-Self-Quota control access right on the Quotas container, the server returns an empty result set.

These access checks are also specified in section 3.1.1.4.4.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
    querySID    OCTET STRING
}
```

Where **querySID** is the SID, in binary form, of the user whose quota is to be retrieved (the binary form of SIDs is documented in [MS-DTYP] section 2.4.2). Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.20 LDAP_SERVER_SHUTDOWN_NOTIFY_OID

This control is used with an LDAP Search request. The Search request has base object scope. The base DN of the search is the DN of the DC's nTDSDSA object, and the search filter is "(objectClass=*)". If the application sending the search request is not running on the same computer as the DC, the result is the error *unwillingToPerform* / *<unrestricted>*.

When sending this control to the DC, the controlValue field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

This control is only supported on the Small Business Server version of the Windows operating system.

Because this control only has an effect for applications running on the same machine as the DC, the effects of this control are not observable on the network. This control causes the DC to notify the client when the DC is shutting down. When the DC receives a search request with this control attached, it does not immediately send a response to the request. Instead, it sends the SearchResultDone response (see [RFC2251] section 4.5.2) to the request when the DC is shutting down.

3.1.1.3.4.1.21 LDAP_SERVER_FORCE_UPDATE_OID

A DC does not perform originating updates that do not affect the state of the DC. For example, given an LDAP Modify operation that sets the value of an attribute A to a value V, if the value of A is already V prior to the Modify operation, the DC skips the update and returns success. The stamp associated with A is not changed, and the Modify operation does not cause **replication traffic**.

When the LDAP_SERVER_FORCE_UPDATE_OID control is attached to an update operation, the DC does not perform the optimization described in the previous paragraph. The update always generates a new stamp for the attribute or link value and always replicates.

When sending this control to a DC, the controlValue field of the Control structure is omitted. Sending this control to a DC does not cause the DC to include any controls in its response.

3.1.1.3.4.1.22 LDAP_SERVER_RANGE_RETRIEVAL_NOERR_OID

This control is used to modify the behavior of a range retrieval operation (see section 3.1.1.3.1.3.3). When this control is not specified, if range retrieval is being performed on an attribute whose values are forward link values or back link values, and the value of **low** is greater than or equal to the number of values in the attribute, the DC will return the error *operationsError* / *<unrestricted>*. If this control is specified, no error is returned in this case (and no values are returned). For example, if an object has a member attribute with 500 values, performing the range retrieval

"member;range=500-*" will return *operationsError* / *<unrestricted>* without this control, and *success* with this control.

When sending this control to a DC, the controlValue field of the Control structure is omitted. Sending this control to a DC does not cause the DC to include any controls in its response.

3.1.1.3.4.1.23 LDAP_SERVER_RODC_DCPROMO_OID

If this control is specified and the caller does not have the DS-Install-Replica control access right on the root of the default NC, the result is the error *insufficientAccessRights* / *ERROR_ACCESS_DENIED*.

If the request is an Add of an object of class user or a subclass of user, the presence of this control has the following effects:

- The DC generates a value in the range [1 .. 65535] that is not used as a value of the msDS-SecondaryKrbTgtNumber attribute on an object in this domain, and assigns the generated value to the msDS-SecondaryKrbTgtNumber attribute of the created object. If no such value exists, the result is the error *other* / *ERROR_NO_SYSTEM_RESOURCES*.
- The generated value for msDS-SecondaryKrbTgtNumber is appended (in decimal form) to the string "krbtgt", and the resulting string is assigned to the sAMAccountName attribute on the created object.
- The userAccountControl bits ADS_UF_ACCOUNT_DISABLE and ADS_UF_DONT_EXPIRE_PASSWD (section 2.2.15) are set on the object's userAccountControl attribute.
- The object's account password is set to a randomly generated value that satisfies all criteria in [MS-SAMR] section 3.1.1.7.2 and is processed as described in [MS-SAMR] section 3.1.1.8.5.

  Note In Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, the DC servicing the request need not be the PDC FSMO role owner.

If the request is an Add of an object of class nTDSDSA, the presence of this control has the following effects:

- The DC creates the nTDSDSA object using the information provided in the Add request. The only special effect of the control is to perform the checking of the DS-Install-Replica control access right (specified previously in this section) to authorize the nTDSDSA object creation. Without this control, an Add that attempts to create an nTDSDSA object will fail because the class is system-only (section 3.1.1.2.4.8).

When sending this control to a DC, the **controlValue** field of the Control structure is omitted. Sending this control to a DC does not cause the DC to include any controls in its response.

3.1.1.3.4.1.24 LDAP_SERVER_DN_INPUT_OID

This control is used to specify the DN of an object during certain LDAP operations.

When used with an LDAP search operation that queries the constructed attribute msDS-IsUserCachableAtRodc on a computer object that represents an RODC, the server will return the administrative policy regarding whether the secret attributes of the security principal represented by the DN specified in the control can be cached on the RODC. If the caller does not have the Read-Only-Replication-Secret-Synchronization control access right on the root of the default NC, the error

operationsError / *ERROR_DS_CANT_RETRIEVE_ATTRS* is returned. This access check is also specified in section 3.1.1.4.4.

When sending this control to the DC, the controlValue field is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
    InputDN    OCTET STRING
}
```

Where **InputDN** is a UTF-8 encoding of the DN of a security principal. The DN may be either an RFC 2253–style DN or one of the alternative DN formats described in section 3.1.1.3.1.2.4.

3.1.1.3.4.1.25 LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID

The LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID control is used with an LDAP search operation to specify that link attributes that refer to deleted-objects are visible to the search operation. If used in conjunction with LDAP_SERVER_SHOW_DELETED_OID or LDAP_SERVER_SHOW_RECYCLED_OID, link attributes that are stored on deleted-objects are also visible to the search operation. This applies to both the search filter and the set of attributes returned by the search operation. When this control is not used, linked attribute values referring to deleted-objects and link valued attributes stored on deleted-objects are not visible to search operation filters, and are not returned as requested attributes for the search operation.

Extended control names	Link values neither stored on nor referring to deleted-objects	Link values not stored on but referring to deleted-objects	Link values stored on deleted-objects but not referring to deleted-objects	Link values stored on and referring to deleted-objects
LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID	Visible	Visible	Not Visible	Not Visible
LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID in conjunction with LDAP_SERVER_SHOW_DELETED_OID or LDAP_SERVER_SHOW_RECYCLED_OID	Visible	Visible	Visible	Visible

When sending this control to the DC, the **controlValue** field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.26 LDAP_SERVER_SHOW_RECYCLED_OID

The LDAP_SERVER_SHOW_RECYCLED_OID control is used with an LDAP operation to specify that tombstones, deleted-objects, and recycled-objects should be visible to the operation. For example, when the control is used with an LDAP search operation, the search results include any tombstones, deleted-objects, or recycled-objects that match the search filter.

The following table compares the behavior of the two similar controls LDAP_SERVER_SHOW_DELETED_OID (section 3.1.1.3.4.1.14) and LDAP_SERVER_SHOW_RECYCLED_OID.

Extended control name	Deleted-objects	Tombstones	Recycled-objects
LDAP_SERVER_SHOW_DELETED_OID	Visible	Visible	Not Visible
LDAP_SERVER_SHOW_RECYCLED_OID	Visible	Visible	Visible

When sending this control to the DC, the **controlValue** field of the Control structure is omitted. Sending this control to the DC does not cause the server to include any controls in its response.

3.1.1.3.4.1.27 LDAP_SERVER_POLICY_HINTS_OID

The LDAP_SERVER_POLICY_HINTS_OID control is used with an LDAP operation to enforce the password history length constraint ([MS-SAMR] section 3.1.1.7.1) during password set. The password history policy sets how frequently old passwords can be reused.

When sending this control to the DC, the **controlValue** field is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
    Flags    INTEGER
}
```

where **Flags** tells the server whether to apply the password history length constraint on password-set operations. If it is 0x1, then that constraint will be enforced. Otherwise, the constraint is not enforced.

3.1.1.3.4.1.28 LDAP_SERVER_POLICY_HINTS_DEPRECATED_OID

The LDAP_SERVER_POLICY_HINTS_DEPRECATED_OID control has the exact semantics and behaviors as LDAP_SERVER_POLICY_HINTS_OID (section 3.1.1.3.4.1.27); this control MAY be used by clients when the server does not support LDAP_SERVER_POLICY_HINTS_OID. Clients SHOULD use LDAP_SERVER_POLICY_HINTS_OID when it is supported by the server.

3.1.1.3.4.1.29 LDAP_SERVER_DIRSYNC_EX_OID

The LDAP_SERVER_DIRSYNC_EX_OID control is used with an LDAP search operation in exactly the same way as the LDAP_SERVER_DIRSYNC_OID control, except for differences specified in this section. All ASN.1 structures and the meaning of the fields of those structures are the same.

As with the LDAP_SERVER_DIRSYNC_OID control, any attributes can be requested in the search. Only those objects for which these attributes have been created or modified since the time represented by **Cookie** will be considered for inclusion in the search. However, where the LDAP_SERVER_DIRSYNC_OID control returns only those attributes that have changed, the LDAP_SERVER_DIRSYNC_EX_OID control also returns unchanged attributes when the attribute name in the request is appended with the string ";dirSyncAlwaysReturn".

3.1.1.3.4.1.30 LDAP_SERVER_UPDATE_STATS_OID

The LDAP_SERVER_UPDATE_STATS_OID control can be used with any LDAP operation. When sending this control to the DC, the **controlValue** field of the Control structure is omitted.

When the server receives a request with the LDAP_SERVER_UPDATE_STATS_OID control attached to it, the server includes a response control in the response that contains statistics. The **controlType** field of the returned Control structure is set to the OID of the LDAP_SERVER_UPDATE_STATS_OID control. The **controlValue** field is included in the returned Control structure.

The returned **controlValue** field is the BER encoding of the following ASN.1 structure:

```
SEQUENCE OF SEQUENCE {
    statID       LDAPOID
    statValue    OCTET STRING
}
```

where **statID** is an OID that corresponds to a specific statistic name, and **statValue** is a value related to that statistic. Each statistic specifies an encoding for its value.

The following table specifies the statistics that a DC MUST return. A DC MAY return other implementation-defined statistics. No other statistics are returned by DCs in any Windows Server operating system.

Statistic name	OID (specified by statID)
Highest USN Allocated	1.2.840.113556.1.4.2208
Invocation ID Of Server	1.2.840.113556.1.4.2209

3.1.1.3.4.1.30.1 Highest USN Allocated

The **statValue** for this **statID** contains the highest USN that the DC allocated during the LDAP operation. USNs allocated by an LDAP operation make up a set of USNs such that no LDAP operation other than the current operation can write the USN into the DC's state. Note that while no other LDAP operation can write these USNs, it is not required that the current operation actually write any or all of these USNs. If the USNs allocated by this LDAP operation make up the empty set, a value of 0 is returned in the **statValue**.

The value in the **statValue** field is a 64-bit integer, in little-endian byte order.

3.1.1.3.4.1.30.2 Invocation ID Of Server

The **statValue** for this **statID** contains dc.invocationId (section 3.1.1.1.9). This value is returned in little-endian byte order.

3.1.1.3.4.1.31 LDAP_SERVER_TREE_DELETE_EX_OID

The LDAP_SERVER_TREE_DELETE_EX_OID control is used with an LDAP delete operation to cause the server to recursively delete the entire subtree of objects located underneath the object specified in the delete operation. The object specified in the delete operation is also deleted.

When sending this control to the DC, the **controlValue** field is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
    countOfObjectsToDelete    INTEGER
}
```

where **countOfObjectsToDelete** is a limit on the number of objects that will be deleted while processing this control. If the value of **countOfObjectsToDelete** is less than 2, then the value 2 is used rather than the value specified. If the value of **countOfObjectsToDelete** is greater than 16,384, then the value 16,384 is used.

The server deletes between 1 and **countOfObjectsToDelete** objects, inclusive. If the server does not delete the entire tree in a single LDAP delete request, it MUST NOT delete the root of the tree (the object specified in the delete operation), and MUST return the error code *adminLimitExceeded* / *ERROR_DS_TREE_DELETE_NOT_FINISHED*.

3.1.1.3.4.1.32 LDAP_SERVER_SEARCH_HINTS_OID

The LDAP_SERVER_SEARCH_HINTS_OID control is used with an LDAP search operation. This control supplies hints to the search operation on how to satisfy the search. When sending this control to the DC, the **controlValue** field is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE OF SEQUENCE{
    hintId    LDAPOID
    hintValue OCTET STRING
}
```

where **hintId** is an OID that corresponds to a specific hint name, and **hintValue** is a value related to that hint. Each hint specifies an encoding for its value.

The following table specifies the hints that a DC MUST honor. A DC MAY honor other implementation-defined search hints. No other search hints are honored by DCs in any Windows Server operating system.

Statistic name	OID (as specified by hintId)
Require Sort Index	1.2.840.113556.1.4.2207
Soft Size Limit	1.2.840.113556.1.4.2210

Multiple instances of the LDAP_SERVER_SEARCH_HINTS_OID control can be included with a single LDAP search operation. The hints are applied in the order in which the controls are encoded in the LDAP request; that is, a later hint can override an earlier hint, overriding both **hintValue** and control criticality. This behavior allows the application of different criticality to individual hints.

If the control is critical and an unrecognized search hint is specified, the DC returns the error *unwillingToPerform* / <*unrestricted*>. If the control is not critical, unrecognized hints are ignored.

3.1.1.3.4.1.32.1 Require Sort Index

The **hintValue** for this hint is a BER encoding specified by the following ASN.1 structure:

```
SEQUENCE {
    IndexOnly    BOOLEAN
}
```

If the value of **IndexOnly** is false, or if no LDAP_SERVER_SORT_OID control accompanies the LDAP_SERVER_SEARCH_HINTS_OID control, then the hint is ignored.

This hint suggests to the DC that it use an index (as specified by the search flags IX and PI in section 2.2.9) over the attribute specified in the LDAP_SERVER_SORT_OID control to satisfy the search.

If the sort control is critical and no index is available, the search will fail with the error *DB_ERR_CANT_SORT* / *<unrestricted>*.

If the sort control is not critical and no index is available, the hint is ignored.

Exactly what an index is in relationship to a DC is implementation-specific. Therefore, the determination that an index is not available is not constrained by the protocol, but rather is implementation-specific. This hint is provided only as a facility to make suggestions to a DC that it favor search-operation execution that is based on information specified in the sort control rather than information that is specifically derived from the scope of the search, the filter, or any other parameters of the search.

3.1.1.3.4.1.32.2 Soft Size Limit

The **hintValue** for this hint is a BER encoding specified by the following ASN.1 structure:

```
SEQUENCE {
    limitValue    INTEGER
}
```

If an LDAP_SERVER_SORT_OID control does not accompany this hint, this hint is ignored.

Given that the value of **LimitValue** is *X*, given an imposed LDAP size limit of *Y* (whether specified in the LDAP search operation or imposed by an implementation-specific default value), and given that a sort order is specified in an LDAP_SERVER_SORT_OID control, when these values are all applied to an LDAP search operation, the LDAP search operation conceptually results in a list of objects to return as a response to the request. Due to the size limit, the cardinality of the list is less than or equal to *Y*. The elements in the list are ordered by the attribute specified in the LDAP_SERVER_SORT_OID control. If the list of objects contains fewer than *X* objects, or exactly *X* objects, then the Soft Size Limit hint has no affect. If the LDAP search operation identifies more than *X* objects, then any objects in the list subsequent to the X^{th} object that do not have a value of the sort attribute that is equal to the sort value of the X^{th} object (as defined by the equality comparison rules for that attribute) are removed from the list before the response is returned to the client.

On a return value of *success* from the LDAP search operation, if fewer than *X* values are returned, then all objects satisfying the search operation have been returned. On a return value of *success*, if *X* or more and fewer than *Y* objects are returned, then all objects satisfying the LDAP search operation with values for the sort attribute less than or equal to the X^{th} object have been returned, but there might be more objects satisfying the search with greater values for the sort attribute. On a return value of *sizeLimitExceeded*, not all objects with values for the sort attribute equivalent to the X^{th} object have been returned.

3.1.1.3.4.1.33 LDAP_SERVER_EXPECTED_ENTRY_COUNT_OID

The LDAP_SERVER_EXPECTED_ENTRY_COUNT_OID is used with an LDAP search operation to potentially modify the return code of the operation.

When sending this control to the DC, the **controlValue** field is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE {
    searchEntriesMin    INTEGER
    searchEntriesMax    INTEGER
}
```

When the search operation would normally return *success* / *<unrestricted>* and the number of **searchEntries** returned by the search is less than **searchEntriesMin** or greater than **searchEntriesMax**, the return code of the search operation is modified to be *constraintViolation* / *<unrestricted>*. Note that this control affects only the return value of the search operation. It does not affect any other part of the returned data from the search operation.

3.1.1.3.4.1.34 LDAP_SERVER_SET_OWNER_OID

The LDAP_SERVER_SET_OWNER_OID is used with an LDAP add operation to specify the owner of the object to be created. The owner is to be set into the owner portion of the security descriptor stored in the ntSecurityDescriptor attribute of the object to be created.

When sending this control to the DC, the **controlValue** field is set to the BER encoding of the following ASN.1 structure.

```
SID octetString
```

The supplied SID value is a valid SDDL string representation of a SID ([MS-DTYP] section 2.4.2.1).

If an owner is specified both via this control and via a value for the nTSecurityDescriptor attribute, the value specified by this control takes precedence.

3.1.1.3.4.1.35 LDAP_SERVER_BYPASS_QUOTA_OID

The LDAP_SERVER_BYPASS_QUOTA_OID is used with an LDAP add operation to specify that exceeding quota limitations MUST NOT cause the add to fail.

3.1.1.3.4.2 LDAP Extended Operations

LDAP extended operations are an extensibility mechanism in version 3 of LDAP, as discussed in [RFC2251] section 4.12. The following sections describe the LDAP extended operations that are implemented by DCs in Windows Server 2003 operating system, Active Directory Application Mode (ADAM), Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system.

The LDAP extended operations supported by a DC are exposed as OIDs in the supportedExtension attribute of the rootDSE. Each OID is mapped to a human-readable name as shown in the following table.

Extended operation name	OID
LDAP_SERVER_FAST_BIND_OID	1.2.840.113556.1.4.1781
LDAP_SERVER_START_TLS_OID	1.3.6.1.4.1.1466.20037
LDAP_TTL_REFRESH_OID	1.3.6.1.4.1.1466.101.119.1

Extended operation name	OID
LDAP_SERVER_WHO_AM_I_OID	1.3.6.1.4.1.4203.1.11.3
LDAP_SERVER_BATCH_REQUEST_OID	1.2.840.113556.1.4.2212

Only Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 DCs support extended operations. The following table specifies the set of LDAP extended operations supported in each Windows Server operating system or ADAM version that supports extended operations.

Extended operation name	Windows Server 2003	ADAM	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
LDAP_SERVER_FAST_BIND_OID	X	X	X	X	X	X
LDAP_SERVER_START_TLS_OID	X	X	X	X	X	X
LDAP_TTL_REFRESH_OID	X	X	X	X	X	X
LDAP_SERVER_WHO_AM_I_OID		X	X	X	X	X
LDAP_SERVER_BATCH_REQUEST_OID					X	X

Each of these operations is executed by performing an LDAP ExtendedRequest operation, specifying the OID of the extended operation as the requestName field in the ExtendedRequest (see [RFC2251] section 4.12). The server responds to an ExtendedRequest by returning an ExtendedResponse, the fields of which are also documented in section 4.12 of the RFC.

3.1.1.3.4.2.1 LDAP_SERVER_FAST_BIND_OID

The presence of this OID in the supportedExtension attribute indicates that the DC provides support for fast bind mode. In fast bind mode, the server validates (authenticates) the credentials of LDAP bind requests that are sent on the connection. However, unlike a regular (non–fast bind mode) bind, the DC performs authentication only. The DC does not perform **authorization** steps, such as computing the group memberships of the authenticated security principal.

The LDAP_SERVER_FAST_BIND_OID operation puts the LDAP connection on which it was sent into fast bind mode on the DC. The server will reject this operation with the error *unwillingToPerform* / *ERROR_DS_UNWILLING_TO_PERFORM* if a successful bind has already been performed on the connection.

Note that a client can retrieve the supportedExtension attribute from the root DSE without having first performed a bind (since the supportedExtension attribute is anonymously accessible, and LDAPv3 does not require a bind to be performed for anonymous access). A client MUST NOT specify any control other than LDAP_SERVER_EXTENDED_DN_OID when querying the root DSE anonymously. Thus, a client can determine if the server supports fast bind mode without first having to bind to the server.

Only **simple binds** are accepted on a connection in this mode. All other types of bind operations are rejected with the error *unwillingToPerform* / *ERROR_DS_INAPPROPRIATE_AUTH*. The connection is always treated as if no bind had occurred for the purposes of all other LDAP operations; that is, the connection is treated as the anonymous user (in other words, an anonymous bind).

To send this extended operation to the DC, the client sends an LDAP ExtendedRequest with the **requestName** field containing the operation's OID. The **requestValue** field is omitted. The server will return an ExtendedResponse with the **responseName** field containing the operation's OID and the response field omitted.

The following shows a typical sequence of operations in fast bind:

1. The client establishes an LDAP connection with the DC.
2. (Optional) The client checks the supportedExtension attribute on the root DSE to confirm that the DC supports fast bind mode.
3. The client sends the LDAP_SERVER_FAST_BIND_OID extended operation to the DC to put the LDAP connection into fast bind mode.
4. The client performs one or more simple binds on the connection.

3.1.1.3.4.2.2 LDAP_SERVER_START_TLS_OID

This presence of this OID in the supportedExtension attribute indicates that the DC provides support for the LDAP StartStopTLS protocol as described in [RFC2830].

A connection cannot be put into TLS mode if it is using an integrity validation or encryption mechanism that was negotiated as part of a bind request (for example, a SASL-layer encryption mechanism). Such an attempt will be rejected with the error *operationsError* / *ERROR_SUCCESS*.

3.1.1.3.4.2.3 LDAP_TTL_REFRESH_OID

The presence of this OID in the supportedExtension attribute indicates that the DC provides support for dynamic objects as defined in [RFC2589]. This extended operation is sent to the DC to refresh a specific dynamic object that has already been created. The extended operation is documented in [RFC2589]. The refresh operation is treated as a modify operation (section 3.1.1.5.3) of the entryTTL attribute (section 3.1.1.4.5.12).

If the modify is successful, the responseTtl field ([RFC2589] section 4.2) is populated from the dynamic object's entryTTL constructed attribute according to section 3.1.1.4.5.12, using the msDS-Entry-Time-To-Die (section 3.1.1.5.3.3) and DynamicObjectMinTTL (section 3.1.1.3.4.7) attributes, and honoring the dynamic object's requirements, as specified in section 6.1.7.

3.1.1.3.4.2.4 LDAP_SERVER_WHO_AM_I_OID

The presence of this OID in the supportedExtension attribute indicates that the DC provides support for the "Who Am I?" LDAP extended operation described in [RFC4532]. Active Directory implements this operation in conformance with that RFC.

If the client is authenticated as a Windows security principal, the authzId returned in the response will contain the string "u:" followed by either (1) the NetBIOS domain name, followed by a backslash ("\"), followed by the sAMAccountName of the security principal, or (2) the SID of the security principal, in SDDL SID string format ([MS-DTYP] section 2.4.2.1). If the client is authenticated as an AD LDS security principal, the returned authzId will contain the string "dn:" followed by the DN of

the security principal. If the client has not authenticated, the returned authzId will be the empty string.

Active Directory does not implement Proxied Authentication Control of [RFC4370], so section 4.1 of [RFC4532] is not applicable to Active Directory.

3.1.1.3.4.2.5 LDAP_SERVER_BATCH_REQUEST_OID

The presence of this OID in the supportedExtension attribute indicates that the DC provides support for the batched LDAP extended operation. In a batched LDAP extended operation, the DC accepts an extended operation that contains a sequence of LDAP messages (that is, LDAP operations) encoded and packed into the operation data and then operates on the individual messages sequentially.

When sending this extended operation to the DC, the data field is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE of OCTET STRING
```

Each OCTET STRING contains a BER encoded ([ITUX690]) **LDAPMessage** as defined in [RFC2251].

The DC MUST support the following values of the **protocolOp** field of an LDAP message.

- searchRequest
- modifyRequest
- addRequest
- deleteRequest

The DC MAY support any of the other legal values of the **protocolOp** field of an LDAP message. No version of Windows Server operating system supports any of these other values.

The DC MUST accept the following controls (defined in section 3.1.1.3.4.1) as part of the encoded **LDAPMessage**:

- LDAP_SERVER_DOMAIN_SCOPE_OID
- LDAP_SERVER_EXTENDED_DN_OID
- LDAP_SERVER_GET_STATS_OID
- LDAP_SERVER_PERMISSIVE_MODIFY_OID
- LDAP_SERVER_SD_FLAGS_OID
- LDAP_SERVER_SEARCH_OPTIONS_OID
- LDAP_SERVER_SHOW_DELETED_OID
- LDAP_SERVER_DN_INPUT_OID
- LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID
- LDAP_SERVER_SHOW_RECYCLED_OID

The DC MAY support other controls. No version of Windows Server supports any other controls.

If the DC returns any return code for the batched LDAP extended operation other than *success* / <*unrestricted*>, then the DC returns no data for the batched LDAP extended operation.

If the DC returns any data for the batched LDAP extended operation, the data is set to the BER encoding of the following ASN.1 structure.

```
SEQUENCE of LDAPMessage
```

If the DC receives an **LDAPMessage** containing unsupported **protocolOp** values or controls, or if the data for the batched LDAP extended operation is not a legal BER encoding as required, the DC must return the error *protocolError* / <*unrestricted*>.

If the number of individual messages in the return data exceeds the DC's limit, the overall batched LDAP extended operation returns the error *sizeLimitExceeded* / <*unrestricted*>. This limit is controlled by the MaxBatchReturnMessages LDAP policy (see section 3.1.1.3.4.6).

If the amount of time spent processing the batched LDAP extended operation exceeds the DC's limit, the overall batched LDAP extended operation returns the error *timeLimitExceeded* / *ERROR_INVALID_PARAMETER*. This limit is implementation-defined. In Windows Server 2012 operating system and Windows Server 2012 R2 operating system, this limit is controlled by the MaxQueryDuration LDAP policy (see section 3.1.1.3.4.6).

If any operation in a batched LDAP extended operation results in an LDAP return code other than *success* / <*unrestricted*>, then all subsequent operations in that batched LDAP operation are not performed and all prior operations are "rolled back"; that is, no changes that would have been caused by the operations are committed to the DC's state. Note that, other than where explicitly stated, the return codes of these individual operations do not affect the return code of the batched LDAP extended operation.

If an individual operation in the batched LDAP extended operation returns *busy* / <*unrestricted*>, then the batched LDAP extended operation returns the return code generated by that individual operation.

If no other error conditions are present, the DC returns the error code *success* / <*unrestricted*>.

If the DC returns any return code for the batched LDAP extended operation other than *success* / <*unrestricted*>, then all operations in that batched LDAP operation are "rolled back"; that is, no changes caused by the operations are committed to the DC's state.

The returned data for the batched LDAP extended operation is the sequence containing the return messages generated by performing the individual operations encoded in the incoming data. Note especially that if an individual operation fails, causing the whole sequence to be interrupted and "rolled back", the return sequence of messages includes all messages generated up to and including the message returning the individual operation's failure code. In this case, the returned data can show successful modifications to DC state, but since the final message in the incoming sequence of operations was not completed with a successful return code, these messages indicate only that the operations that modify the DC state would have succeeded and been committed if they had been the last operation in the sequence of messages; that is, these messages indicate that the operations up to the operation that failed would have succeeded.

3.1.1.3.4.3 LDAP Capabilities

The following sections specify the capabilities exposed by DCs on the supportedCapabilities attribute of the rootDSE. Capabilities are exposed in that attribute as OIDs, each of which is mapped to a human-readable name, as shown in the following table.

Capability name	OID
LDAP_CAP_ACTIVE_DIRECTORY_OID	1.2.840.113556.1.4.800
LDAP_CAP_ACTIVE_DIRECTORY_LDAP_INTEG_OID	1.2.840.113556.1.4.1791
LDAP_CAP_ACTIVE_DIRECTORY_V51_OID	1.2.840.113556.1.4.1670
LDAP_CAP_ACTIVE_DIRECTORY_ADAM_DIGEST	1.2.840.113556.1.4.1880
LDAP_CAP_ACTIVE_DIRECTORY_ADAM_OID	1.2.840.113556.1.4.1851
LDAP_CAP_ACTIVE_DIRECTORY_PARTIAL_SECRETS_OID	1.2.840.113556.1.4.1920
LDAP_CAP_ACTIVE_DIRECTORY_V60_OID	1.2.840.113556.1.4.1935
LDAP_CAP_ACTIVE_DIRECTORY_V61_R2_OID	1.2.840.113556.1.4.2080
LDAP_CAP_ACTIVE_DIRECTORY_W8_OID	1.2.840.113556.1.4.2237

Not all versions of Windows Server operating system and Active Directory Application Mode (ADAM) support all the LDAP capabilities. The following table indicates which capabilities are supported in which version.

Capability name	Windows 2000 operating system	Windows 2000 Server operating system SP3	Windows Server 2003 operating system	ADAM RTW	ADAM SP1	Windows Server 2008 operating system ADDS	Windows Server 2008 ADLDS	Windows Server 2008 R2 operating system ADDS	Windows Server 2008 R2 ADLDS	Windows Server 2012 operating system ADDS	Windows Server 2012 ADLDS	Windows Server 2012 R2 operating system ADDS	Windows Server 2012 R2 ADLDS
LDAP_CAP_ACTIVE_DIRECTORY_OID	X	X	X			X		X		X		X	
LDAP_CAP_ACTIVE_DIRECTORY_LDAP_INTEG_OID		X	X	X	X	X	X	X	X	X	X	X	X
LDAP_CAP_ACTIVE_DIRECTORY_V51_OID			X			X	X	X	X	X	X	X	X

Capability name	Windows 2000 operating system	Windows 2000 Server operating system SP3	Windows Server 2003 operating system	ADAMRTW	ADAMSP1	Windows Server 2008 operating system ADDS	Windows Server 2008 ADLDS	Windows Server 2008 R2 operating system ADDS	Windows Server 2008 R2 ADLDS	Windows Server 2012 operating system ADDS	Windows Server 2012 ADLDS	Windows Server 2012 R2 operating system ADDS	Windows Server 2012 R2 ADLDS
LDAP_CAP_ACTIVE_DIRECTORY_ADAM_DIGEST					X*		X*		X*		X*		X*
LDAP_CAP_ACTIVE_DIRECTORY_ADAM_OID				X	X		X		X		X		X
LDAP_CAP_ACTIVE_DIRECTORY_PARTIAL_SECRETS_OID						X*		X*		X*		X*	
LDAP_CAP_ACTIVE_DIRECTORY_V60_OID						X	X	X	X	X	X	X	X
LDAP_CAP_ACTIVE_DIRECTORY_V61_R2_OID								X	X	X	X	X	X
LDAP_CAP_ACTIVE_DIRECTORY_W8_OID										X	X	X	X

* These capabilities are only exposed by the server in certain conditions. For each of these conditional capabilities, the section describing the capability describes the conditions that apply.

3.1.1.3.4.3.1 LDAP_CAP_ACTIVE_DIRECTORY_OID

The presence of this capability indicates that the LDAP server is running Active Directory and is running as AD DS.

3.1.1.3.4.3.2 LDAP_CAP_ACTIVE_DIRECTORY_LDAP_INTEG_OID

The presence of this capability indicates that the LDAP server on the DC is capable of signing and sealing on an NTLM authenticated connection, and that the server is capable of performing subsequent binds on a signed or sealed connection.

3.1.1.3.4.3.3 LDAP_CAP_ACTIVE_DIRECTORY_V51_OID

On an Active Directory DC operating as AD DS, the presence of this capability indicates that the LDAP server is running at least the Windows Server 2003 operating system version of Active Directory.

On an Active Directory DC operating as AD LDS, the presence of this capability indicates that the LDAP server is running at least the Windows Server 2008 operating system version of Active Directory.

3.1.1.3.4.3.4 LDAP_CAP_ACTIVE_DIRECTORY_ADAM_DIGEST

On a DC operating as AD LDS, the presence of this capability indicates that the DC accepts DIGEST-MD5 binds for AD LDS security principals (section 5.1.1.5). An AD LDS DC's DIGEST-MD5 bind functionality depends upon the value of the ADAMDisableSSI configurable setting as specified in section 3.1.1.3.4.7.

3.1.1.3.4.3.5 LDAP_CAP_ACTIVE_DIRECTORY_ADAM_OID

The presence of this capability indicates that the LDAP server is running Active Directory as AD LDS.

3.1.1.3.4.3.6 LDAP_CAP_ACTIVE_DIRECTORY_PARTIAL_SECRETS_OID

On an Active Directory DC operating as AD DS, the presence of this capability indicates that the DC is an RODC.

3.1.1.3.4.3.7 LDAP_CAP_ACTIVE_DIRECTORY_V60_OID

The presence of this capability indicates that the LDAP server is running at least the Windows Server 2008 operating system version of Active Directory.

3.1.1.3.4.3.8 LDAP_CAP_ACTIVE_DIRECTORY_V61_R2_OID

The presence of this capability indicates that the LDAP server is running at least the Windows Server 2008 R2 operating system version of Active Directory.

3.1.1.3.4.3.9 LDAP_CAP_ACTIVE_DIRECTORY_W8_OID

The presence of this capability indicates that the LDAP server is running at least the Windows Server 2012 operating system version of Active Directory.

3.1.1.3.4.4 LDAP Matching Rules (extensibleMatch)

The following sections describe the matching rules supported by DCs when performing LDAP search requests. Unlike, for example, extended controls and extended operations, there is no attribute exposed by the DC that specifies which matching rules it supports. The identifiers for these matching rules are used in an extensibleMatch clause in the Filter portion of a SearchRequest, as described in [RFC2251] section 4.5.1. Matching rules are identified by an OID that corresponds to a human-readable name, as shown in the following table.

Capability name	OID
LDAP_MATCHING_RULE_BIT_AND	1.2.840.113556.1.4.803
LDAP_MATCHING_RULE_BIT_OR	1.2.840.113556.1.4.804
LDAP_MATCHING_RULE_TRANSITIVE_EVAL	1.2.840.113556.1.4.1941
LDAP_MATCHING_RULE_DN_WITH_DATA	1.2.840.113556.1.4.2253

Windows 2000 operating system, Windows Server 2003 operating system, and Active Directory Application Mode (ADAM) support the LDAP_MATCHING_RULE_BIT_AND and LDAP_MATCHING_RULE_BIT_OR matching rules. Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system support those two rules and the LDAP_MATCHING_RULE_TRANSITIVE_EVAL rule, in both AD DS and AD LDS. Windows Server 2012 R2 supports those three rules and the LDAP_MATCHING_RULE_DN_WITH_DATA rule, in both AD DS and AD LDS.

3.1.1.3.4.4.1 LDAP_MATCHING_RULE_BIT_AND

This rule is equivalent to a bitwise "AND" operation. When this matching rule is used as a clause in a query filter, the clause is satisfied only if all the bits set to '1' in the value included in the clause correspond to bits set to '1' in the value stored in the directory.

3.1.1.3.4.4.2 LDAP_MATCHING_RULE_BIT_OR

This rule is equivalent to a bitwise "OR" operation. When this matching rule is used as a clause in a query filter, the clause is satisfied only if at least one of the bits set to '1' in the value included in the clause corresponds to a bit set to '1' in the value stored in the directory.

3.1.1.3.4.4.3 LDAP_MATCHING_RULE_TRANSITIVE_EVAL

This rule provides recursive search of a link attribute. A filter F of the form "(A: 1.2.840.113556.1.4.1941:=V)", where A is a link attribute and V is a value, evaluates to True for an object whose DN is D if the following method EvalTransitiveFilter(A, V, D) returns true, and False if the method returns false. If A is not a link attribute, the filter F evaluates to Undefined.

EvalTransitiveFilter(A: attribute, V: value, D: DN)

- If A is of Object(DN-String), Object(DN-Binary), Object(OR-Name), or Object(Access-Point) syntax, let V' equal the **object_DN** portion of V. Otherwise, let V' equal V.
- Return the value of EvalTransitiveFilterHelper(A, V', D, {})

EvalTransitiveFilterHelper(A: attribute, V': value, ToVisit: DN, Visited: SET OF DN)

- If A is of Object(DN-String), Object(DN-Binary), Object(OR-Name), or Object(Access-Point) syntax, let C be the set of the **object_DN** components of the values of ToVisit.A. Otherwise, let C be the set of the values of ToVisit.A. Note that C is a set of DNs.
- If V' is in C, return true.
- Let Visited' equal the Visited set plus {ToVisit}.
- For each DN NextToVisit in C

- If NextToVisit is in Visited, do nothing.
- Let Result = EvalTransitiveFilterHelper(A, V', NextToVisit, Visited')
- If Result is true, return true.

- Return false.

3.1.1.3.4.4.4 LDAP_MATCHING_RULE_DN_WITH_DATA

This rule provides a way to match on portions of values of syntax Object(DN-String) and Object(DN-Binary).

Let F be a filter of the form "(A: 1.2.840.113556.1.4.2253:=V)", where A is a link attribute and V is a value of syntax Object(DN-String) (section 3.1.1.2.2.2.1) or Object(DN-Binary) (section 3.1.1.2.2.2.3). This filter evaluates to True for an object whose DN is D if the method defined below, EvalDNWithDataFilter(A,V,D), returns true, and False if the method returns false. If A is not of syntax Object(DN-String) or Object(DN-Binary), the filter F evaluates to Undefined.

EvalDNWithDataFilter(A: attribute, V: value, D: DN)

- For either syntax, let O be the DN portion of the value V and B be the string or binary portion of the value V. If the attribute is of syntax Object(DN-String), B is the value of the string considered strictly as the sequence of bytes of the string. Note that O can be the rootDSE. Note also that B can have 0 length.
- For every V' where V' is a value of attribute A on object D:
  - Let O' be the DN portion of value V' and let B' be the string or binary portion of the value V'.
  - If O is not equal to O' and O is not equal to the rootDSE, continue processing other values of V'.
  - If B is not equal to the initial bytes of B', continue processing other values of V'. Note especially that only byte values are used in this comparison. No special handling of B as a string is performed (for example, no case-insensitivity, locale specific comparisons, etc.).
  - Return true.
- If this method does not return true, it returns false.

3.1.1.3.4.5 LDAP SASL Mechanisms

The following sections describe the SASL mechanisms that are implemented by DCs in Windows 2000 operating system, Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system. SASL is described in [RFC2222], and the usage of SASL and other authentication methods in LDAP is described in [RFC2829]. The SASL mechanisms supported by a DC are exposed as strings in the supportedSASLMechanisms attribute of the rootDSE.

Not all versions of Windows Server operating system and Active Directory Application Mode (ADAM) support all the LDAP SASL mechanisms. The following table indicates which SASL mechanisms are supported in which version.

Mechanism name	Windows 2000	Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2	AD LDS
GSSAPI	X	X	X
GSS-SPNEGO	X	X	X
EXTERNAL		X	X
DIGEST-MD5		X	X

Additional details of LDAP authentication in Active Directory are in section 5.1.

3.1.1.3.4.5.1 GSSAPI

The presence of the "GSSAPI" string value in the supportedSASLMechanisms attribute indicates that the DC accepts the GSSAPI security mechanism for LDAP bind requests. The GSSAPI mechanism for SASL is described in [RFC2222] section 7.2, and GSSAPI is described in more detail in [RFC2078]. Active Directory supports Kerberos when using GSSAPI; see [MS-KILE] and [RFC1964] for details of Kerberos.

3.1.1.3.4.5.2 GSS-SPNEGO

The presence of the "GSS-SPNEGO" string value in the supportedSASLMechanisms attribute indicates that the DC accepts the GSS-SPNEGO security mechanism for LDAP bind requests. This mechanism is documented in [RFC4178]. Active Directory supports Kerberos (see [MS-KILE]) and NTLM (see [MS-NLMP]) when using GSS-SPNEGO.

3.1.1.3.4.5.3 EXTERNAL

The presence of the "EXTERNAL" string value in the **supportedSASLMechanisms** attribute indicates that the DC accepts external security mechanisms for LDAP bind requests. The EXTERNAL SASL mechanism is described in [RFC2222] section 7.4, and [RFC2829]. In the case of DCs, the external authentication information that is used to validate the identity of the client making the bind request comes from the client certificate presented by the client during the **SSL/TLS handshake** that occurs in response to the client sending an LDAP_SERVER_START_TLS_OID extended operation. When the server receives an EXTERNAL SASL bind following a successful LDAP_SERVER_START_TLS_OID extended operation in which a valid certificate was presented by the client, the server causes the connection to be bound as the identity represented by that certificate.

3.1.1.3.4.5.4 DIGEST-MD5

The presence of the "DIGEST-MD5" string value in the supportedSASLMechanisms attribute indicates that the DC accepts the **digest** security mechanism for LDAP bind requests. The usage of digest authentication with LDAP is documented in [RFC2829] section 6.1, and in [RFC2831].

3.1.1.3.4.6 LDAP Policies

The DC's LDAP interface supports various policies that can be configured by an administrator. The names of these policies are listed on the supportedLDAPPolicies attribute on the rootDSE. These policies are listed in the following table, which also lists which versions of Windows and Active Directory Application Mode (ADAM) support which policies.

Policy name	Windows 2000 operating system	Windows Server 2003 operating system, ADAM, and Windows Server 2008 operating system	Windows Server 2008 R2 operating system	Windows Server 2012 operating system and Windows Server 2012 R2 operating system
MaxActiveQueries	X*			
InitRecvTimeout	X	X	X	X
MaxConnections	X	X	X	X
MaxConnIdleTime	X	X	X	X
MaxDatagramRecv	X	X	X	X
MaxNotificationPerConn	X	X	X	X
MaxPoolThreads	X	X	X	X
MaxReceiveBuffer	X	X	X	X
MaxPageSize	X	X	X	X
MaxQueryDuration	X	X	X	X
MaxResultSetSize	X	X	X	X
MaxTempTableSize	X	X	X	X
MaxValRange		X	X	X
MaxResultSetsPerConn			X	X
MinResultSets			X	X
MaxBatchReturnMessages				X

* Support for this policy was removed in Windows Server 2003.

LDAP policies are specified using the lDAPAdminLimits attribute. The lDAPAdminLimits attribute of a queryPolicy object is a multivalued string where each string value encodes a name-value pair. In the encoding, the name and value are separated by an "=". For example, the encoding of the name "MaxActiveQueries" with value "0" is "MaxActiveQueries=0". Each name is the name of an LDAP policy, and the value is a value of that policy.

There can be multiple queryPolicy objects in a forest. A DC determines the queryPolicy object that contains its policies according to the following logic:

- If the queryPolicyObject attribute is present on the DC's nTDSDSA object, the DC uses the queryPolicy object referenced by it.
- Otherwise, if the queryPolicyObject attribute is present on the nTDSSiteSettings object for the **site** to which the DC belongs, the DC uses the queryPolicy object referenced by it.

- Otherwise, the DC uses the queryPolicy object whose DN is "CN=Default Query Policy,CN=Query-Policies" relative to the nTDSService object (for example, "CN=Default Query Policy, CN=Query-Policies, CN=Directory Service, CN=Windows NT, CN=Services" relative to the root of the config NC).

The effect of setting an LDAP policy is outside the state model. The effect of each policy, as well as the default value used if the policy's value is not specified in an lDAPAdminLimits attribute, is shown in the following table.

Policy name	Default value	Description
MaxActiveQueries	20	The maximum number of concurrent LDAP search operations that are permitted to run at the same time on a DC. When this limit is reached, the DC returns a *busy* / *ERROR_DS_ADMIN_LIMIT_EXCEEDED* error.
InitRecvTimeout	120	The maximum time, in seconds, that a DC waits for the client to send the first request after the DC receives a new connection. If the client does not send the first request in this amount of time, the server disconnects the client.
MaxConnections	5000	The maximum number of simultaneous LDAP connections that a DC will accept. If a connection comes in after the DC reaches this limit, the DC will drop another connection. The connection that is selected to drop is not constrained by the protocol and is determined based on the implementation.
MaxConnIdleTime	900	The maximum time, in seconds, that the client can be idle before the DC closes the connection. If a connection is idle for more than this time, the DC disconnects the client.
MaxDatagramRecv	4096	The maximum size, in bytes, of a UDP datagram request that a DC will process. Requests that are larger than this value are ignored by the DC.
MaxNotificationPerConn	5	The maximum number of outstanding notification search requests (using the LDAP_SERVER_NOTIFICATION_OID control) that the DC permits on a single connection. When this limit is reached, the server returns an *adminLimitExceeded* / *ERROR_DS_ADMIN_LIMIT_EXCEEDED* error to any new notification searches that are requested on that connection.
MaxPoolThreads	4	The maximum number of threads per processor that a DC dedicates to listening for network input or output. This value also determines the maximum number of threads per processor that can work on LDAP requests at the same time.
MaxReceiveBuffer	10,485,760	The maximum size, in bytes, of a request that the server will accept. If the server receives a request that is larger than this, it will drop the connection.
MaxPageSize	1000	The maximum number of objects that are returned in a single search result, independent of how large each returned object is. To perform a search where the result might exceed this number of objects, the client must specify the paged search control.
MaxQueryDuration	120	The maximum time, in seconds, that a DC will spend on a

Policy name	Default value	Description
		single search or batched LDAP extended operation (in Windows Server 2012 and Windows Server 2012 R2). When this limit is reached, the DC returns a *timeLimitExceeded* / *ERROR_INVALID_PARAMETER* error.
MaxResultSetSize	262,144	The maximum number of bytes that a DC stores to optimize the individual searches that make up a paged search. The data that is stored is outside the state model and is implementation-specific.
MaxTempTableSize	10,000	The maximum number of rows that a DC will create in a temporary database table to hold intermediate results during query processing.
MaxValRange	1500	The maximum number of values that can be retrieved from a multivalued attribute in a single search request. Windows 2000 DCs do not support this policy and instead always use a setting of 1000 values.
MaxResultSetsPerConn	10	The maximum number of individual paged searches per LDAP connection for which a DC will store optimization data. The data that is stored is outside the state model and is implementation-specific.
MinResultSets	3	The minimum number of individual paged searches for which a DC will store optimization data. The data that is stored is outside the state model and is implementation-specific.
MaxBatchReturnMessages	1100	The maximum number of messages that can be returned when processing an LDAP_SERVER_BATCH_REQUEST_OID extended operation (section 3.1.1.3.4.2.5).

3.1.1.3.4.7 LDAP Configurable Settings

A forest supports several administrator-controlled settings that affect LDAP. The name of each setting is included in the supportedConfigurableSettings attribute on the rootDSE. These settings are listed in the following table. The table also lists which versions of Windows Server operating system and Active Directory Application Mode (ADAM) support which settings. The settings are stored on the msDS-Other-Settings attribute of the **directory service** object, as specified in section 6.1.1.2.4.1.1. For more information, see [ADDLG].

Setting name	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2003 operating system with Service Pack 1 (SP1)	ADAMRTW	ADAMSP1	Windows Server 2008 operating system ADDS	Windows Server 2008 ADLDS	Windows Server 2008 R2 operating system ADDS	Windows Server 2008 R2 ADLDS	Windows Server 2012 operating system ADDS	Windows Server 2012 ADLDS	Windows Server 2012 R2 operating system ADDS	Windows Server 2012 R2 ADLDS
DynamicObjectDefaultTTL		X	X	X	X	X	X	X	X	X	X	X	X
DynamicObjectMinTTL		X	X	X	X	X	X	X	X	X	X	X	X
DisableVLVSupport			X		X	X	X	X	X	X	X	X	X
ADAMAllowADAMSecurityPrincipalsInConfigPartition					X		X		X		X		X
ADAMDisableLogonAuditing				X	X		X		X		X		X
ADAMDisablePasswordPolicies				X	X		X		X		X		X
ADAMDisableSPNRegistration					X		X		X		X		X
ADAMDisableSSI					X		X		X		X		X
ADAMLastLogonTimestampWindow				X	X		X		X		X		X
MaxReferrals				X	X	X	X	X	X	X	X	X	X
ReferralRefreshInterval				X	X	X	X	X	X	X	X	X	X
RequireSecureP				X	X		X		X		X		X

Setting name	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2003 operating system with Service Pack 1 (SP1)	ADAM RTW	ADAM SP1	Windows Server 2008 operating system ADDS	Windows Server 2008 ADLDS	Windows Server 2008 R2 operating system ADDS	Windows Server 2008 R2 ADLDS	Windows Server 2012 operating system ADDS	Windows Server 2012 ADLDS	Windows Server 2012 R2 operating system ADDS	Windows Server 2012 R2 ADLDS
roxyBind													
RequireSecureSimpleBind				X	X		X		X		X		X
SelfReferralsOnly				X	X	X	X	X	X	X	X	X	X

The DynamicObjectDefaultTTL is the default entryTTL value for a new dynamic object. The value is in seconds and defaults to 86400. The minimum value is 1 and the maximum value is 31557600 (one year).

The DynamicObjectMinTTL is the minimum valid entryTTL value for a new dynamic object. The value is in seconds and defaults to 900. The minimum value is 1 and the maximum value is 31557600 (one year).

When the DisableVLVSupport setting is set to 1, the DC excludes the OIDs for the LDAP_CONTROL_VLVREQUEST and LDAP_CONTROL_VLVRESPONSE controls from the supportedControl attribute of the rootDSE. Additionally, if the LDAP_CONTROL_VLVREQUEST control is attached to an incoming LDAP request and is not marked as critical, the DC ignores the control. If the control is attached to an incoming LDAP request and is marked critical, the DC fails the request with the error *unavailableCriticalExtension* / *ERROR_INVALID_PARAMETER*. If the DisableVLVSupport setting is not specified, it defaults to 0.

When ADAMAllowADAMSecurityPrincipalsInConfigPartition equals 1, security principals (that is, objects that have an objectSid attribute) may be created in the Config NC. When equal to 0, attempts to create a security principal in the Config NC are rejected with the error *unwillingToPerform* / *ERROR_DS_CANT_CREATE_IN_NONDOMAIN_NC*. If ADAMAllowADAMSecurityPrincipalsInConfigPartition is not specified, it defaults to 0.

The effect of ADAMDisableLogonAuditing is outside the state model. When ADAMDisableLogonAuditing equals 1, the DC does not generate audit events when an AD LDS security principal (section 5.1.1.5) authenticates to the server. If set to 0, the DC attempts to generate audit events when an AD LDS security principal authenticates to the server; policy on the

computer running the DC determines whether audit events are actually generated. If ADAMDisableLogonAuditing is not specified, it defaults to 0.

When ADAMDisablePasswordPolicies does not equal 1 and an LDAP bind is performed or a password is changed on an AD LDS security principal, the DC enforces the current password policy in effect on the AD LDS server as reported by SamrValidatePassword ([MS-SAMR] section 3.1.5.13.7). When ADAMDisablePasswordPolicies is set to 1, the DC does not enforce any such policies. If ADAMDisablePasswordPolicies is not explicitly specified, it defaults to 0.

When ADAMDisableSPNRegistration equals 1, a DC running as AD LDS does not register its SPNs on the servicePrincipalName of the computer object as described in [MS-DRSR] section 2.2.2. When ADAMDisableSPNRegistration equals 0, a DC running as AD LDS performs SPN registration as described in that document. If ADAMDisableSPNRegistration is not explicitly specified, it defaults to 0.

When ADAMDisableSSI equals 1, a DC running as AD LDS does not support DIGEST-MD5 authentication for AD LDS security principals. If ADAMDisableSSI equals 0, a DC running as AD LDS supports DIGEST-MD5 for AD LDS security principals. ADAMDisableSSI has no effect on a DC running as AD DS. If ADAMDisableSSI is not explicitly specified, it defaults to 0.

ADAMLastLogonTimestampWindow specifies how frequently, in days, AD LDS updates the lastLogonTimestamp attribute when an AD LDS security principal (see section 5.1.1.5) authenticates to the server. For an AD LDS security principal O, if a successful LDAP bind as that security principal is performed at time T, and the difference between O!lastLogonTimestamp and T is greater than ADAMLastLogonTimestampWindow days, then the AD LDS DC sets O!lastLogonTimestamp to T. Otherwise, the AD LDS DC leaves O!lastLogonTimestamp unchanged. If ADAMLastLogonTimestampWindow is not explicitly specified, it defaults to 7.

MaxReferrals specifies the maximum number of LDAP URLs that the DC will include in a referral or continuation reference. The default value is 3.

The effect of ReferralRefreshInterval is outside the state model. A Windows DC maintains an in-memory cache of referral information so that it can return referrals and continuation references without consulting the directory state. ReferralRefreshInterval specifies how frequently, in minutes, a DC refreshes the in-memory cache from the directory state. The default value is 5.

When RequireSecureProxyBind is set to 1, AD LDS will reject (with the error *confidentialityRequired* / <*unrestricted*>) an LDAP simple bind request that requests authentication as an AD LDS bind proxy (section 5.1.1.5) if that request is not performed on an SSL/TLS-encrypted or SASL-protected connection with a cipher strength of at least 128 bits. If RequireSecureProxyBind is set to 0, no such restriction is imposed. If RequireSecureProxyBind is not explicitly specified, it defaults to 1.

When RequireSecureSimpleBind is set to 1, AD LDS will reject (with the error *confidentialityRequired* / <*unrestricted*>) an LDAP simple bind request that requests authentication as an AD LDS security principal (section 5.1.1.5) if that request is not performed on an SSL/TLS-encrypted or SASL-protected connection with a cipher strength of at least 128 bits. If RequireSecureSimpleBind is set to 0, no such restriction is imposed. If RequireSecureSimpleBind is not explicitly specified, it defaults to 0.

If SelfReferralsOnly is set to 1, then the DC will only return referrals and continuation references that refer to itself. It will not return referrals and continuation references to NCs of which it does not have an NC replica. Referrals and continuation references to NCs of which it does have an NC replica will name itself as the referred-to server.

3.1.1.3.4.8 LDAP IP-Deny List

The IP Deny list specifies a set of IP addresses from which the DC will reject incoming LDAP connection requests. The IP Deny list is stored in the lDAPIPDenyList attribute on the queryPolicy object. The DC retrieves the lDAPIPDenyList attribute from the same queryPolicy object that it retrieves the lDAPAdminLimits attribute from in section 3.1.1.3.4.6

The lDAPIPDenyList attribute is a multivalued attribute. Each value of the attribute is a string in the following form:

X.X.X.X M.M.M.M

where **X.X.X.X** is an IP address and **M.M.M.M** is a network mask. A connection from an IP address Y.Y.Y.Y will be rejected if the bitwise AND of Y.Y.Y.Y and **M.M.M.M** equals **X.X.X.X.**

For example, the value "157.59.132.0 255.255.255.0" would cause requests from IP addresses 157.59.132.0 through 157.59.132.255 to be rejected. The value "157.59.132.245 255.255.255.255" would reject only IP address 157.59.132.245.

The IP Deny list is only supported on IPv4 connections. Active Directory does not support this mechanism on IPv6 connections.

3.1.1.4 Reads

References:

- [RFC2251]
- Special Objects and Forest Requirements: section 6.1
- [MS-DRSR]
- [XMLSCHEMA2/2]
- Quota Calculation: section 3.1.1.5
- Range Retrieval of Attribute Values: section 3.1.1.3
- Referrals in LDAPv2 and LDAPv3: section 3.1.1.3
- [MS-ADSC]
- [MS-ADA1]
- [MS-ADA2]
- [MS-ADA3]
- Function GetWellknownObject: section 3.1.1.1

3.1.1.4.1 Introduction

LDAP reads are specified in [RFC2251] section 4.5. Generally and imprecisely, reads are searches starting at some object in Active Directory and restricted by the requester to either the object, the object's children, or the tree of objects rooted by object. After applying that restriction, the search is then restricted to the objects and the values for attributes on those objects to which the requester has access. The search is finally restricted to the objects that match the search filter. The requested attributes and their values for those matching objects are then returned to the requester. The RFC

specifies the details for LDAP reads. This section covers access checks for LDAP reads, extended access checks for reading the specified attributes, the attributes used to construct the specified constructed attributes, and the effect of defunct attributes and classes on reads.

This section does not provide details on the classes and attributes mentioned here. For details, see [MS-ADSC], [MS-ADA1], [MS-ADA2], and [MS-ADA3].

3.1.1.4.2 Definitions

The following functions are used to specify the behavior of several of the constructed attributes. They are collected together here because of the dependencies they have on each other.

Let SUPCLASSES (top) be the empty set. For other classes O, let SUPCLASSES(O) be the union of O!subClassOf and SUPCLASSES(O!subClassOf).

Let AUXCLASSES(O) be the union of

O!systemAuxiliaryClass

and O!auxiliaryClass

and AUXCLASSES(O!systemAuxiliaryClass)

and AUXCLASSES(O!auxiliaryClass)

and AUXCLASSES(C) for all C in SUPCLASSES(O)

Let SUBCLASSES(O) be the set of all C such that O is in SUPCLASSES(C).

Let POSSSUP_NOSUBCLASSES(O) be the union of

O!systemPossSuperiors

and O!possSuperiors

and POSSSUP_NOSUBCLASSES(C) for all C in SUPCLASSES(O)

Let POSSSUPERIORS(O) be the union of

POSSSUP_NOSUBCLASSES(O)

and SUBCLASSES(C) for all C in POSSSUP_NOSUBCLASSES(O)

Let CLASSATTS(O) be the union of

O!mustContain

and O!systemMustContain

and O!mayContain

and O!systemMayContain

and CLASSATTS(C) for all C in SUPCLASSES(O)

and CLASSATTS(C) for all C in AUXCLASSES(O)

Let SPC(O) be true when O or any SUPCLASSES(O) is one of builtinDomain, samServer, samDomain, group, or user; and false, otherwise.

3.1.1.4.3 Access Checks

An object is not visible to a requester if the requester is not granted the necessary rights. But even if an object is visible to a requester, the requester may lack the necessary rights to see individual attributes. The values for attributes that are not visible to the requester are treated as "does not exist" in the returned attributes and the LDAP filter. For example, if the requester requests the value for displayName but that attribute is not visible, then the returned value will be the same as it would have been if the attribute displayName did not exist on that Object. Likewise, if displayName were part of the LDAP filter, then, similarly, the filter would behave just as if displayName did not exist on that Object.

Let O be the Object being considered during search.

Let ON be the root object of the NC containing O.

Let OP be O!parent.

Let OA be the Attribute, or the property set containing the Attribute, that is being considered for O during search.

Generally, the security context of the requester must be granted rights RIGHT_DS_LIST_CONTENTS (defined in section 5.1.3.2) on OP by OP!nTSecurityDescriptor.

Generally, the security context of the requester must be granted rights RIGHT_DS_READ_PROPERTY on OA by O!nTSecurityDescriptor. Otherwise, the value is treated as "does not exist" in the returned attributes and the LDAP filter. This behavior changes for special attributes, for attributes with special search flags in their definition, and for some attributes because of dSHeuristics (section 6.1.1.2.4.1.2), as specified in section 3.1.1.4.4.

3.1.1.4.4 Extended Access Checks

Some attributes require different access than that specified in the previous section.

The security context of the requester must be granted the indicated rights on OA by O!nTSecurityDescriptor unless otherwise specified. If not granted, then the value is treated as "does not exist" in the returned attributes and the LDAP filter.

OA	Requires right(s)
nTSecurityDescriptor	(ACCESS_SYSTEM_SECURITY) and (RIGHT_READ_CONTROL)
msDS-QuotaEffective	(RIGHT_DS_READ_PROPERTY on the Quotas container, described in section 6.1.1.4.3) or ((the client is querying the quota for the security principal it is authenticated as) and (DS-Query-Self-Quota control access right on the Quotas container))
msDS-QuotaUsed	(RIGHT_DS_READ_PROPERTY on the Quotas container, described in section 6.1.1.4.3) or ((the client is querying the quota for the security principal it is authenticated as) and (DS-Query-Self-Quota control access right on the Quotas container))

OA	Requires right(s)
userPassword	When the fUserPwdSupport heuristic in the dSHeuristics attribute (see section 6.1.1.2.4.1.2) is false, the requester must be granted RIGHT_DS_READ_PROPERTY. When fUserPwdSupport is true, access is never granted.
pekList	Access is never granted
currentValue	Access is never granted
dBCSPwd	Access is never granted
unicodePwd	Access is never granted
ntPwdHistory	Access is never granted
priorValue	Access is never granted
supplementalCredentials	Access is never granted
trustAuthIncoming	Access is never granted
trustAuthOutgoing	Access is never granted
lmPwdHistory	Access is never granted
initialAuthIncoming	Access is never granted
initialAuthOutgoing	Access is never granted
msDS-ExecuteScriptPassword	Access is never granted
Attribute whose attributeSchema has CF (fCONFIDENTIAL, 0x0x00000080) set in searchFlags.	(RIGHT_DS_READ_PROPERTY) and (RIGHT_DS_CONTROL_ACCESS)
sDRightsEffective	See section 3.1.1.4.5.4
allowedChildClassesEffective	See section 3.1.1.4.5.5
allowedAttributesEffective	See section 3.1.1.4.5.7
msDS-Approx-Immed-Subordinates	See section 3.1.1.4.5.15
msDS-QuotaEffective	See section 3.1.1.4.5.22
msDS-ReplAttributeMetaData msDS-ReplValueMetaData	The security context of the requester must be granted the following rights on the replPropertyMetaData attribute: (RIGHT_DS_READ_PROPERTY) or (DS-Replication-Manage-Topology by ON!nTSecurityDescriptor)
msDS-NCReplInboundNeighbors	The security context of the requester must be granted the following rights on repsFrom: (RIGHT_DS_READ_PROPERTY) or (DS-Replication-Manage-Topology) or (DS-Replication-Monitor-Topology)

OA	Requires right(s)
msDS-NCReplOutboundNeighbors	The security context of the requester must be granted the following rights on repsTo: (RIGHT_DS_READ_PROPERTY) or (DS-Replication-Manage-Topology) or (DS-Replication-Monitor-Topology)
msDS-NCReplCursors	The security context of the requester must be granted the following rights on replUpToDateVector: (RIGHT_DS_READ_PROPERTY) or (DS-Replication-Manage-Topology) or (DS-Replication-Monitor-Topology)
msDS-IsUserCachableAtRodc	The security context of the requester must be granted the Read-Only-Replication-Secret-Synchronization control access right on the root of the default NC.
msDS-ManagedPassword	The security context of the requester must be granted the RIGHT_DS_READ_PROPERTY control access right on the security descriptor in the msDS-GroupMSAMembership attribute.
Attribute whose attributeSchema has SE (fPARTITIONSECRET, 0x0x00001000) set in searchFlags.	(RIGHT_DS_READ_PROPERTY) must be granted on the object, and the DS-Read-Partition-Secrets control access right must be granted on the object that is the root of the naming context to which the object belongs.

3.1.1.4.5 Constructed Attributes

Individual constructed attributes, other than rootDSE Attributes (section 3.1.1.3.2), are specified in [MS-ADA1], [MS-ADA2], and [MS-ADA3]. But briefly, constructed attributes have the property that they are attributes for which the attribute value is computed by using other attributes, sometimes from other objects. Regardless of this property, constructed attributes are defined to be those attributes that meet one of the following three criteria:

- The attributeSchema object's systemFlags attribute has the ATTR_IS_CONSTRUCTED bit (section 2.2.10) set to one.
- The attribute is a rootDSE attribute (section 3.1.1.3.2).
- The attribute is a back link attribute.

The objects and attributes for specified constructed attributes are covered in this section.

Except as otherwise noted, these constructed attributes are applicable to both AD DS and AD LDS.

3.1.1.4.5.1 subSchemaSubEntry

The value is the DN equal to the schema NC's DN appended to "CN=Aggregate,".

3.1.1.4.5.2 canonicalName

The value is the canonical name of the object (section 3.1.1.1.7).

3.1.1.4.5.3 allowedChildClasses

Let TO be the object from which the allowedChildClasses attribute is being read.

The value of TO!allowedChildClasses the set of lDAPDisplayNames read from each Object O where

- (O.distinguishedName is in the schema NC)
- and (O!objectClass is classSchema)
- and (not O!systemOnly)
- and (not O!objectClassCategory is 2)
- and (not O!objectClassCategory is 3)
- and (there exists C in TO!objectClass such that C is in POSSSUPERIORS(O))

3.1.1.4.5.4 sDRightsEffective

Let TO be the object from which the sDRightsEffective attribute is being read.

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
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SSI	DSI	GSI	OSI

Note Bits are presented in big-endian byte order.

The value of TO!sDRightsEffective is derived as follows from the bits shown in the preceding table:

- OSI (OWNER_SECURITY_INFORMATION, 0x00000001) and GSI (GROUP_SECURITY_INFORMATION, 0x00000002) are both set if TO!nTSecurityDescriptor grants RIGHT_WRITE_OWNER to the requester.
- DSI (DACL_SECURITY_INFORMATION, 0x00000004) is set if TO!nTSecurityDescriptor grants RIGHT_WRITE_DAC to the requester.
- SSI (SACL_SECURITY_INFORMATION, 0x00000008) is set if TO!nTSecurityDescriptor grants RIGHT_ACCESS_SYSTEM_SECURITY to the requester.

3.1.1.4.5.5 allowedChildClassesEffective

The allowedChildClassesEffective attribute has different behavior on AD DS and AD LDS.

If the DC is running as AD LDS, then let fAllowPrincipals equal true if the value of the ADAMAllowADAMSecurityPrincipalsInConfigPartition configuration setting (section 3.1.1.3.4.7) is 1, false otherwise. If the ADAMAllowADAMSecurityPrincipalsInConfigPartition configuration setting is not supported, then let fAllowPrincipals = false.

Let TO be the object from which the allowedChildClassesEffective attribute is being read.

TO!allowedChildClassesEffective contains each object class O in TO!allowedChildClasses such that:

- (

  (TO!nTSecurityDescriptor grants RIGHT_DS_CREATE_CHILD via a simple access control entry (ACE) to the client for instantiating an object beneath TO)

  or

  (TO.nTSecurityDescriptor grants RIGHT_DS_CREATE_CHILD via an object-specific ACE to the client for instantiating an object of class O beneath TO)

)

- and (fAllowPrincipals or (not TO!distinguishedName in config NC) or (not SPC(O)))
- and (fAllowPrincipals or (not TO!distinguishedName in schema NC) or (not SPC(O)))

Simple ACEs and object-specific ACEs are discussed in section 5.1.3.

3.1.1.4.5.6 allowedAttributes

Let TO be the object from which the allowedAttributes attribute is being read.

The value of TO!allowedAttributes is the set of lDAPDisplayNames read from each Object O where:

- (O.dn is in the schema NC)
- and (O!objectClass is attributeSchema)
- and (there exists C in TO!objectClass such that O is in CLASSATTS(C))

3.1.1.4.5.7 allowedAttributesEffective

Let TO be the object from which the allowedAttributesEffective attribute is being read.

The value of TO!allowedAttributesEffective is the subset of values returned by allowedAttributes for which values (O) conform to the following:

- TO!nTSecurityDescriptor grants RIGHT_DS_WRITE_PROPERTY on O to the requester
- and (O!linkID is even or O!linkID is not present)
- (and (not bit 0x4 is set in O!systemFlags) or O!lDAPDisplayName is entryTTL)

3.1.1.4.5.8 fromEntry

Let TO be the object from which the fromEntry attribute is being read.

The value of TO!fromEntry is true if TO!instanceType has bit 0x4 set, otherwise false.

3.1.1.4.5.9 createTimeStamp

Let TO be the object from which the createTimeStamp attribute is being read.

The value of TO!createTimeStamp is TO!whenCreated.

3.1.1.4.5.10 modifyTimeStamp

Let TO be the object from which the modifyTimeStamp attribute is being read.

The value of TO!modifyTimeStamp is TO!whenChanged.

3.1.1.4.5.11 primaryGroupToken

Let TO be the object from which the primaryGroupToken attribute is being read.

The value of TO!primaryGroupToken is the RID from TO!objectSid when there exists C in TO!objectClass such that C is the group class. Otherwise, no value is returned. That is, if TO is a group, then the value of this attribute is the RID from the group's SID. If TO is not a group, no value is returned when this attribute is read from TO.

3.1.1.4.5.12 entryTTL

Let TO be the object from which the entryTTL attribute is being read.

The value of TO!entryTTL is the number of seconds in TO!msDS-Entry-Time-To-Die minus the current system time, and is constrained to the range 0..0xFFFFFFFF by returning 0 if the difference is less than 0, and 0xFFFFFFFF if the difference is greater than 0xFFFFFFFF.

3.1.1.4.5.13 msDS-NCReplInboundNeighbors, msDS-NCReplCursors, msDS-ReplAttributeMetaData, msDS-ReplValueMetaData

If the object from which msDS-NCReplInboundNeighbors or msDS-NCReplCursors is being read is not the root object of an NC, the result of the read is no value.

Otherwise, reading any of these four attributes on an object returns an alternate representation of the structures returned by IDL_DRSGetReplInfo() applied to that object. The result is either a binary data structure or XML (IDL_DRSGetReplInfo and its associated structures are documented in [MS-DRSR] section 4.1.13). The relationship between these constructed attributes and the IDL_DRSGetReplInfo data is shown in the following table.

Constructed attribute	Equivalent DS_REPL_INFO code*	XML structure**	Binary structure***
msDS-NCReplInboundNeighbors	DS_REPL_INFO_NEIGHBORS	DS_REPL_NEIGHBORW	DS_REPL_NEIGHBORW_BLOB
msDS-NCReplCursors	DS_REPL_INFO_CURSORS_3_FOR_NC	DS_REPL_CURSORS_3W	DS_REPL_CURSOR_BLOB
msDS-ReplAttributeMetaData	DS_REPL_INFO_METADATA_2_FOR_OBJ	DS_REPL_ATTR_META_DATA_2	DS_REPL_ATTR_META_DATA_BLOB
msDS-ReplValueMetaData	DS_REPL_INFO_METADATA_2_FOR_ATTR_VALUE	DS_REPL_VALUE_META_DATA_2	DS_REPL_VALUE_META_DATA_BLOB

* See [MS-DRSR] section 4.1.13.1.4.

** See [MS-DRSR] section 4.1.13.1.

*** See section 2.2.

The information returned is exactly the same information as is returned by a call to IDL_DRSGetReplInfo when specifying the value in the second column as the value for DRS_MSG_GETREPLINFO_REQ_V1.InfoType or DRS_MSG_GETREPLINFO_REQ_V2.InfoType.

Without any attribute qualifier, the data is returned as XML. The parent element of the XML is the name of the structure contained in the "XML structure" column in the table, and the child element names and order in the XML exactly follow the names of the fields in that structure as well. The meaning of each child element is the same as the meaning of the corresponding field in the structure. Values of integer types are represented as decimal strings. Values of FILETIME type are represented as XML dateTime values in UTC, for example, "04-07T18:39:09Z", as specified in [XMLSCHEMA2/2]. Values of GUID fields are represented as GUIDStrings.

If the ";binary" attribute qualifier is specified when the attribute is requested, the value of this attribute is returned as binary data; specifically, the structure contained in the "Binary Structure" column. In this representation, fields that would contain strings are represented as integer offsets (relative to the beginning of the binary data) to a null-terminated UTF-16 encoded string embedded in the returned binary data.

3.1.1.4.5.14 msDS-NCReplOutboundNeighbors

The msDS-NCReplOutboundNeighbors attribute is equivalent to msDS-NCReplInboundNeighbors, except that it retrieves representations of each repsTo value for the requested Object (that is, information related to replication notifications for event-driven replication), while msDS-NCReplInboundNeighbors retrieves representations of each repsFrom value (that is, information related to inbound replication). Like msDS-NCReplInboundNeighbors, it can return the data in either XML or binary form, depending on the presence of the ";binary" attribute qualifier, and uses the DS_REPL_NEIGHBOR and DS_REPL_NEIGHBORW_BLOB structures for its XML and binary representations, respectively.

3.1.1.4.5.15 msDS-Approx-Immed-Subordinates

Let TO be the object from which the msDS-Approx-Immed-Subordinates attribute is being read.

The value of TO!msDS-Approx-Immed-Subordinates is the approximate number of direct descendants of this object if TO!nTSecurityDescriptor grants RIGHT_DS_LIST_CONTENTS to the client. This estimate has no guarantee or requirement of accuracy. If the client does not have the RIGHT_DS_LIST_CONTENTS access right, the value 0 is returned as the estimate.

3.1.1.4.5.16 msDS-KeyVersionNumber

The msDS-KeyVersionNumber attribute exists on AD DS but not on AD LDS.

Let TO be the object from which the msDS-KeyVersionNumber attribute is being read.

If the fKVNOEmuW2k heuristic of the dSHeuristics attribute (see section 6.1.1.2.4.1.2) is true, TO!msDS-KeyVersionNumber equals 1. Otherwise, TO!msDS-KeyVersionNumber equals the dwVersion field of the AttributeStamp associated with TO's unicodePwd attribute. See section 3.1.1.1.9 for more information about AttributeStamp and dwVersion.

3.1.1.4.5.17 msDS-User-Account-Control-Computed

The msDS-User-Account-Control-Computed attribute has different behavior on AD DS and AD LDS.

Let TO be the object from which the msDS-User-Account-Control-Computed attribute is being read.

For AD DS, the following description applies.

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

Note Bits are presented in big-endian byte order.

If the object TO is not in a domain NC, TO!msDS-User-Account-Control-Computed = 0.

If the object TO is in a domain NC, let D be the root of that NC, and let ST be the current time, read from the system clock. Then the value of TO!msDS-User-Account-Control-Computed is the preceding bit pattern, where:

- LO (ADS_UF_LOCKOUT, 0x00000010) is set if:
  - (none of bits ADS_UF_WORKSTATION_TRUST_ACCOUNT, ADS_UF_SERVER_TRUST_ACCOUNT, ADS_UF_INTERDOMAIN_TRUST_ACCOUNT are set in TO!userAccountControl)
  - and (TO!lockoutTime is nonzero and either (1) Effective-LockoutDuration (regarded as an unsigned quantity) < 0x8000000000000000, or (2) ST + Effective-LockoutDuration (regarded as a signed quantity) ≤ TO!lockoutTime), where Effective-LockoutDuration is defined in [MS-SAMR] section 3.1.1.5.
- PE (ADS_UF_PASSWORD_EXPIRED, 0x00800000) is set if:
  - (none of bits ADS_UF_SMARTCARD_REQUIRED, ADS_UF_DONT_EXPIRE_PASSWD, ADS_UF_WORKSTATION_TRUST_ACCOUNT, ADS_UF_SERVER_TRUST_ACCOUNT, ADS_UF_INTERDOMAIN_TRUST_ACCOUNT are set in TO!userAccountControl)
  - and (TO!pwdLastSet = null, or TO!pwdLastSet = 0, or (Effective-MaximumPasswordAge ≠ 0x8000000000000000 and (ST - TO!pwdLastSet) > Effective-MaximumPasswordAge)), where Effective-MaximumPasswordAge is defined in [MS-SAMR] section 3.1.1.5.

For AD LDS, the following description applies.

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

Note Bits are presented in big-endian byte order.

The value of TO!msDS-User-Account-Control-Computed attribute is the preceding bit pattern, where:

- AD (ADS_UF_ACCOUNT_DISABLE, 0x00000002) is set if:

- TO!msDS-UserAccountDisabled is true

- LO (ADS_UF_LOCKOUT, 0x00000010) is set if:
  - TO!ms-DS-UserAccountAutoLocked is true
- PNR (ADS_UF_PASSWD_NOTREQD, 0x00000020) is set if:
  - TO!ms-DS-UserPasswordNotRequired is true
- DEP (ADS_UF_DONT_EXPIRE_PASSWD, 0x00010000) is set if:
  - TO!msDS-UserDontExpirePassword is true
- PE (ADS_UF_PASSWORD_EXPIRED, 0x00800000) is set if:
  - TO!msDS-UserPasswordExpired is true

3.1.1.4.5.18 msDS-Auxiliary-Classes

Let TO be the object from which the msDS-Auxiliary-Classes attribute is being read.

The value of TO!msDS-Auxiliary-Classes is the set of lDAPDisplayNames from each Object O such that (O is in TO!objectClass) and (O is not in SUPCLASSES(Most Specific class of TO)).

3.1.1.4.5.19 tokenGroups, tokenGroupsNoGCAcceptable

The tokenGroups attribute exists on both AD DS and AD LDS. The tokenGroupsNoGCAcceptable attribute exists on AD DS but not on AD LDS.

These two computed attributes return the set of SIDs from a transitive group membership expansion operation on a given object.

For AD DS, the tokenGroups attribute is not present if no GC server is available to evaluate the transitive reverse memberships. The tokenGroupsNoGCAcceptable attribute can always be retrieved, but if no GC server is available, the set of SIDs may be incomplete.

Let U be the object from which the tokenGroups or tokenGroupsNoGCAcceptable attribute is being read.

- If U!objectSid does not exist, U!tokenGroups and U!tokenGroupsNoGCAcceptable are not present.
- Otherwise, U!tokenGroups and U!tokenGroupsNoGCAcceptable are the result of the algorithm in [MS-DRSR] section 4.1.8.3 (IDL_DRSGetMemberships) using DRS_MSG_REVMEMB_REQ_V1.OperationType=RevMembGetGroupsForUser, DRS_MSG_REVMEMB_REQ_V1.ppDsNames=U, and DRS_MSG_REVMEMB_REQ_V1.pLimitingDomain = the domain for which the server is a DC.

3.1.1.4.5.20 tokenGroupsGlobalAndUniversal

The tokenGroupsGlobalAndUniversal attribute exists on AD DS but not on AD LDS.

This computed attribute returns the set of SIDs of global and universal groups resulting from a transitive group membership expansion operation on a given object. This attribute is not present if no GC server is available to evaluate the transitive reverse memberships.

Let U be the object from which the tokenGroupsGlobalAndUniversal attribute is being read.

- If U!objectSid does not exist, U!tokenGroupsGlobalAndUniversal is not present.
- Otherwise let S be the set of SIDs returned by invoking the algorithm in [MS-DRSR] section 4.1.8.3 (IDL_DRSGetMemberships) using DRS_MSG_REVMEMB_REQ_V1.OperationType=RevMembGetAccountGroups, DRS_MSG_REVMEMB_REQ_V1.ppDsNames=U, and DRS_MSG_REVMEMB_REQ_V1.pLimitingDomain = the domain for which the server is a DC.
- Let accumulator set T be the Null set.
- For each SID s in S:
  - Let X be the set of SIDs returned by invoking the algorithm in [MS-DRSR] section 4.1.8.3 (IDL_DRSGetMemberships) using DRS_MSG_REVMEMB_REQ_V1.OperationType=RevMembGetUniversalGroups, DRS_MSG_REVMEMB_REQ_V1.ppDsNames=s, and DRS_MSG_REVMEMB_REQ_V1.pLimitingDomain = NULL.
  - T = T union X.
- U!tokenGroupsGlobalAndUniversal is the union of T and S.

3.1.1.4.5.21 possibleInferiors

Let TO be the object from which the possibleInferiors attribute is being read.

Let C be the classSchema object corresponding to TO!governsID.

The value of TO!possibleInferiors is the set of O!governsID for each Object O where

- (O is in the schema NC)
- and (O!objectClass is classSchema)
- and (not O!systemOnly)
- and (not O!objectClassCategory is 2)
- and (not O!objectClassCategory is 3)
- and ((C is contained in POSSSUPERIORS(O))

3.1.1.4.5.22 msDS-QuotaEffective

Let TO be the object from which the msDS-QuotaEffective attribute is being read.

Let R be the root object of the NC containing TO.

Let SID be the sid specified by the LDAP extended control LDAP_SERVER_QUOTA_CONTROL_OID or, if none is specified, the requester's SID.

Let SIDS be the set of SIDs including SID and the set of SIDs returned by tokenGroups.

The value of TO!msDS-QuotaEffective is the maximum of all O!msDS-QuotaAmount for each object O where:

- (TO is the object:

GetWellknownObject(n: R, guid: GUID_NTDS_QUOTAS_CONTAINER_W))

- and (O is a child of TO)
- and (the client has access to O as specified in section 3.1.1.4.3)
- and (the client has access to O!msDS-QuotaAmount as specified in section 3.1.1.4.3)
- and (the client has access to O!msDS-QuotaTrustee as specified in section 3.1.1.4.3)
- and (there exists S in SIDS such that S is equal to O!msDS-QuotaTrustee)

3.1.1.4.5.23 msDS-QuotaUsed

Let TO be the object from which the msDS-QuotaUsed attribute is being read.

Let C be the Most Specific Class from TO!objectClass.

Let R be the root object of the NC containing TO.

Let SID be the SID specified by the LDAP extended control LDAP_SERVER_QUOTA_CONTROL_OID or, if none is specified, the requester's SID.

The value of TO!msDS-QuotaUsed is:

- (cLive + ((cTombstoned * TO!msDS-TombstoneQuotaFactor)+99)/100)

where:

- cLive is the number of non-tombstoned objects associated with SID, and cTombstoned is the number of tombstoned objects associated with SID, as detailed in section 3.1.1.5.2.5, Quota Calculation.

when:

- (TO is the object:

  GetWellknownObject(n: R, guid: GUID_NTDS_QUOTAS_CONTAINER_W))

3.1.1.4.5.24 msDS-TopQuotaUsage

Let TO be the object from which the msDS-TopQuotaUsage attribute is being read.

Let R be the root object of the NC containing TO.

TO!msDS-TopQuotaUsage equals a set of XML-encoded strings sorted by the element quotaUsed when:

- TO is the object:

  GetWellknownObject(n: R, guid: GUID_NTDS_QUOTAS_CONTAINER_W)

Each string represents the quota information for a SID as specified in section 3.1.1.5.2.5, Quota Calculation. The format of the XML-encoded string is:

<MS_DS_TOP_QUOTA_USAGE>

<partitionDN>*DN of the NC containing TO* **</partitionDN>**

<ownerSID>*SID of quota user* **</ownerSID>**

<quotaUsed>*rounded up value of quota used (computed)* **</quotaUsed>**

<tombstoneCount>*value in the TombstoneCount column* **</tombstoneCount>**

<totalCount>*value in the TotalCount column* **</totalCount>**

</MS_DS_TOP_QUOTA_USAGE>

where quotaUsed is computed as specified in msDS-QuotaUsed with cLive set to (totalCount - tombstoneCount).

The number of values returned can be controlled with the ";range" syntax as detailed in Range Retrieval of Attribute Values in section 3.1.1.3.1.3.3. The default range is 10 for this attribute.

3.1.1.4.5.25 ms-DS-UserAccountAutoLocked

The ms-DS-UserAccountAutoLocked attribute exists on AD LDS but not on AD DS.

Let TO be the object from which the ms-DS-UserAccountAutoLocked attribute is being read. Let ST be the current time, read from the system clock.

If the machine running AD LDS is joined to a domain D, TO!ms-DS-UserAccountAutoLocked is true if both of the following are true:

- The LDAP configurable setting ADAMDisablePasswordPolicies ≠ 1.
- TO!lockoutTime ≠ 0 and either (1) D!lockoutDuration (regarded as an unsigned quantity) < 0x8000000000000000, or (2) ST + D!lockoutDuration (regarded as a signed quantity) ≤ TO!lockoutTime.

If the machine running AD LDS is not joined to a domain, TO!ms-DS-UserAccountAutoLocked is true if both of the following are true:

- The LDAP configurable setting ADAMDisablePasswordPolicies ≠ 1.
- TO!lockoutTime ≠ 0 and (current time - TO!lockoutTime) ≤ X, where X is determined by the policy of the machine on which AD LDS is running.

3.1.1.4.5.26 msDS-UserPasswordExpired

The msDS-UserPasswordExpired attribute exists on AD LDS but not on AD DS.

Let TO be the object from which the msDS-UserPasswordExpired attribute is being read. Let ST be the current time, read from the system clock.

If the machine running AD LDS is joined to a domain, let D be the root of the domain NC of the joined domain. Then TO!msDS-UserPasswordExpired is true if all of the following are true:

- The LDAP configurable setting ADAMDisablePasswordPolicies ≠ 1.
- None of bits ADS_UF_SMARTCARD_REQUIRED, ADS_UF_DONT_EXPIRE_PASSWD, ADS_UF_WORKSTATION_TRUST_ACCOUNT, ADS_UF_SERVER_TRUST_ACCOUNT, ADS_UF_INTERDOMAIN_TRUST_ACCOUNT is set in TO!userAccountControl.
- TO!pwdLastSet = null, or TO!pwdLastSet = 0, or (D!maxPwdAge ≠ 0x8000000000000000 and (ST - TO!pwdLastSet) > D!maxPwdAge)).

If the machine running AD LDS is not joined to a domain, then TO!msDS-UserPasswordExpired is true if all of the following are true:

- The LDAP configurable setting ADAMDisablePasswordPolicies ≠ 1.
- None of bits ADS_UF_SMARTCARD_REQUIRED, ADS_UF_DONT_EXPIRE_PASSWD, ADS_UF_WORKSTATION_TRUST_ACCOUNT, ADS_UF_SERVER_TRUST_ACCOUNT, ADS_UF_INTERDOMAIN_TRUST_ACCOUNT is set in TO!userAccountControl.
- TO!pwdLastSet = null, or TO!pwdLastSet = 0, or (ST - TO!pwdLastSet) > X, where X is determined by the policy of the machine on which AD LDS is running.

3.1.1.4.5.27 msDS-PrincipalName

The msDS-PrincipalName attribute has different behavior on AD DS and AD LDS.

Let TO be the object from which the msDS-PrincipalName attribute is being read.

For AD DS, the value of TO!msDS-PrincipalName is either (1) the NetBIOS domain name, followed by a backslash ("\"), followed by TO!sAMAccountName, or (2) the value of TO!objectSid in SDDL SID string format ([MS-DTYP] section 2.4.2.1).

For AD LDS, let OBJSID be the value of TO!objectSid. If OBJSID is the SID of a security principal of the computer on which Active Directory is running, then TO!msDS-PrincipalName is the NetBIOS computer name, followed by a backslash ("\"), followed by the name of the security principal. If the computer on which Active Directory is running is a member of a domain, and OBJSID is a SID for a security principal S in that domain, then TO!msDS-PrincipalName is the NetBIOS domain name, followed by a backslash ("\"), followed by S!sAMAccountName. Otherwise, the value of TO!msDS-PrincipalName is the value of TO!objectSid in SDDL SID string format ([MS-DTYP] section 2.4.2.1).

3.1.1.4.5.28 parentGUID

This attribute is not present on an object that is the root of an NC. For all other objects, let TO be the object from which the parentGUID attribute is being read and let TP be TO!parent. TO!parentGUID is equal to TP!objectGUID.

3.1.1.4.5.29 msDS-SiteName

The msDS-SiteName attribute exists on AD DS but not on AD LDS.

Let TO be the object on which msDS-SiteName is being read. If TO is an nTDSDSA object or a server object, then TO!msDS-SiteName is equal to the value of the RDN of the site object under which TO is located. For example, given a TO that is an nTDSDSA object with the DN "CN=NTDS Settings, CN=TESTDC-01, CN=Servers, CN=Default-First-Site-Name, CN=Sites, CN=Configuration, DC=fabrikam, DC=com", the value of TO!msDS-SiteName is "Default-First-Site-Name".

If TO is a computer object, then let TS be the server object named by TO!serverReferenceBL. TO!msDS-SiteName equals TS!msDS-SiteName.

If TO is neither a computer, server, nor nTDSDSA object, then TO!msDS-SiteName is not present.

3.1.1.4.5.30 msDS-isRODC

The msDS-isRODC attribute exists on AD DS but not on AD LDS.

This attribute indicates whether a specified DC is an RODC. Let TO be the object on which msDS-isRODC is being read. If TO is not an nTDSDSA, computer, or server object, then TO!msDS-isRODC is not present.

- If TO is an nTDSDSA object:
  - If TO!objectCategory equals the DN of the classSchema object for the nTDSDSA object class, then TO!msDS-isRODC is false. Otherwise, TO!msDS-isRODC is true.
- If TO is a server object:
  - Let TN be the nTDSDSA object whose DN is "CN=NTDS Settings," prepended to the DN of TO. Apply the previous rule for the "TO is an nTDSDSA object" case, substituting TN for TO.
- If TO is a computer object:
  - Let TS be the server object named by TO!serverReferenceBL. Apply the previous rule for the "TO is a server object" case, substituting TS for TO.

3.1.1.4.5.31 msDS-isGC

The msDS-isGC attribute exists on AD DS but not on AD LDS.

This attribute indicates whether a specified DC is a GC server. Let TO be the object on which msDS-isGC is being read. If TO is not an nTDSDSA, computer, or server object, then TO.msDS-isGC is not present.

- If TO is an nTDSDSA object:
  - TO!msDS-isGC iff TO!options has the NTDSDSA_OPT_IS_GC bit set (section 6.1.1.2.2.1.2.1.1).
- If TO is a server object:
  - Let TN be the nTDSDSA object whose DN is "CN=NTDS Settings," prepended to the DN of TO. Apply the previous rule for the "TO is an nTDSDSA object" case, substituting TN for TO.
- If TO is a computer object:
  - Let TS be the server object named by TO!serverReferenceBL. Apply the previous rule for the "TO is a server object" case, substituting TS for TO.

3.1.1.4.5.32 msDS-isUserCachableAtRodc

The msDS-IsUserCachableAtRodc attribute exists on AD DS but not on AD LDS.

This attribute indicates whether a specified RODC is permitted by administrator policy to cache the secret attributes of a specified security principal. The DN of the security principal is specified using the LDAP Control LDAP_SERVER_DN_INPUT_OID. The DN specified may be either an RFC 2253–style DN or one of the alternate DN formats specified in section 3.1.1.3.1.2.4.

Let TO be the object on which msDS-IsUserCachableAtRodc is being read. If TO is not an nTDSDSA, computer, or server object, then TO!msDS-IsUserCachableAtRodc is not present.

- If TO is a computer object:
  - If TO!userAccountControl does not have the ADS_UF_PARTIAL_SECRETS_ACCOUNT bit set, TO!msDS-IsUserCachableAtRodc is not present.

- If TO!userAccountControl has the ADS_UF_PARTIAL_SECRETS_ACCOUNT bit set, the value of TO!msDS-IsUserCachableAtRodc is calculated as follows:
  - Let D be the DN of the user principal specified using LDAP Control LDAP_SERVER_DN_INPUT_OID. If the DN of a security principal is not explicitly specified, D is the DN of the current requester.
  - TO!msDS-IsUserCachableAtRodc = GetRevealSecretsPolicyForUser(TO!distinguishedName, D) (procedure GetRevealSecretsPolicyForUser is defined in [MS-DRSR] section 4.1.10.5.14).
- If TO is a server object:
  - Let TC be the computer object named by TO!serverReference. Apply the previous rule for the "TO is a computer object" case, substituting TC for TO.
- If TO is an nTDSDSA object:
  - Let TS be the server object that is the parent of TO. Apply the previous rule for the "TO is a server object" case, substituting TS for TO.

3.1.1.4.5.33 msDS-UserPasswordExpiryTimeComputed

The msDS-UserPasswordExpiryTimeComputed attribute exists on AD DS but not on AD LDS.

This attribute indicates the time when the password of the object will expire. Let TO be the object on which the attribute msDS-UserPasswordExpiryTimeComputed is read. If TO is not in a domain NC, then TO!msDS-UserPasswordExpiryTimeComputed = null. Otherwise let D be the root of the domain NC containing TO. The DC applies the following rules, in the order specified below, to determine the value of TO!msDS-UserPasswordExpiryTimeComputed:

- If any of the ADS_UF_SMARTCARD_REQUIRED, ADS_UF_DONT_EXPIRE_PASSWD, ADS_UF_WORKSTATION_TRUST_ACCOUNT, ADS_UF_SERVER_TRUST_ACCOUNT, ADS_UF_INTERDOMAIN_TRUST_ACCOUNT bits is set in TO!userAccountControl, then TO!msDS-UserPasswordExpiryTimeComputed = 0x7FFFFFFFFFFFFFFF.
- Else, if TO!pwdLastSet = null, or TO!pwdLastSet = 0, then TO!msDS-UserPasswordExpiryTimeComputed = 0.
- Else, if Effective-MaximumPasswordAge = 0x8000000000000000, then TO!msDS-UserPasswordExpiryTimeComputed = 0x7FFFFFFFFFFFFFFF (where Effective-MaximumPasswordAge is defined in [MS-SAMR] section 3.1.1.5).
- Else, TO!msDS-UserPasswordExpiryTimeComputed = TO!pwdLastSet + Effective-MaximumPasswordAge (where Effective-MaximumPasswordAge is defined in [MS-SAMR] section 3.1.1.5).

3.1.1.4.5.34 msDS-RevealedList

The msDS-RevealedList attribute exists on AD DS (starting with Windows Server 2008 operating system) but not on AD LDS.

The msDS-RevealedList attribute exists only on the computer object of an RODC. The value of msDS-RevealedList is a multivalued DN-String. The string portion of each value is the lDAPDisplayName of a secret attribute, and the DN portion of each value names an object. Each value represents the presence of a value for the named attribute on the named object on the RODC; in other words, the value has been "revealed" to the RODC.

The msDS-RevealedList attribute is constructed from the msDS-RevealedUsers attribute as follows.

Let *O* be the object from which the msDS-RevealedList attribute is being read.

Let *RESULT* be a set of DN-String, initially empty.

For each *V* (a DN-Binary) in *O*!msDS-RevealedUsers do the following:

- Let *USER* be the object with DN *V.object_DN*.
- Let *P* (a PROPERTY_META_DATA, see [MS-DRSR] section 4.1.10.2.19) equal *V.binary_value*.
- Let *SCH* equal SchemaObj(*P.attrType*) ([MS-DRSR] section 5.179).
- Let *RV* be a DN-String with *RV.string_value* equal *SCH*!lDAPDisplayName and *RV.object_DN* equal *V.object_DN*.
- Let *A* be *SCH*!lDAPDisplayName.
- If AttributeStampCompare(*P.propMetadataExt*, AttrStamp(*USER*, *P.attrType*)) = 0, set *RESULT* = *RESULT* + {*RV* }. (See [MS-DRSR] section 4.1.10.3.5 for procedure AttributeStampCompare, and [MS-DRSR] section 5.13 for procedure AttrStamp.)

Return the set *RESULT* (if empty, the msDS-RevealedList attribute is not present).

3.1.1.4.5.35 msDS-RevealedListBL

The msDS-RevealedListBL attribute exists on AD DS (starting with Windows Server 2008 operating system) but not on AD LDS.

This attribute behaves precisely like a back link attribute for the msDS-RevealedList constructed attributes described in the previous section.

Therefore, the msDS-RevealedList attribute exists only on a user object, one or more of whose secret attributes have been "revealed" to an RODC. The value is the set of RODCs (represented by their computer objects) to which one or more of the given user object's secret attributes have been revealed.

3.1.1.4.5.36 msDS-ResultantPSO

The msDS-ResultantPSO attribute exists on AD DS beginning with Windows Server 2008 operating system. This attribute does not exist on AD LDS. This attribute specifies the effective password policy applied on this object.

The value of msDS-ResultantPSO is a single value of Object (DS-DN) syntax. This attribute is constructed as follows:

Let *RESULTSET* be a set of DS-DN, initially empty.

Let *U* be the object from which the msDS-ResultantPSO attribute is being read.

- If the domain functional level is less than DS_BEHAVIOR_WIN2008, then there is no value in this attribute.
- If *U*!objectClass does not contain the value "user", then there is no value in this attribute.
- If the bit for ADS_UF_NORMAL_ACCOUNT (see section 2.2.15) is not set in *U*!userAccountControl, then there is no value in this attribute.

- If the RID in *U*!objectSid is equal to DOMAIN_USER_RID_KRBTGT, then there is no value in this attribute.
- If the *U*!msDS-SecondaryKrbTgtNumber attribute has a value, then there is no value in this attribute.
- Let RESULTSET be the values of *U*!msDS-PSOApplied that are of object class msDS-PasswordSettings and are under the Password Settings container (see section 6.1.1.4.11.1)
- If RESULTSET is empty:
  - Let *S* be the set of objects returned by invoking the algorithm in [MS-DRSR] section 4.1.8.3 (IDL_DRSGetMemberships) using DRS_MSG_REVMEMB_REQ_V1.OperationType=RevMembGetAccountGroups, DRS_MSG_REVMEMB_REQ_V1.ppDsNames=U, and DRS_MSG_REVMEMB_REQ_V1.pLimitingDomain = the domain for which the server is a DC.
  - For each *O* (an object) in *S* do the following:
    - *RESULTSET* = *RESULTSET* union *O*!msDS-PSOApplied
- Sort objects in set *RESULTSET* according to msDS-PasswordSettingsPrecedence values, breaking ties with objectGUID values, with smaller values coming first.
- Return the first element in the sorted *RESULTSET* (if empty, the msDS-ResultantPSO attribute is not present).

3.1.1.4.5.37 msDS-LocalEffectiveDeletionTime

The msDS-LocalEffectiveDeletionTime attribute exists on AD DS and AD LDS, beginning with Windows Server 2008 R2 operating system.

This attribute contains the value that a replica maintains as the time when the object was transformed into a tombstone or deleted-object.

Each DC is permitted to modify this value locally for implementation-specific reasons outside the state model. Therefore, this value does not necessarily accurately reflect when the object was actually transformed. However, no replica is permitted to modify this value to be earlier than the actual time that the object was transformed. This value is not replicated. Therefore, for a specific object, each DC may have a different value for this attribute.

When the Recycle Bin optional feature is enabled, each replica will transform the deleted-object into a recycled-object some time after the difference that exists between the current time and the value of msDS-LocalEffectiveDeletionTime is greater than the value of the deleted-object lifetime.

When the Recycle Bin optional feature is not enabled, the DC makes no use of this value. When this attribute exists on a tombstone, it is not used by the replica.

3.1.1.4.5.38 msDS-LocalEffectiveRecycleTime

The msDS-LocalEffectiveRecycleTime attribute exists on AD DS and AD LDS, beginning with Windows Server 2008 R2 operating system.

This attribute contains the value that a replica maintains as the time when the object was transformed into a tombstone or recycled-object.

Each DC is permitted to modify this value locally for implementation-specific reasons outside the state model. Therefore, this value does not necessarily accurately reflect when the object was actually transformed. However, no replica is permitted to modify this value to be earlier than the actual time that the object was transformed. This value is not replicated. Therefore, for a specific object, each replica may have a different value for this attribute.

Each replica will remove the tombstone or recycled-object some time after the difference that exists between the current time and the value of msDS-LocalEffectiveRecycleTime is greater than the value of the tombstone lifetime.

3.1.1.4.5.39 msDS-ManagedPassword

The msDS-ManagedPassword attribute exists in AD DS, beginning with Windows Server 2012 operating system. This attribute contains a BLOB with password information for group-managed service accounts.

Let TO be the object on which msDS-ManagedPassword is being read. If TO is not an msDS-GroupManagedServiceAccount object, then TO!msDS-ManagedPassword is not present. If the DC is not writable, then TO!msDS-ManagedPassword cannot be constructed and the request is forwarded to a writable DC by the RODC.

The value of TO!msDS-ManagedPassword is obtained by calling GetgMSAPasswordBlob(TO) (defined later in this section), which uses the functions defined next.

Define function PostProcessPasswordBuffer(*Password*: OCTET STRING), which returns an octet string [ITUX680] as follows:

1. Let RESULT be set to *Password*, which is a BLOB.
2. Take RESULT and convert each wide (2-byte) NULL character into a wide value of 1 (0x00 0x01) to guarantee that the resulting string is a Unicode string with no intervening NULL characters that would limit its length.
3. Set the last wide character in RESULT to NULL to terminate the string.
4. Return RESULT.

Define function MaxClockSkew(), which returns the integer 3,000,000,000. This is a quantity of 10^(-7) second units of time; that is, five minutes in 100ns units.

Define function GmsaSD(), which returns the security descriptor corresponding to the policy on all msDS-GroupManagedServiceAccount object keys:

```
static const BYTE gmsaSecurityDescriptor[] = {/* O:SYD:(A;;FRFW;;;S-1-5-9) */
    0x01, 0x00, 0x04, 0x80, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x14, 0x00, 0x00, 0x00, 0x02, 0x00, 0x1c, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00,
    0x14, 0x00, 0x9f, 0x01, 0x12, 0x00, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x09,
    0x00, 0x00, 0x00, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x12, 0x00, 0x00, 0x00
    };
```

Define function GenerateGmsaPassword(*Key*: L1 or L2 key ([MS-GKDI] section 2.2.4), *HashAlg*: null-terminated Unicode string , *AccountSID*: SID), which returns a password and a key identifier, KeyID, as follows:

1. Use the same processing rules as defined for KDF ([MS-GKDI] section 3.1.4.1.2) where:

- HashAlg (for KDF) contains *HashAlg*.
- K_I contains *Key*.
- Label contains the Unicode (wide-character) NULL-terminated string "GMSA PASSWORD" (without the quotes).
- Context contains the binary representation of the *AccountSID* parameter.
- L contains the password length, in bytes, including the terminating NULL.

2. Return KeyID and the password to the caller.

Define function MarshalPassword(*Current_Password*: OCTET STRING, *Previous_Password*: OCTET STRING (optional), *QueryPasswordInterval*: FILETIME, *UnchangedPasswordInterval*: FILETIME). This function returns an msDS-ManagedPassword BLOB using the MSDS-MANAGEDPASSWORD_BLOB structure from section 2.2.18, which is constructed as follows:

- The **Version** field is set to 0x0001.
- The **Reserved** field is set to 0x0000.
- The **Length** field is set to the length, in bytes, of the msDS-ManagedPassword BLOB.
- The **CurrentPasswordOffset** field is set to the offset, in bytes, from the beginning of this structure to the **CurrentPassword** field.
- The **PreviousPasswordOffset** field is set to the offset, in bytes, from the beginning of this structure to the **PreviousPassword** field. If the *Previous_Password* parameter is not included, this field is set to 0x0000.
- The **QueryPasswordIntervalOffset** field is set to the offset, in bytes, from the beginning of this structure to the **QueryPasswordInterval** field.
- The **UnchangedPasswordIntervalOffset** field is set to the offset, in bytes, from the beginning of this structure to the **UnchangedPasswordInterval** field.
- The **CurrentPassword** field is set to *Current_Password*.
- The **PreviousPassword** field is set to *Previous_Password*. If the *Previous_Password* parameter is not included, then this field MUST be absent.
- The **AlignmentPadding** field is constructed with enough bytes of padding to align the **QueryPasswordInterval** field to a 64-bit boundary.
- The **QueryPasswordInterval** field is set to *QueryPasswordInterval*.
- The **UnchangedPasswordInterval** field is set to *UnchangedPasswordInterval*.

Define function GetIntervalId(*TimeStamp*: FILETIME), which returns L0KeyID: INTEGER, L1KeyID: INTEGER, and L2KeyID: INTEGER as follows:

1. Let KeyCycleDuration be the integer 360,000,000,000. This is a quantity of 10^(-7) second units of time; that is, 10 hours in 100ns units.
2. Let TimeStamp be the number of 100ns intervals since January 1,1601, UTC.
3. Let L1_KEY_ITERATION be 32.

4. Let L2_KEY_ITERATION be 32.
5. L0KeyID = (ULONG)(*TimeStamp* / KeyCycleDuration / L2_KEY_ITERATION / L1_KEY_ITERATION)
6. L1KeyID = (*TimeStamp* / KeyCycleDuration / L2_KEY_ITERATION) & (L1_KEY_ITERATION - 1)
7. L2KeyID = (*TimeStamp* / KeyCycleDuration) & (L2_KEY_ITERATION - 1)
8. Return L0KeyID, L1KeyID, and L2KeyID.

Define function GKDIGetKeyStartTime(*KeyID*: GUID), which returns a FILETIME structure as follows:

1. Extract the variables L0, L1, and L2 from the Group Key Envelope structure ([MS-GKDI] section 2.2.4) identified by KeyID. The Group Key Envelope fields of relevance are **L0 index**, **L1 index**, and **L2 index**, respectively.
2. Let KeyCycleDuration be the integer 360,000,000,000. This is a quantity of 10^(-7) second units of time; that is, 10 hours in 100ns units.
3. Let L1_KEY_ITERATION be 32.
4. Let L2_KEY_ITERATION be 32.
5. Return ((L0 * L1_KEY_ITERATION * L2_KEY_ITERATION) + (L1 * L2_KEY_ITERATION) + L2) * KeyCycleDuration

Define function GetPasswordBasedOnTimeStamp(*TimeStamp*: FILETIME, *AccountSID*: SID), which returns an msDS-ManagedPassword BLOB (section 2.2.18) and KeyID: GUID as follows:

1. Call GetIntervalID() with the supplied *TimeStamp* to compute variables L0, L1, and L2.
2. Call *GetKey()* ([MS-GKDI] section 3.1.4.1) to compute the output key where:
   - *hBinding* contains an RPC binding handle ([C706] and [MS-RPCE]) to a GKDI server.
   - *cbTargetSD* contains the length, in bytes, of the security descriptor supplied in *pbTargetSD*.
   - *pbTargetSD* contains a pointer to the security descriptor returned by GmsaSD().
   - *pRootKeyID* is set to NULL.
   - *L0KeyID* contains L0 returned in step 1.
   - *L1KeyID* contains L1 returned in step 1.
   - *L2KeyID* contains L2 returned in step 1.
3. Compute the group key using the output key from step 2 and the same processing rules as defined in step 2 of [MS-GKDI] section 3.2.4.3.
4. Call GenerateGmsaPassword() to obtain the password BLOB and KeyID where:
   - *Key* contains the group key from step 3.
   - *HashAlg* contains **Hash algorithm name** from the KDF parameters ([MS-GKDI] section 2.2.1) of the output key from step 2.
   - *AccountSID* contains the *AccountSID* parameter passed to this function.

5. Call PostProcessPasswordBuffer() with the returned password BLOB to make it into a properly NULL-terminated Unicode string.

6. Return the password BLOB and KeyID to the caller.

Define function GetPasswordBasedOnKeyID(*Key-ID*: GUID*, Account-SID*: SID), which returns an msDS-ManagedPassword BLOB (section 2.2.18) as follows:

1. Extract the variables L0, L1, and L2 and the root key ID from the Group Key Envelope data structure ([MS-GKDI] section 2.2.4) identified by *Key-ID*. The Group Key Envelope fields of relevance are **L0 index**, **L1 index**, **L2 index**, and **Root key identifier**, respectively.

2. Call GetKey() ([MS-GKDI] section 3.1.4.1) to compute the output key where:

   - *hBinding* contains an RPC binding handle ([C706] and [MS-RPCE]) to a GKDI server.
   - *cbTargetSD* contains the length, in bytes, of the security descriptor supplied in *pbTargetSD*.
   - *pbTargetSD* contains a pointer to the security descriptor returned by GmsaSD().
   - *pRootKeyID* is set to the root key ID returned in step 1.
   - *L0KeyID* contains L0 returned in step 1.
   - *L1KeyID* contains L1 returned in step 1.
   - *L2KeyID* contains L2 returned in step 1.

3. Compute the group key using the output key from step 2 and the same processing rules as defined in step 3 of [MS-GKDI] section 3.2.4.3.

4. Call GenerateGmsaPassword() where:

   - *Key* contains the group key from step 3.
   - *HashAlg* contains **Hash algorithm name** from the KDF parameters ([MS-GKDI] section 2.2.1) of the output key from step 2.
   - *AccountSID* contains the *Account-SID* parameter passed to this function.

5. Call PostProcessPasswordBuffer() to convert the returned BLOB into a properly NULL-terminated Unicode string.

6. Return the password BLOB to the caller.

Define function GetgMSAPasswordBlob(TO: OBJECT), which returns an msDS-ManagedPassword BLOB structure (section 2.2.18) as follows using integer arithmetic where divisions are rounded down without a remainder.

1. If the connection is not encrypted, ERROR_DS_CONFIDENTIALITY_REQUIRED is returned.

2. If the caller does not have the RIGHT_DS_READ_PROPERTY control access right on the security descriptor in the TO!msDS-GroupMSAMembership attribute ([MS-ADA2] section 2.313), the error operationsError / ERROR_DS_CANT_RETRIEVE_ATTRS is returned. This access check is also specified in section 3.1.1.4.4.

3. Convert the TO!msDS-ManagedPasswordInterval attribute ([MS-ADA2] section 2.351) into the rollover interval as follows:

1. Let KeyCycleDuration be the integer 360,000,000,000. This is a quantity of 10^(-7) second units of time; that is, 10 hours in 100ns units.

2. Let GKDIRolloverInterval = (TO!msDS-ManagedPasswordInterval * 24 / KeyCycleDuration) * KeyCycleDuration.

4. Let a variable called CurrentKeyExpirationTime be computed as follows:

    1. If the TO!msDS-ManagedPasswordId attribute ([MS-ADA2] section 2.350) exists, call GKDIGetKeyStartTime() where:

        - *KeyID* contains TO!msDS-ManagedPasswordId.

        Set CurrentKeyExpirationTime = the time returned by GKDIGetKeyStartTime() + GKDIRolloverInterval.

    2. Otherwise, set CurrentKeyExpirationTime = the TO!creationTime attribute ([MS-ADA1] section 2.131).

5. If TO!msDS-ManagedPasswordId does not exist or CurrentKeyExpirationTime is less than the current time, then:

    1. Let StaleCount be zero.

    2. Let NewKeyStartTime = CurrentKeyExpirationTime.

    3. Let NewKeyStartTime = NewKeyStartTime + GDKIRolloverInterval and StaleCount = StaleCount +1 until NewKeyStartTime is greater than the current time.

    4. Call GetPasswordBasedOnTimestamp() where:

        - *Timestamp* contains NewKeyStartTime.

        - *AccountSID* contains the TO!objectSid attribute ([MS-ADA3] section 2.45).

        Let NewKeyID be the returned KeyID. Let NewPassword be the returned password.

    5. Let a variable called OldKeyID be computed as follows:

        1. If StaleCount is zero AND TO!msDS-ManagedPasswordId exists and is not NULL:

            1. Call GetPasswordBasedOnKeyID() where:

                - *Key-ID* contains TO!msDS-ManagedPasswordId.

                - *Account-SID* contains TO!objectSid ([MS-ADA3] section 2.45).

            2. Let OldPassword be the returned password and set OldKeyID to the value of TO!msDS-ManagedPasswordId.

        2. Otherwise, if the current time - TO!creationTime >= GDKIRolloverInterval, the current key cannot be reused as the previous key. Call GetPasswordBasedOnTimeStamp() where:

            - *Timestamp* contains NewKeyStartTime – GDKIRolloverInterval.

            - *AccountSID* contains TO!objectSid.

            Set OldKeyID to the returned KeyID. Let OldPassword be the returned password.

3. Otherwise, the account is not old enough to have a previous password and neither the OldKeyID nor the OldPassword will be returned.

6. Let variables called QueryPasswordInterval and UnchangedPasswordInterval be computed as follows:

   1. Let NewKeyExpirationTime = NewKeyStartTime + GKDIRolloverInterval.

   2. Call MaxClockSkew() and let MaxClockSkew be the returned value.

   3. If NewKeyExpirationTime - the current time <= MaxClockSkew:

      - Let QueryPasswordInterval be NewKeyExpirationTime - the current time.

      - Let UnchangedPasswordInterval be 0.

   4. Otherwise:

      - Let QueryPasswordInterval be NewKeyExpirationTime - the current time.

      - Let UnchangedPasswordInterval be NewKeyExpirationTime - the current time - MaxClockSkew.

7. Call MarshalPassword() where:

   - *Current_Password* contains NewPassword.

   - *Previous_Password* contains OldPassword.

   - *QueryPasswordInterval* contains QueryPasswordInterval.

   - *UnchangedPasswordInterval* contains UnchangedPasswordInterval.

   Return the resulting msDS-ManagedPassword BLOB.

6. If CurrentKeyExpirationTime - the current time <= MaxClockSkew(), create a new key that will be valid in the next epoch:

   1. Call GetPasswordBasedOnTimeStamp() where:

      - *Timestamp* contains CurrentKeyExpirationTime.

      - *AccountSID* contains TO!objectSid.

      Let NewKeyID be the returned KeyID. Let NewPassword be the returned password.

   2. Call GetPasswordBasedOnKeyID() where:

      - *Key-ID* contains TO!msDS-ManagedPasswordId.

      - *Account-SID* contains TO!objectSid.

      Let OldPassword be the returned password and do not return OldKeyID.

   3. Let QueryPasswordInterval = CurrentKeyExpirationTime - the current time.

   4. Let UnchangedPasswordInterval = CurrentKeyExpirationTime + GKDIRolloverInterval - MaxClockSkew - the current time.

5. Call MarshalPassword() where:

   - *Current_Password* contains NewPassword.
   - *Previous_Password* contains OldPassword.
   - *QueryPasswordInterval* contains QueryPasswordInterval.
   - *UnchangedPasswordInterval* contains UnchangedPasswordInterval.

   Return the resulting msDS-ManagedPassword BLOB.

7. Otherwise, create a current key:

   1. Call GetPasswordBasedOnKeyID() where:

      - *Key-ID* contains TO!msDS-ManagedPasswordId.
      - *Account-SID* contains TO!objectSid.

      Let NewPassword be the returned password.

   2. If the TO!msDS-ManagedPasswordPreviousId attribute ([MS-ADA2] section 2.352) exists, call GetPasswordBasedOnKeyID() where:

      - *Key-ID* contains TO!msDS-ManagedPasswordPreviousId.
      - *Account-SID* contains TO!objectSid.

      Let OldPassword be the returned password.

   3. Let QueryPasswordInterval = CurrentKeyExpirationTime - the current time.
   4. Let UnchangedPasswordInterval = CurrentKeyExpirationTime - MaxClockSkew - the current time.
   5. Call MarshalPassword() where:

      - *Current_Password* contains NewPassword.
      - *Previous_Password* contains OldPassword.
      - *QueryPasswordInterval* contains QueryPasswordInterval.
      - *UnchangedPasswordInterval* contains UnchangedPasswordInterval.

      Return the resulting msDS-ManagedPassword BLOB.

3.1.1.4.6 Referrals

When the server returns a referral as documented in section 3.1.1.3.1.4, it must determine which server(s) to refer the client to. The set of servers to which the client will be referred is the set of values returned by the following algorithm.

Let N be the DSNAME of the base of the LDAP search.

Let NSID be the sid portion of N.

Let NGUID be the guid portion of N.

Let NSTR be the dn portion of N.

The value is:

- (the values of O!dnsRoot for the object O where:
  - (NSTR is not present)
  - and (NGUID is not present)
  - and (NSID is present)
  - and ((O!nCName)!objectSid matches the domain sid from NSID)
  - and (O!parent is the Partitions container)
  - and (O!objectClass's most specific class is crossRef)
  - and (O!Enabled is true))
- and (the value for Root-Domain-NC!dnsRoot after prepending "gc._msdcs." and either replacing the first matching ":*" with ":3268" or, if there are no matches of ":*", then by appending ":3268" when:
  - (NSTR is not present)
  - and (NGUID is present))
- and (the values of O!dnsRoot for the object O where:
  - (NSTR is present)
  - and (O!nCName is a prefix for NSTR and is the longest prefix among all O satisfying these conditions)
  - and (O!parent is the Partitions container)
  - and (O!objectClass's **most specific object class** is crossRef)
  - and (O!Enabled is true))
- and (the value is Root-Domain-NC!superiorDNSRoot when:
  - (NSTR is present)
  - and (Root-Domain-NC!superiorDNSRoot is present)
  - and (there exists no object O such that
    - ((O!nCName is a prefix for NSTR)
    - and (O!parent is the Partitions container)
    - and (O!objectClass's most specific class is crossRef)
    - and (O!Enabled is true)))

- and (the value is the transform of TO.dn into a dotted string by concatenating the value for the first dc component with values for subsequent components separated by "." (for example, CN=bob,DC=One,DC=Two is transformed into One.Two) when:
  - ((NSTR is present)
  - and (Root-Domain-NC!superiorDNSRoot is not present)
  - and (there exists no object O such that
    - ((O!nCName is a prefix for NSTR)
    - and (O!parent is the Partitions container)
    - and (O!objectClass's most specific class is crossRef)
    - and (O!Enabled is true)))))

3.1.1.4.7 Continuations

When the server returns a continuation reference as documented in section 3.1.1.3.1.4, it must determine which server(s) to refer the client to. The set of servers to which the client will be referred is the set of values returned by the following algorithm.

Let TO be the base object of an LDAP Search.

Let NC be the NC replica containing TO.

The values are made up of:

- The values from O!dnsRoot for all objects O where
  - (O.dn is listed in NC!subRefs)
  - and (O!Enabled is true)
  - and (O!objectClass's most specific class is crossRef)
  - and
    - (((O!nCName is a prefix of TO.dn for all but the first component)

      and (the scope of the search is LDAP_SCOPE_ONELEVEL))

      or
    - ((O!nCName is a prefix of TO.dn)

      and (the scope of the search is LDAP_SCOPE_SUBTREE)))
- and the value for Root-Domain-NC!dnsRoot after prepending "gc._msdcs." and either replacing the first matching ":*" with ":3268" or, if there are no matches of ":*", then by appending ":3268" if and only if:
  - (TO!objectClass's most specific object class is addressBookContainer)

    and (the scope of the search is LDAP_SCOPE_ONELEVEL)

3.1.1.4.8 Effects of Defunct Attributes and Classes

If the forest functional level is less than DS_BEHAVIOR_WIN2003, a search that mentions a defunct class or attribute succeeds just as if the class or attribute were not defunct.

If the forest functional level is DS_BEHAVIOR_WIN2003 or greater:

- Instances of a defunct attribute cannot be read.
- Instances of a defunct class can be read using the filter term (objectClass=*).
- When reading any OID-valued attribute that contains identifiers for schema objects, if the attribute identifies a defunct schema object, the read returns an OID (the attributeID if an attribute, the governsID if a class) not a name (the lDAPDisplayName of an attribute or class). This behavior applies to the mustContain, systemMustContain, mayContain, systemMayContain, subClassOf, auxiliaryClass, and possSuperiors attributes of schema objects (that is, attributeSchema or classSchema objects that are located in the schema NC). This behavior also applies to the objectClass attribute of all other objects.

3.1.1.5 Updates

3.1.1.5.1 General

References

- LDAP attributes: fSMORoleOwner, invocationId, objectGUID.
- State model attributes: rdnType.
- LDAP classes: classSchema, crossRef, nTDSDSA, rIDManager.
- Glossary: config NC, default NC, **dsname**, NC replica, replicated attribute, schema NC.
- Abstract attribute repsTo: see [MS-DRSR] section 5.170.
- IDL_DRSReplicaSync method: see [MS-DRSR] section 4.1.23.
- DRS_MSG_REPSYNC message: see [MS-DRSR] section 4.1.23.1.1.
- DRS_MSG_REPSYNC_V1 message: see [MS-DRSR] section 4.1.23.1.2.
- Urgent replication specified by SAM: see [MS-SAMR] section 3.1.1.8.

This section specifies operations that are common for all originating update and **replicated update** operations. An update could be an Add, Modify, Modify DN, or Delete operation.

3.1.1.5.1.1 Enforce Schema Constraints

The originating update is validated for schema constraints as explained in Restrictions on Schema Extensions in section 3.1.1.2. Schema constraints are not enforced for replicated updates.

During an originating update of the Add and Modify operations, the server validates that the object being added or modified is consistent with the schema definition of the object of the objectClass values that are assigned to the object (see section 3.1.1.2 for more information):

- The mayContain/mustContain constraints that are applicable based on the selected objectClass values are enforced. The computation of the mayContain/mustContain set takes into

consideration the complete inheritance chain of the structural objectClass and the 88 object class as well as any auxiliary classes supplied. If any attributes in the mustContain set are not provided, the Add fails with *objectClassViolation* / <unrestricted>. If any attributes provided are not present in either the mayContain or mustContain sets, the Add fails with *objectClassViolation* / <unrestricted>. Exception: In AD LDS, the objectSid attribute is present on all application NC roots, even if this violates the schema mayContain/mustContain constraints.

- All attribute values are formed correctly according to the attribute syntax and satisfy schema constraints, such as single-valuedness, rangeLower/rangeUpper, and so on. See sections 3.1.1.2.3 through 3.1.1.2.5 for more information.
- All attribute values must be compliant with the rangeUpper and rangeLower constraints of the schema (see section 3.1.1.2.3). If a supplied value violates a rangeUpper or rangeLower constraint, then the Add fails with *constraintViolation* / <*unrestricted*>.
- All attribute values must be compliant with the isSingleValued constraint of the schema (see section 3.1.1.2.3). If multiple values are provided for an attribute that is single-valued, then the Add fails with *constraintViolation* / <*unrestricted*>.
- The attributeType of the first label of the object DN matches the rDNAttID of the structural object class or the 88 object class. Otherwise, *namingViolation* / *ERROR_DS_RDN_DOESNT_MATCH_SCHEMA* is returned. For example, it is not allowed to create an organizationalUnit with CN=test RDN; the correct RDN for an organizationalUnit object is OU=test. If there is no class C for which the attributeType is equal to C!rDNAttID, *namingViolation* / <*unrestricted*> is returned.

3.1.1.5.1.2 Naming Constraints

During an originating update of the Add, Modify, and Modify DN operations, the server validates the following naming constraints. Unless otherwise specified, the server returns the error *namingViolation* / <*unrestricted*> if a naming constraint is not met.

- The RDN must not contain a character with value 0xA.
- The RDN must not contain a character with value 0x0; otherwise, the server SHOULD return the error *invalidDNSyntax* / <*unrestricted*>. However, if the DC functional level is DS_BEHAVIOR_WIN2000, the server will not return an error.
- The DN must be compliant with [RFC2253].
- The RDN size must be less than 255 characters.

Naming constraints are not enforced for replicated updates.

3.1.1.5.1.3 Uniqueness Constraints

During an originating update of the Add, Modify, and Undelete operations on a DC with functional level of DS_BEHAVIOR_WIN2012R2 or greater, the server enforces the following constraint for the servicePrincipalName and userPrincipalName attributes if present on the object.

- In AD DS, if the DC functional level is DS_BEHAVIOR_WIN2012R2 or greater, then the new attribute value must be unique within the entire forest. If the DC is not a GC, then the DC should issue an LDAP search against a GC to determine uniqueness.
- In AD LDS, if the DC functional level is DS_BEHAVIOR_WIN2012R2 or greater, then the new attribute value must be unique within its own partition.

If another object exists with a duplicate userPrincipalName value, the operation fails with an extended error of *ERROR_DS_UPN_VALUE_NOT_UNIQUE_IN_FOREST*. If another object exists with a duplicate servicePrincipalName value, the operation fails with an extended error of *ERROR_DS_SPN_VALUE_NOT_UNIQUE_IN_FOREST*.

Uniqueness constraints are not enforced for replicated updates.

3.1.1.5.1.4 Transactional Semantics

The effects of an originating update are captured in the state model by committing a transaction. When the originating update is initiated by a protocol request, such as an LDAP Modify, the transaction is committed before sending the appropriate protocol response. The transaction has the ACID properties [GRAY] and provides at least degree 2 isolation of concurrent read and update requests [GRAY].

Transactions that are used to implement Active Directory provide degree 2 isolation of concurrent read and update requests.

Each Search request or Update request is performed as a transaction. When multiple Search requests are used to retrieve a large set of results, each request is its own transaction. An originating update is processed as one or more transactions. In some cases a request will cause transactions to occur after the response has been sent. Section 3.1.1.1.16 and the remainder of section 3.1.1.5 specify the transaction boundaries used for all originating updates and describes all cases where processing continues after the response.

3.1.1.5.1.5 Stamp Construction

Stamps for replicated attributes and link values will be updated for each originating update as defined in section 3.1.1.1. When applying replicated updates, stamps are constructed as described in [MS-DRSR] section 4.1.10.6.

3.1.1.5.1.6 Replication Notification

Each NC replica on the server has an associated abstract attribute repsTo. When an originating or replicated update occurs in the NC replica on the server, the server notifies each destination DC that has an entry in repsTo. The server notifies the destination DC by calling method IDL_DRSReplicaSync. The destination DC contacts the server and requests it to provide updates—this is event-driven replication as described in section 3.1.1.1.14.

The server sends replication notifications as follows:

Let *N* be the NC replica where the originating or replicated update has occurred on the server.

For each *i* in [0 .. (*N*!repsTo).*length*-1] do the following:

- Let *E* be *N*!repsTo[*i*].
- Let *C* be the crossRef object corresponding to *N*.
- Let *pmsgIn* be a reference to a structure of type DRS_MSG_REPSYNC.
- Set *pmsgIn->V1.pNC* to dsname of *N*.
- Let *O* be the nTDSDSA object of the server.
- Set *pmsgIn->V1.uuidDsaSrc* to *O*!objectGUID.

- Set *pmsgIn->V1.ulOptions* to (DRS_ASYNC_OP | DRS_UPDATE_NOTIFICATION).
- If (*E.replicaFlags* & DRS_WRITE_REP ≠ 0) then set *pmsgIn->V1.ulOptions* to (*pmsgIn->V1.ulOptions* | DRS_WRIT_REP).
- If the originating/replicated update satisfies the condition for urgent replication then set *pmsgIn->V1.ulOptions* to (*pmsgIn->V1.ulOptions* | DRS_SYNC_URGENT).
- Let *H* be the handle obtained by calling IDL_DRSBind against *E.uuidDsa*. If (*E.replicaFlags* & DRS_REF_GCSPN ≠ 0), then, for mutual authentication of the IDL_DRSBind client, use the service principal name associated with *E.uuidDsa* that begins with "GC" ([MS-DRSR] section 2.2.3.2).
- If (*pmsgIn->V1.ulOptions* & DRS_SYNC_URGENT = 0), then wait for an implementation-specific time *T*. If i = 0 the default time *T* is 15 seconds; if i > 0 the default time *T* is 3 seconds.
- Let *R* be the result of calling IDL_DRSReplicaSync(*H*, *1*, *pmsgIn*).
- Let *Z* be the current time.
- If *E.timeLastAttempt* > *Z* or *Z.timeLastAttempt* - *Z* > an implementation-specific duration *U*, update *N*!repsTo[*i*] as follows:
  - Set *E.timeLastAttempt* to *Z*.
  - Set *E.ulResultLastAttempt* to *R*.
  - If *R* = *0*, set *E.timeLastSuccess* to *Z* and set *E.cConsecutiveFailures* to 0.
  - If *R* ≠ *0*, increment *E.cConsecutiveFailures* by 1.
  - Set *N*!repsTo[*i*] to *E*.

  The default duration *U* is one hour.

3.1.1.5.1.7 Urgent Replication

Let *N* be the NC replica on the server. There are few originating/replicated updates in *N* that need to be replicated immediately to each destination DC that has an entry in *N*!repsTo. Updates that need to be replicated immediately are listed below:

- Creation of nTDSDSA object.
- Creation of crossRef object.
- Updates to schema object (attributeSchema or classSchema).
- Deletion of nTDSDSA object.
- Deletion of crossRef object.
- Update to secret object.
- Update to rIDManager object.
- Update to pwdLastSet and userAccountControl attributes as specified in [MS-SAMR] section 3.1.1.8.

- Unless the DC is running as ADAM or AD LDS, update to lockoutTime attribute as specified in [MS-SAMR] section 3.1.1.8.

The server behavior for urgent replication is specified in section 3.1.1.5.1.6.

3.1.1.5.1.8 Updates Performed Only on FSMOs

Certain originating update operations in Active Directory must be performed on a single master. For example, all schema updates must happen on the schema master FSMO DC; creation and deletion of crossRef objects representing naming contexts must happen on the domain naming FSMO DC. If the operation is attempted on a DC that does not hold the FSMO role, then it issues a referral to the current FSMO role owner. The following section describes how such updates are handled. The processing is not performed when applying replicated updates.

The following types and functions are used in specifying the FSMO-related processing of originating update.

The function IsEffectiveRoleOwner(roleObject:object) verifies that the current DC is the valid owner of the given FSMO role. The FSMO ownership is considered valid if a successful replication of the corresponding NC occurred with some replication partner. This function is defined later in this section.

For a given FSMO role, the function RoleUpdateScope(roleObject:Object) returns the set of objects and their attributes that can only be updated on the FSMO role owner DC. For example, for Schema Master FSMO Role (section 6.1.5.1), the set contains all objects residing within schema NC, with all of their attributes. The function is defined later in this section.

Define variable *timeLastReboot* equal to the time when the server last rebooted.

Define function IsEffectiveRoleOwner(*roleObject*: object), which returns a Boolean as follows:

- Let *S* be the nTDSDSA object of the server.
- If *S* ≠ *roleObject*!fSMORoleOwner, then return false.
- Let *N* be the NC containing roleObject.
- If there exists at least one entry *E* in *N*!repsFrom such that *E.timeLastSuccess* > *timeLastReboot*, then return true.
- Otherwise return false.

Let RoleType be the enumeration (SchemaMasterRole, DomainNamingMasterRole, InfrastructureMasterRole, RidAllocationMasterRole, PdcEmulationMasterRole).

Define function RoleObject(*n*: NC, *roleType*: RoleType), which returns an object as follows:

- If *roleType* = SchemaMasterRole,
  - if *n* = Schema NC, return *n*, otherwise return null.
- If *roleType* = DomainNamingMasterRole,
  - if *n* = Config NC, return Partition container of *n*, otherwise return null.
- If roleType = InfrastructureMasterRole,
  - if *n* = Default NC (AD DS), return Infrastructure container of *n*, otherwise return null.

- If *roleType* = RidAllocationMasterRole,
  - if *n* = Default NC (AD DS), return RID Manager container of *n*, otherwise return null.
- If *roleType* = PdcEmulationMasterRole,
  - if *n* = Default NC (AD DS), return *n*, otherwise return null.
- Otherwise return null.

Define function RoleUpdateScope(*roleObject*: object), which returns the set *S* as follows. *S* is a set such that each element is an object and a list of attributes associated with the object.

- Let *n* be the NC containing *roleObject*.
- Let *roleType* be the role corresponding to the *roleObject*; that is, RoleObject(*n*, *roleType*) = *roleObject*.
- If *roleType* = SchemaMasterRole, the union of:
  - The set of all objects and all attributes in the *roleObject*'s NC.
  - The RoleObject(Config NC, DomainNamingMasterRoll) with the msDS-Behavior-Version attribute.
- If *roleType* = DomainNamingMasterRole, the union of
  - *roleObject* and all attributes except msDS-Behavior-Version.
  - The objects that are children of *roleObject* and all attributes.
- If *roleType* = InfrastructureMasterRole, the union of
  - *roleObject* and all attributes.
  - The Updates container u of *roleObject*'s NC and all attributes.
  - The objects that are children u and all attributes.
- If *roleType* = RidAllocationMasterRole, the union of
  - *roleObject* and all attributes.
  - Let *I* = GetWellKnownObject(n, GUID_INFRASTRUCTURE_CONTAINER).
  - All children *C* of *I* and all attributes, such that *C*!objectClass contains infrastructureUpdate and *C*!proxiedObjectName is present.
  - If *C* is the computer object for the DC requesting the FSMO operation, *C* and all attributes.
  - The DC's rIDSet object.
- If *roleType* = PdcEmulationMasterRole,
  - *roleObject* and all attributes.
  - *n* with attributes wellKnownObjects and msDS-Behavior-Version.
- Otherwise return NULL.

Given those preliminaries, the following processing is performed on each object *O* on which an originating update is being made.

Let *O.A* be the attribute that is being updated.

Let *N* be the *NC* containing *O*.

For each RoleType *T* do the following:

- Let *R* = RoleObject(*N*, *T*)
- If *R* exists, then
  - Let *S* = RoleUpdateScope(*R*).
  - If *O* is not an element of {*S*} or *O.A* is not an element of {*S*}, then proceed with the originating update operation.
  - If *R*!fSMORoleOwner ≠ *distinguished name of the nTDSDSA object of the server*, then let *K* = (*R*!fSMORoleOwner)!parent. Return the error *referral* / <*unrestricted*> to *K*!dNSHostName.
  - If IsEffectiveRoleOwner(*R*) = true, proceed with the originating update operation.
  - Otherwise, return the error *busy* / <*unrestricted*>.

3.1.1.5.1.9 Allow Updates Only When They Are Enabled

Originating and replicated updates are only allowed when dc.fEnableUpdates is TRUE. When dc.fEnableUpdates is FALSE, the server returns the error unavailable / ERROR_DS_SHUTTING_DOWN.

3.1.1.5.1.10 Originating Updates Attempted on an RODC

In addition to the constraints described in section 3.1.1.5.1.9, an RODC does not perform originating updates. When an originating update is requested on an RODC, the RODC generates an LDAP referral ([RFC2251] sections 3.2 and 4.1.11) to a DC holding a writable NC replica, as specified in this section. By following the referral, the client can perform the desired update.

Define *O* as follows:

- If the originating update is an add, let *O* be the parent of the object to be added.
- If the originating update is a modify, modify DN, or delete, let *O* be the object to be updated.

If *O* does not exist, return the error *noSuchObject* / *ERROR_DS_OBJ_NOT_FOUND*. Otherwise, let *N* be the NC containing *O*. Using techniques described in section 6.3.6, find a DC *D* that has a writable NC replica for *N*. Generate an LDAP referral to *D* as specified in [RFC2251] section 4.1.11.

3.1.1.5.1.11 Constraints and Processing Specifics Defined Elsewhere

In addition to the constraints and processing specifics defined in the remainder of section 3.1.1.5, update operations MUST conform to the constraints and processing details defined in [MS-SAMR] and [MS-DRSR]. The constraints specified in [MS-SAMR] are enforced only for originating updates.

3.1.1.5.2 Add Operation

References

LDAP attributes: objectClass, nTSecurityDescriptor, instanceType, distinguishedName, objectGUID, objectSid, entryTTL, msDS-Entry-Time-To-Die, systemFlags, msDS-AllowedToDelegateTo, objectCategory, defaultObjectCategory, defaultHidingValue, showInAdvancedViewOnly, msDS-DefaultQuota, msDS-QuotaTrustee, msDS-TombstoneQuotaFactor, subRefs, nCName, Enabled, uSNLastObjRem, uSNDSALastObjRemoved, whenCreated, uSNCreated, replPropertyMetaData, isDeleted, instanceType, proxiedObjectName, msDS-LockoutObservationWindow, msDS-LockoutDuration, msDS-MaximumPasswordAge, msDS-MinimumPasswordAge, msDS-MinimumPasswordLength, msDS-PasswordHistoryLength.

LDAP classes: dynamicObject, crossRef, trustedDomain, secret, classSchema, attributeSchema, msDS-QuotaControl, foreignSecurityPrincipal.

Constants

- Win32/status error codes: ERROR_DS_OBJ_CLASS_NOT_DEFINED, ERROR_DS_ILLEGAL_MOD_OPERATION, ERROR_DS_OBJECT_CLASS_REQUIRED, ERROR_DS_OBJ_CLASS_NOT_SUBCLASS, ERROR_DS_BAD_INSTANCE_TYPE, ERROR_DS_ADD_REPLICA_INHIBITED, ERROR_DS_CANT_ADD_SYSTEM_ONLY, ERROR_DS_CLASS_MUST_BE_CONCRETE, ERROR_DS_BAD_NAME_SYNTAX, ERROR_DS_ATT_NOT_DEF_IN_SCHEMA, ERROR_DS_NOT_SUPPORTED, ERROR_DS_RDN_DOESNT_MATCH_SCHEMA, STATUS_QUOTA_EXCEEDED, ERROR_DS_REFERRAL, ERROR_DS_CROSS_REF_EXISTS, ERROR_DS_RANGE_CONSTRAINT, ERROR_DS_ROLE_NOT_VERIFIED, ERROR_DS_NO_CROSSREF_FOR_NC, ERROR_DS_SPN_VALUE_NOT_UNIQUE_IN_FOREST, ERROR_DS_UPN_VALUE_NOT_UNIQUE_IN_FOREST
- **Access mask** bits, control access rights: RIGHT_DS_CREATE_CHILD, Add-GUID
- Security privileges: SE_ENABLE_DELEGATION_PRIVILEGE
- instanceType flags: IT_NC_HEAD, IT_WRITE, IT_NC_ABOVE
- Generic systemFlags bits: FLAG_CONFIG_ALLOW_RENAME, FLAG_CONFIG_ALLOW_MOVE, FLAG_CONFIG_ALLOW_LIMITED_MOVE
- Schema systemFlags bits: FLAG_ATTR_IS_RDN
- crossRef systemFlags bits: FLAG_CR_NTDS_NC, FLAG_CR_NTDS_DOMAIN, FLAG_CR_NTDS_NOT_GC_REPLICATED

The add operation results in addition of a new object to the directory tree. The requester supplies the following data:

- The DN of the new object.
- The set of attributes defining the new object.

3.1.1.5.2.1 Security Considerations

For regular object creation, the requester must have RIGHT_DS_CREATE_CHILD on the parent object for the objectClass of the object being added.

In the case of Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, in the absence of RIGHT_DS_CREATE_CHILD, computer object creation requires that the **RpcImpersonationAccessToken.Privileges[]** field MUST have the SE_MACHINE_ACCOUNT_NAME privilege (defined in [MS-LSAD] section 3.1.1.2.1).

For application NC creation (see section 3.1.1.5.2.6), the requester must have sufficient permissions to create the crossRef object in the Partitions container on the domain naming FSMO, or to take over an existing crossRef object (in case of pre-created crossRef). See section 3.1.1.5.2.6 for more details.

If the msDS-AllowedToDelegateTo attribute is specified as a part of the add operation, then the requester must possess SE_ENABLE_DELEGATION_PRIVILEGE.

If any attributes being added are marked in the schema as partition secrets (see the SE flag in section 2.2.9), the requester must have the control access right DS-Write-Partition-Secrets on the root object of the naming context to which the modified object belongs.

Access checks are not performed for replicated updates.

3.1.1.5.2.2 Constraints

The following constraints are enforced for originating update Add operations. If any of these constraints are not satisfied, the server returns an error.

These constraints are not enforced for replicated updates.

- The object DN value is a syntactically valid DN (see LDAP, section 3.1.1.3). If it is not, Add returns *namingViolation* / *ERROR_DS_NAME_UNPARSEABLE*.
- If instanceType attribute value is specified, then the following constraints MUST be satisfied:
  - If the DC functional level is DS_BEHAVIOR_WIN2000, then multiple integer values are permitted. However, if the DC functional level is DS_BEHAVIOR_WIN2003 or greater, then there must be exactly one integer value; otherwise Add returns *unwillingToPerform* / *ERROR_DS_BAD_INSTANCE_TYPE*.
  - If the instanceType value has IT_NC_HEAD bit set, then IT_WRITE MUST be set. If this is the case, then this operation is considered to be an *NC-Add* operation, and additional constraints and processing specifics apply (see NC-Add Operation (section 3.1.1.5.2.8) for details).
  - If IT_NC_HEAD is set, but IT_WRITE is not set, Add returns *unwillingToPerform* / *ERROR_DS_ADD_REPLICA_INHIBITED*.
  - If IT_NC_HEAD is not set in the value, and the DC functional level is DS_BEHAVIOR_WIN2003 or greater, then the only allowed values are zero and IT_WRITE; otherwise Add returns *unwillingToPerform* / *ERROR_DS_BAD_INSTANCE_TYPE*.
- If the operation is not *NC-Add*, then the parent object MUST be in an NC whose full replica is hosted at this DC; otherwise *referral* / *ERROR_DS_REFERRAL* is returned.
- If the operation is not *NC-Add*, then the parent object MUST be present in the directory. The parent DN is computed from the passed-in DN value by removing the first RDN label. If the parent object is not found in the directory, then *noSuchObject* / *ERROR_DS_OBJ_NOT_FOUND* is returned.
- At least one objectClass value MUST be specified. Otherwise, Add returns *objectClassViolation* / *ERROR_DS_OBJECT_CLASS_REQUIRED*.
- The objectClass attribute MUST be specified only once in the input attribute list. Otherwise, Add returns *attributeOrValueExists* / *ERROR_DS_ATT_ALREADY_EXISTS* if the DC functional level is DS_BEHAVIOR_WIN2000, and *objectClassViolation* / *ERROR_DS_ILLEGAL_MOD_OPERATION* if the DC functional level is DS_BEHAVIOR_2003 or greater.

- All objectClass values correspond to classes that are defined and active in the schema.
  - If a defunct class is referenced, then Add returns *objectClassViolation* / *ERROR_DS_OBJ_CLASS_NOT_DEFINED* if the DC functional level is DS_BEHAVIOR_2003 or lower, and *noSuchAttribute* / *ERROR_INVALID_PARAMETER* if the DC functional level is DS_BEHAVIOR_WIN2008 or greater.
  - If the objectClass does not exist in the schema, Add returns *noSuchAttribute* / *ERROR_INVALID_PARAMETER*.
- The set of non-auxiliary objectClass values defines a (possibly incomplete) inheritance chain with a single, most specific structural objectClass or a single 88 object class. If this is not true, Add returns *objectClassViolation* / *ERROR_DS_OBJ_CLASS_NOT_SUBCLASS*.
- If the forest functional level is DS_BEHAVIOR_WIN2003 or higher, then auxiliary classes can be included while setting the value for the objectClass attribute. If the forest functional level is lower than DS_BEHAVIOR_WIN2003, then including auxiliary classes while setting the value of the objectClass attribute results in *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED* being returned by the server.
- If the **fschemaUpgradeInProgress** field is false on the *LDAPConnection* instance in dc.ldapConnections ([MS-DRSR] section 5.115) corresponding to the LDAP connection on which the operation is being performed and the structural objectClass or the 88 object class is not marked as systemOnly, then Add returns *unwillingToPerform* / *ERROR_DS_CANT_ADD_SYSTEM_ONLY*.
- The objectClass's objectClassCategory is either 0 (88 object class) or 1 (structural object class). If it is not, Add returns *unwillingToPerform* / *ERROR_DS_CLASS_MUST_BE_CONCRETE*.
- The structural objectClass is not a Local Security Authority (LSA)–specific object class (section 3.1.1.5.2.3). If it is, Add returns *unwillingToPerform* / *ERROR_DS_CANT_ADD_SYSTEM_ONLY*.
- If the structural objectClass is crossRef, then crossRef requirements (section 3.1.1.5.2.7), as well as NC naming requirements (section 3.1.1.5.2.6), are enforced.
- It is disallowed to create objects with duplicate RDN values under the same parent container. See section 3.1.1.3.1.2.1 for more information.
- All attribute names/OIDs refer to attributes that are defined and active in the schema. If an unknown or defunct attribute is referenced, Add returns *noSuchAttribute* / *ERROR_INVALID_PARAMETER*.
- Object quota requirements are satisfied for the requester in the NC where the object is being added (see section 3.1.1.5.2.5).
- The objectClass being created satisfies the possSuperiors schema constraint (section 3.1.1.2) for the objectClass of the parent object. Otherwise, *objectClassViolation* / *ERROR_DS_ILLEGAL_SUPERIOR* is returned if the DC functional level is DS_BEHAVIOR_WIN2000, and *namingViolation* / *ERROR_DS_ILLEGAL_SUPERIOR* is returned if the DC functional level is DS_BEHAVIOR_WIN2003 or greater.
- The set of attributes provided for object creation is consistent with the schema as described in section 3.1.1.5.1.1.
- If the requester has supplied a value for the RDN attribute, then it matches the first label of the supplied DN value in both attribute type and attribute value, according to the LDAP Unicode string comparison rules in section 3.1.1.3.

- The RDN value satisfies schema constraints (rangeLower/rangeUpper, single-valuedness, syntax, and so on).
- If a **site object** is being created, then the RDN value is a valid DNS name label (according to the DNS RFC [RFC1035]). Otherwise, *invalidDNSyntax* / *ERROR_DS_BAD_NAME_SYNTAX* is returned.
- If a subnet **object** is being created, then the RDN value MUST be a valid subnet object name, according to the algorithm described in section 6.1.1.2.2.2.1. Otherwise, *invalidDNSyntax* / *ERROR_DS_BAD_NAME_SYNTAX* is returned.
- In the following two cases, the requester specifies the objectGUID or the objectSid during Add:
  - The requester is allowed to specify the objectGUID if the following four conditions are all satisfied:
    - The fSpecifyGUIDOnAdd heuristic is true in the dSHeuristics attribute (see section 6.1.1.2.4.1.2).
    - The requester has the Add-GUID control access right (section 5.1.3.2.1) on the NC root of the NC where the object is being added.
    - The requester-specified objectGUID is not currently in use in the forest.
    - Active Directory is operating as AD DS.
  - The requester is required to specify the objectSid when creating a bind proxy object (section 3.1.1.8.2) in an AD LDS NC. The objectSid value specified for a bind proxy object must be resolvable by the machine running the AD LDS DC to an active Windows user. If the SID cannot be resolved to an active Windows user, Add returns *unwillingToPerform* / *ERROR_DS_SECURITY_ILLEGAL_MODIFY*. If the requester-specified objectSid value is present on an **existing object** in the same NC, Add returns *unwillingToPerform* / *ERROR_DS_SECURITY_ILLEGAL_MODIFY*.

  In all other cases, it is an error (*unwillingToPerform* / *ERROR_DS_SECURITY_ILLEGAL_MODIFY*) for the requester to specify the objectGUID or objectSid during Add; these values are automatically generated (as specified in section 3.1.1.5.2.4, "Processing Specifics") by the system as required.
- If the requester has specified an owner using the LDAP_SERVER_SET_OWNER_OID LDAP control and has specified a value for the nTSecurityDescriptor, the owner in the security descriptor is set to the owner supplied by the control. Any other portions of the security descriptor are unchanged. The resultant value is a valid security descriptor value in self-relative format, and it satisfies the security descriptor constraints (see "Security Descriptor Requirements" in section 6.1.3).
- If the requester has specified an owner using the LDAP_SERVER_SET_OWNER_OID LDAP control but has not specified a value for nTSecurityDescriptor, a new value for nTSecurityDescriptor is created: a security descriptor with the owner set to the owner supplied by the control. No other portions of the security descriptor are valid. The resultant value is a valid security descriptor value in self-relative format, and it satisfies the security descriptor constraints (see "Security Descriptor Requirements" in section 6.1.3).
- If the requester has not specified an owner using the LDAP_SERVER_SET_OWNER_OID LDAP control but has specified a value for nTSecurityDescriptor, the value is a valid security descriptor value in self-relative format, and it satisfies the security descriptor constraints (see "Security Descriptor Requirements" in section 6.1.3).

- If the requester has specified a value for the objectCategory attribute, then it points to an existing classSchema object in the schema container.
- If the requester has specified a value for the servicePrincipalName attribute, then it is a syntactically valid SPN value (see section 5.1.1.4, "Mutual Authentication").
- If the requester has specified values for the servicePrincipalName or userPrincipalName attributes, those values must meet the constraints specified in section 3.1.1.5.1.3.
- If the DC functional level is DS_BEHAVIOR_WIN2003 or greater and the msDS-Entry-Time-To-Die attribute is set, then the objectClass value includes the dynamicObject auxiliary class.
- If the DC functional level is DS_BEHAVIOR_WIN2003 or greater, then it is disallowed for a non-dynamicObject child to be created under a dynamicObject parent (see section 6.1.7). If this constraint is violated, then *unwillingToPerform* / *ERROR_DS_UNWILLING_TO_PERFORM* is returned.
- If the DC functional level is DS_BEHAVIOR_WIN2008 or greater, the following constraints are enforced on objects of class msDS-PasswordSettings:
  - The msDS-PasswordHistoryLength attribute is less than or equal to 1024.
  - The msDS-MinimumPasswordAge attribute is less than or equal to 0.
  - The msDS-MaximumPasswordAge attribute is less than or equal to 0.
  - The msDS-MaximumPasswordAge attribute is less than the value of the msDS-MinimumPasswordAge attribute on the same object after the Add would have completed.
  - The msDS-MinimumPasswordLength attribute is less than or equal to 256.
  - The msDS-LockoutDuration attribute is less than or equal to 0.
  - The msDS-LockoutObservationWindow attribute is less than or equal to 0.
  - The msDS-LockoutDuration attribute is less than or equal to the value of the msDS-LockoutObservationWindow attribute on the same object after the Add would have completed.

  Otherwise, *unwillingToPerform* / *ERROR_DS_SECURITY_ILLEGAL_MODIFY* is returned.
- An AD LDS security principal object (section 5.1.1.5) can be created in an application NC. In addition, if the ADAMAllowADAMSecurityPrincipalsInConfigPartition configurable setting (section 3.1.1.3.4.7) is supported and equals 1, an AD LDS security principal object can also be created in the config NC. An AD LDS security principal object can never be created in the schema NC.
- In AD LDS, if the LDAP policy ADAMDisablePasswordPolicies does not equal 1, and a password value (either unicodePwd or userPassword) is specified in an Add, the password must satisfy the current password policy in effect on the AD LDS server as reported by SamrValidatePassword ([MS-SAMR] section 3.1.5.13.7). If the provided password value does not satisfy the password policy, the Add returns *constraintViolation* / *ERROR_PASSWORD_RESTRICTION*.
- In AD LDS, if the fAllowPasswordOperationsOverNonSecureConnection heuristic of the dSHeuristics attribute (see section 6.1.1.2.4.1.2) is not true, and a password value (either unicodePwd or userPassword) is specified in an Add, the LDAP connection must be encrypted with cipher strength of at least 128 bits. If the connection does not pass the test, the Add returns *operationsError* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.

- In AD LDS, if the userPrincipalName value is specified in an Add, then the value must be unique within all NCs on this DC. If another object exists with the same userPrincipalName value, the Add returns *attributeOrValueExists* / *ERROR_DS_NAME_NOT_UNIQUE*.
- In AD LDS, the following attributes are disallowed in an Add: badPwdCount, badPasswordTime, lastLogonTimestamp, pwdLastSet. If one of these attributes is specified in an add, the Add returns *constraintViolation* / *ERROR_DS_ATTRIBUTE_OWNED_BY_SAM*.
- In AD DS, the following attributes are disallowed in an Add for objects of class user: badPasswordTime, badPwdCount, dBCSPwd, isCriticalSystemObject, lastLogoff, lastLogon, lastLogonTimestamp, lmPwdHistory, logonCount, memberOf, msDS-User-Account-Control-Computed, ntPwdHistory, objectSid, rid, sAMAccountType, and supplementalCredentials. If one of these attributes is specified in an Add, the Add returns *unwillingToPerform* / *ERROR_DS_ATTRIBUTE_OWNED_BY_SAM*.
- In AD DS, the following attributes are disallowed in an Add for objects of class group: isCriticalSystemObject, memberOf, objectSid, rid, sAMAccountType, and userPassword. If one of these attributes is specified in an Add, the Add returns *unwillingToPerform* / *ERROR_DS_ATTRIBUTE_OWNED_BY_SAM*.
- In AD DS, the following attributes are disallowed in an Add for an object whose class is not a SAM-specific object class (see 3.1.1.5.2.3): isCriticalSystemObject, lmPwdHistory, ntPwdHistory, objectSid, samAccountName, sAMAccountType, supplementalCredentials, and unicodePwd. If one of these attributes is specified in an Add, the Add returns *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- Additional constraints are enforced if the object being created is a SAM-specific object (section 3.1.1.5.2.3); [MS-SAMR] section 3.1.1.6 specifies these constraints.
- Additional constraints are enforced if the object being created is a schema object (section 3.1.1.5.2.3). See section 3.1.1.2, "Active Directory Schema", for more details.
- In the case of Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, if the object being created is a **computer object** and all of the following conditions hold true:
  - The requester does not have RIGHT_DS_CREATE_CHILD access on the Container-Object object.
  - The **RpcImpersonationAccessToken.Privileges[]** field has the SE_MACHINE_ACCOUNT_NAME privilege (defined in [MS-LSAD] section 3.1.1.2.1).

  Then these constraints apply:

  - Following is the list of allowed and required attributes that must be specified:
    - dNSHostName
    - servicePrincipalName
    - userAccountControl
    - unicodePwd*
    - objectClass
    - sAMAccountName

*If the account is created with UF_ACCOUNTDISABLE set in userAccountControl, unicodePwd is not required.

- Iterate over the list of attributes specified in the request:
  - If the attribute is not in the preceding list of required attributes, the Add returns ERROR_DS_MISSING_REQUIRED_ATT.
  - If the attribute is userAccountControl and the UF_WORKSTATION_TRUST_ACCOUNT bit is not set or any bit other than UF_WORKSTATION_TRUST_ACCOUNT | UF_ACCOUNTDISABLE is set, Add returns ERROR_DS_SECURITY_ILLEGAL_MODIFY.
  - If the attribute is unicodePwd and the value is of zero length and userAccountControl is either not in the list of attributes in the request or is present but the bit UF_ACCOUNTDISABLE is not set, Add returns ERROR_DS_SECURITY_ILLEGAL_MODIFY.
  - If the attribute unicodePwd is not found in the request and the UF_ACCOUNTDISABLE bit is not set in userAccountControl, the Add returns ERROR_DS_MISSING_REQUIRED_ATT.

3.1.1.5.2.3 Special Classes and Attributes

This section defines three sets of object classes: LSA-specific object classes, SAM-specific object classes, and schema object classes. These sets are mentioned elsewhere in the specification, because special processing is applied to instances of these classes.

Each set includes both the specific object classes mentioned here and any subclasses of these object classes.

- LSA-specific object classes: secret, trustedDomain (originating updates only, in AD DS only).
- SAM-specific object classes: group, samDomain, samServer, user (originating updates only, in AD DS only).
- Schema object classes: attributeSchema, classSchema (originating and replicated updates).

This section also defines one set of attributes: **foreign principal object (FPO)**-enabled attributes. This set is mentioned elsewhere in the specification, because special processing is applied to instances of these attributes.

- FPO-enabled attributes: member, msDS-MembersForAzRole, msDS-NeverRevealGroup, msDS-NonMembers, msDS-RevealOnDemandGroup, msDS-ServiceAccount.

3.1.1.5.2.4 Processing Specifics

- For originating updates, a new objectGUID value is generated and set on the object. For replicated updates, the received objectGUID is set on the object.
- In AD DS, if the object is a security principal (according to its objectClass values), then for originating updates the objectSid value is generated and set on the object (see [MS-SAMR] sections 3.1.1.6 and 3.1.1.9). For replicated updates, the received objectSid is set on the object.
- In AD LDS, if the object being added is an NC root and not the schema NC root, then it is given an objectSid value, ignoring schema constraints. The objectSid value ([MS-DTYP] section 2.4.2), with one SubAuthority value, is generated using the following algorithm:

- The IdentifierAuthority value (6 bytes) is generated as follows: the first 2 bytes are zero, the high 4 bits of the third byte are 0001, and the remaining 3.5 bytes (the lower 4 bits of the third byte, and bytes 4, 5 and 6) are randomly generated.
- The first SubAuthority value (**DWORD**) is randomly generated.

- In AD LDS, if the object being added is an AD LDS security principal object (an object that is not an NC root and contains the objectSid attribute), then the objectSid value is generated using the following algorithm, which produces a SID with 5 SubAuthority values:
  - The Revision byte is 1.
  - The SubAuthorityCount is 5.
  - The IdentifierAuthority is set to the same value as the IdentifierAuthority of the SID of the NC root.
  - The first SubAuthority is set to the same value as the first SubAuthority of the SID of the NC root.
  - A randomly generated GUID value (16 bytes or 4 DWORDs) is taken as second, third, fourth, and fifth SubAuthority values of the new SID value. This GUID value is unrelated to the objectGUID value that is also generated randomly for the object being added.
- In AD LDS, if a group object is being created (that is, an object containing the value group in its objectClass), and the groupType attribute is not specified, then the following value is assigned to groupType: GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED.
- In AD LDS, if an AD LDS user is being created, and the password value (either unicodePwd or userPassword) was not supplied, then the password value is defaulted to an empty string.
- In AD LDS, if an AD LDS user is being created, and the password value is defaulted and does not satisfy the password policy in effect on the AD LDS server (as reported by SamrValidatePassword, [MS-SAMR] section 3.1.5.13.7), then the user is created in the disabled state; that is, msDS-UserAccountDisabled = true. However, if the Add operation specifies the msDS-UserAccountDisabled attribute with the value of false, the add returns *constraintViolation / ERROR_PASSWORD_RESTRICTION*. This processing rule is not effective if the LDAP policy ADAMDisablePasswordPolicies is equal to 1.
- In AD LDS, if an AD LDS user is being created, then badPwdCount and badPasswordTime values are set to zero.
- The nTSecurityDescriptor value is computed and set on the object (see section 6.1.3 for more details).
- Any values specified for attributes that are marked as constructed in the schema are ignored, with one exception: the entryTTL attribute.
- If the value of the entryTTL attribute is specified in the Add request, it is processed as follows:
  - If the value of the entryTTL attribute is less than the DynamicObjectMinTTL LDAP setting, then the entryTTL attribute is set to the value of the DynamicObjectMinTTL setting.
  - The current system time, plus the entryTTL attribute interpreted as seconds, is written into the msDS-Entry-Time-To-Die attribute.

- If dynamicObject is present among objectClass values, but neither entryTTL nor msDS-Entry-Time-To-Die were specified in an originating update, then Add proceeds as if the value of the DynamicObjectDefaultTTL LDAP policy had been specified as the value of the entryTTL attribute.
- Any values specified by the requester for the following attributes are ignored: distinguishedName, subRefs, uSNLastObjRem, uSNDSALastObjRemoved, uSNCreated, replPropertyMetaData, isDeleted, proxiedObjectName.
- For an originating update, any value specified for the whenCreated attribute is ignored and its value is set to the current time according to the system clock on this DC.
- If a value of the systemFlags attribute is specified by the requester, the DC removes any flags not listed below from the systemFlags value before storing it on the new object:
  - FLAG_CONFIG_ALLOW_RENAME
  - FLAG_CONFIG_ALLOW_MOVE
  - FLAG_CONFIG_ALLOW_LIMITED_MOVE
  - FLAG_ATTR_IS_RDN (removed unless the object is an attributeSchema object)
- For the following scenarios, the DC sets additional bits in the systemFlags value of the object created:
  - server objects: FLAG_DISALLOW_MOVE_ON_DELETE, FLAG_CONFIG_ALLOW_RENAME, and FLAG_CONFIG_ALLOW_LIMITED_MOVE.
  - serversContainer and nTDSDSA objects: FLAG_DISALLOW_MOVE_ON_DELETE.
  - site object: FLAG_DISALLOW_MOVE_ON_DELETE and FLAG_CONFIG_ALLOW_RENAME.
  - siteLink, siteLinkBridge, and nTDSConnection objects: FLAG_CONFIG_ALLOW_RENAME.
  - Any object that is not mentioned above and whose parent is the Subnets Container (section 6.1.1.2.2.2): FLAG_CONFIG_ALLOW_RENAME.
  - Any object that is not mentioned above and whose parent is the Sites Container (section 6.1.1.2.2) except the Subnets Container (section 6.1.1.2.2.2) and the Inter-Site-Transports Container (section 6.1.1.2.2.3): FLAG_CONFIG_ALLOW_RENAME.
- If a value for the objectCategory attribute was not specified by the requester, then it is defaulted to the current value of the defaultObjectCategory attribute on the classSchema object corresponding to the 88 object class or the most specific structural object class of the object being added.
- The complete inheritance chain of object classes (starting from the most specific structural object class or 88 object class as well as from all dynamic auxiliary classes specified by the user) is computed and set. The correct ordering of objectClass values is performed (see section 3.1.1.2.4.3 for more details).
- The value of instanceType attribute is written. For originating updates of regular objects, it is IT_WRITE. For NC root object specifics, see NC-Add Operation (section 3.1.1.5.2.8). For replicated updates, the instanceType value computed by the IDL_DRSGetNCChanges client is written.
- distinguishedName attribute is written, matching the DN value of the supplied object.

- The RDN attribute of the correct attribute type is written, as computed from the DN value of the supplied object.
- If the showInAdvancedViewOnly value was not provided by the requester and the defaultHidingValue of the objectClass is true, then the showInAdvancedViewOnly attribute value is set to true.
- If the Add assigns a value to an FPO-enabled attribute (section 3.1.1.5.2.3) of the new object, and the DN value in the add request has <SID=stringizedSid> format (section 3.1.1.3.1.2.4), then the DC creates a corresponding foreignSecurityPrincipal object in the ForeignSecurityPrincipals container (section 6.1.1.4.10) and assigns a reference to the new foreignSecurityPrincipal object as the FPO-enabled attribute value. [MS-SAMR] section 3.1.1.8.9 specifies the creation of the foreignSecurityPrincipal object.
- If attributeSchema or classSchema object is created in schema NC, then apply special processing as described in section 3.1.1.2.5.
- If an infrastructureUpdate object is created, then let O be the object that is created. If (O!dNReferenceUpdate has a value), then for each object P in each NC replica on the server, do the following:
  - Let S be the set of all attributes of P with attribute syntax Object(DS-DN), Object(DN-String), Object(DN-Binary), Object(OR-Name), or Object(Access-Point).
  - For each attribute A in set S and for each value V of A, do the following:
    - If the attribute syntax of A is Object(DS-DN), then let G be P.A.guid_value.
    - Otherwise, let G be P.A.V.object_DN.guid_value.
    - Let RG be O!dNReferenceUpdate.guid_value.
    - Let RD be O!dNReferenceUpdate.dn.
    - If (RG = G), then delete V from P.A.
    - If (RG = G) and A is not a link value attribute, then add attribute value of O!dNReferenceUpdate to P.A
    - If (RG = G) and A is a link value attribute and RDN of RD is not a delete-mangled RDN (see section 3.1.1.5.5), then add value of O!dNReferenceUpdate to P.A.
    - If (RG = G) and A is a link value attribute and RDN of RD is a delete-mangled RDN (see section 3.1.1.5.5) and the Recycle Bin optional feature is enabled (see section 3.1.1.9.1), then add the value of O!dNReferenceUpdate to P.A. However, this value is to be treated as a linked value to or from a deleted-object. That is, the value is not generally visible to LDAP clients unless the LDAP_SHOW_DEACTIVATED_LINK_OID control is used.
- If a crossRef object is being created, the server MUST return ERROR_DS_ROLE_NOT_VERIFIED if the IsEffectiveRoleOwner(RoleObject(Config NC, DomainNamingMasterRole)) function specified in section 3.1.1.5.1.8 returns FALSE.

3.1.1.5.2.5 Quota Calculation

Quotas control the number of objects (including **tombstones**, **deleted-objects**, and **recycled-objects**) that a security principal may own within an NC. A security principal is considered the "owner" of an object if the OWNER field in the object's nTSecurityDescriptor value equals the

security principal's SID. In the event the object owner changes, the quota (USAGE) for the existing and potential new owner is recalculated.

The quota is not enforced in two cases:

- When the requester of an operation is not the same as the potential owner.
- When the requester has specified the LDAP_SERVER_BYPASS_QUOTA_OID control and has been granted the control access right DS-Bypass-Quota on the object that is the root of the NC in which the operation is to be performed.

When a quota is enforced, the USAGE value for the requester is computed. When the USAGE value computed for a requester exceeds their MAX-USAGE value (see below), add, undelete (reanimation), delete, and change-of-owner operations are prevented for the requester and the server returns the *adminLimitExceeded* / *STATUS_QUOTA_EXCEEDED* error.

The USAGE value is computed as follows:

USAGE = *owned_existing_objects* + ceil(*tombstone-factor*/100 * *owned_deleted_objects*)

In the preceding formula, *owned_existing_objects* is the total number of existing-objects that the requester owns. *owned_deleted_objects* is the total number of tombstones, deleted-objects, or recycled-objects (see the Delete operation in section 3.1.1.5.5) that the requester owns. *tombstone-factor* is the integer value stored in the msDS-TombstoneQuotaFactor attribute on the Quotas container in the NC. Ceil() is the "ceiling" mathematical function.

The MAX-USAGE value is computed as follows:

1. A set of applicable msDS-QuotaControl objects in the Quotas container is obtained. An msDS-QuotaControl object is applicable for the requester if its msDS-QuotaTrustee attribute contains a SID that is present in the requester's authorization information.
2. If the set of applicable msDS-QuotaControl objects is non-empty, then the maximum value of the msDS-QuotaAmount attribute is chosen as the MAX-USAGE value.
3. If the set of applicable msDS-QuotaControl objects is empty, then the value of the msDS-DefaultQuota attribute on the Quotas container is chosen as the MAX-USAGE value.

3.1.1.5.2.6 NC Requirements

The following requirements apply to DN s of AD DS NCs (the set of NCs that are parts of the Active Directory forest) other than the config NC and schema NC:

- Each RDN label within the DN has the DC= type.
- Each RDN label within the DN has a value, which is a valid DNS name label.

The following requirements apply to DN s of all Active Directory NCs:

- The full DN of the NC does not match the DN of another existing object in an Active Directory NC.
- If the immediate parent of the NC is not an Active Directory NC, then none of the ancestors (grandparent, grand-grandparent, and so on) are an Active Directory NC. In other words, the set of Active Directory NCs is a set of nonintersecting trees, and each tree does not have "holes".

The following requirements apply to the data stored in NC roots:

- IT_NC_HEAD bit is set in the instanceType attribute.

- If the NC has an immediate parent (which must be an NC root per the preceding rules), then IT_NC_ABOVE bit is be set in its instanceType attribute.
- If the NC has child NCs, then their DNs are listed in its subRefs attribute.

If any server has a replica of the NC and of an NC C, which is a child of the NC, then the NC root of C is the **subordinate reference object** of C. If the server does not have a replica of C, then an object o is present in the server and satisfies the following requirements, and o is the subordinate reference object of C.

- The IT_NC_HEAD bit is set in the instanceType attribute.
- The IT_NC_ABOVE bit is set in the instanceType attribute.
- The IT_UNINSTANT bit is set in the instanceType attribute.
- Object o has the same distinguishedName and objectGUID as the child NC root object.

Object o is not exposed through the LDAP protocol. For information about the replication of subordinate reference objects, see [MS-DRSR] sections 4.1.1.2.2, 4.1.20.2, 5.6, and 5.32.

The default structure of data in NCs is covered in Naming Contexts in section 6.1.1.1.

3.1.1.5.2.7 crossRef Requirements

crossRef objects represent NCs within the Active Directory forest, as well as "external" (foreign) NCs. The relationship between the crossRef and the NC is represented by the nCName attribute on the crossRef. The value of this attribute is the DN of the corresponding NC. Each Active Directory NC has a corresponding crossRef object. A crossRef object can also represent an intention to create a new Active Directory NC with the specified DN.

The following requirements apply to crossRef objects:

- The FLAG_CR_NTDS_NC bit is set in systemFlags if and only if the nCName represents an Active Directory NC.
- The FLAG_CR_NTDS_DOMAIN bit is set in systemFlags if and only if the nCName represents a domain Active Directory NC.
- The FLAG_CR_NTDS_NOT_GC_REPLICATED bit is set in systemFlags if and only if the nCName represents an Application Active Directory NC.
- If the FLAG_CR_NTDS_NC bit is set in systemFlags and the Enabled attribute value is false, then the crossRef represents an intention to create an Active Directory NC. Otherwise, it represents an Active Directory NC that is actually present.

3.1.1.5.2.8 NC-Add Operation

For originating updates, the NC-Add operation is distinguished by the presence of instanceType attribute with (IT_NC_HEAD | IT_WRITE) value in the input attribute set. For replicated updates, the NC-Add operation is distinguished by the presence of instanceType attribute with IT_NC_HEAD value in the input attribute set. The DN of the object represents the new NC DN, and the DC enforces the constraints on NC naming described previously.

For originating updates, the NC-Add operation is only supported for application NCs. If a new domain NC needs to be created, then IDL_DRSAddEntry RPC MUST be used to create the crossRef (see [MS-DRSR] section 4.1.1).

3.1.1.5.2.8.1 Constraints

Regular Add operation constraints apply to the NC-Add operation (as defined in previous sections), with the exception of constraints pertaining to the parent object (for example, the possSuperiors schema constraint).

There are two distinct NC-Add scenarios that are supported with regard to maintaining crossRef requirements:

1. The crossRef corresponding to the new NC does not exist. In this case, a new crossRef object is created. If the DC is the domain naming FSMO, then the crossRef is created locally. Otherwise, the crossRef is created on the domain naming FSMO DC using the IDL_DRSAddEntry call with appropriate parameters (see [MS-DRSR] section 4.1.1 for details).

2. The crossRef corresponding to the new NC has been pre-created (that is, it was created previously). The crossRef object is located finding the object where the value of nCName matches the DN of the NC being created. Once located, the following constraints on the crossRef are validated:

    1. If Enabled is true, the server MUST return ERROR_DS_CROSS_REF_EXISTS.

    2. If the dnsRoot attribute value does not match the dnsName of the DC processing the NC-Add operation, the server MUST return ERROR_DS_MASTERDSA_REQUIRED.

3.1.1.5.2.8.2 Security Considerations

Regular Add access checks do not apply to the NC-Add operation, because the parent object may not even exist in the directory. Instead, the requester must have sufficient permissions to either create a new crossRef or modify the pre-created crossRef object. Regular Add and modify permission checks apply for these operations.

No access check is performed for replicated updates.

3.1.1.5.2.8.3 Processing Specifics

The following operations are performed during an NC-Add operation performed as an originating update:

- The matching crossRef object is obtained (see details in section 3.1.1.5.2.8.1).
- The NC root object is created per the Add request. Regular Add processing applies (as defined in sections 3.1.1.5.2.1 through 3.1.1.5.2.3).
- The default NC tree structure is generated (see Naming Contexts in section 6.1.1.1), and the appropriate wellKnownObjects references are written on the NC root.
- The matching crossRef object is updated as follows: the Enabled attribute is removed, and the dnsRoot is updated to contain the full DNS name of the NC, as computed from the NC DN.
- If the NC being created is child of an NC P, and the server in which the NC is being created has a replica of P, then the new NC root will be the subordinate reference object to the new NC and must be listed in the subRefs attribute of P's NC root. For more information about subordinate reference objects, see section 3.1.1.5.2.6.

These steps are not performed for replicated updates.

3.1.1.5.3 Modify Operation

References

LDAP attributes: objectClass, nTSecurityDescriptor, instanceType, distinguishedName, objectGUID, objectSid, entryTTL, msDS-Entry-Time-To-Die, systemFlags, objectCategory, msDS-AllowedToDelegateTo, member, sAMAccountName, msDS-AdditionalSamAccountName, dNSHostName, msDS-AdditionalDnsHostName, servicePrincipalName, uSNCreated, subRefs, uSNLastObjRem, uSNDSALastObjRemoved, name, isDeleted, isRecycled, hasMasterNCs, msDS-hasMasterNCs, hasPartialReplicaNCs, msDS-hasFullReplicaNCs, whenCreated, managedBy, msDS-LockoutObservationWindow, msDS-LockoutDuration, msDS-MaximumPasswordAge, msDS-MinimumPasswordAge, msDS-MinimumPasswordLength, msDS-PasswordHistoryLength.

LDAP classes: dynamicObject, crossRef, server, computer, foreignSecurityPrincipal.

Well-known object GUIDs: GUID_USERS_CONTAINER_W, GUID_COMPUTERS_CONTAINER_W.

Constants

- Win32/status error codes: ERROR_DS_REFERRAL, ERROR_DS_WKO_CONTAINER_CANNOT_BE_SPECIAL, ERROR_DS_CONFIDENTIALITY_REQUIRED, ERROR_DS_ILLEGAL_MOD_OPERATION, ERROR_DS_RANGE_CONSTRAINT, ERROR_DS_HIGH_DSA_VERSION, ERROR_DS_SPN_VALUE_NOT_UNIQUE_IN_FOREST, ERROR_DS_UPN_VALUE_NOT_UNIQUE_IN_FOREST.
- Access mask bits, control access rights: RIGHT_DS_WRITE_PROPERTY, RIGHT_DS_WRITE_PROPERTY_EXTENDED, Change-Infrastructure-Master, Change-Schema-Master, Change-Rid-Master, Change-PDC, Change-Domain-Master, Reanimate-Tombstones.
- Security privileges: SE_ENABLE_DELEGATION_PRIVILEGE
- systemFlags bits: FLAG_DISALLOW_DELETE, FLAG_DOMAIN_DISALLOW_RENAME, FLAG_DOMAIN_DISALLOW_MOVE, FLAG_ATTR_IS_RDN.
- LDAP: LDAP_SERVER_PERMISSIVE_MODIFY_OID

The modify operation results in modification of a single existing object in the directory tree. The requester supplies the following data:

- The DN of the object.
- The set of attributes defining the modifications that should be performed.

3.1.1.5.3.1 Security Considerations

For originating updates, the following access checks are performed. No access checks are performed for replicated updates.

The requester needs to have RIGHT_DS_WRITE_PROPERTY access to all attributes being directly affected by the modify operation. Note that some attributes may be modified indirectly as a result of triggers and processing rules. The requester is not required to have write access to those attributes.

If any attributes being directly modified are marked in the schema as partition secrets (see the SE flag in section 2.2.9), the requester must have the control access right DS-Write-Partition-Secrets on the root object of the naming context to which the modified object belongs.

Additional access checks may apply if the nTSecurityDescriptor value is being modified. See "Security Descriptor Requirements", section 6.1.3, for more details.

If the modify operation represents an Undelete operation, then additional security checks apply (see the Undelete operation in section 3.1.1.5.3.7).

If the msDS-AllowedToDelegateTo attribute is modified, then the requester must possess SE_ENABLE_DELEGATION_PRIVILEGE.

In AD LDS, if a password value is being modified as a password change operation, then the requester needs to have the User-Change-Password control access right on the object being modified. A password change operation is defined as removing the old password value and adding the new password value, where the old password value matches the current password on the object.

In AD LDS, if a password value is being modified as a password reset operation, then the requester needs to have the User-Force-Change-Password control access right on the object being modified. A password reset operation is defined as a replace operation on the password attribute.

In AD LDS, if a password unexpire operation is being performed, then the requester needs to have the Unexpire-Password control access right on the object being modified. A password unexpire operation is defined as setting the pwdLastSet attribute to the value -1.

3.1.1.5.3.1.1 Validated Writes

In some cases, when the requester does not have RIGHT_DS_WRITE_PROPERTY access on an attribute, but has RIGHT_DS_WRITE_PROPERTY_EXTENDED access (also called "validated write"), then the write is allowed, subject to additional constraints for the attribute value. The following subsections specify the additional checks that are performed for validated writes of the specified attributes.

See section 5.1.3.2.2 for the validated write rights GUIDs.

3.1.1.5.3.1.1.1 Member

The operation is either add value or remove value, and the value is the DN of the user object representing the requester. In other words, it is allowed that one can add/remove oneself to and from a group.

The requester must have the Self-Membership validated write right.

3.1.1.5.3.1.1.2 dNSHostName

The object has class computer or server (or a subclass of computer or server).

In AD DS, the value of the dNSHostName attribute being written is in the following format: *computerName.fullDomainDnsName*, where *computerName* is the current sAMAccountName of the object (without the final "$" character), and the *fullDomainDnsName* is the DNS name of the domain NC or one of the values of msDS-AllowedDNSSuffixes on the domain NC (if any) where the object that is being modified is located.

The requester must have the Validated-DNS-Host-Name validated write right.

3.1.1.5.3.1.1.3 msDS-AdditionalDnsHostName

The functional level of the DC on which the modification is taking place is at least DS_BEHAVIOR_WIN2012.

The object has class computer or server (or a subclass of computer or server).

In AD DS, the value of the msDS-AdditionalDnsHostName attribute being written is in the following format: *anyDnsLabel.suffix*, where *anyDnsLabel* is a valid DNS name label, and *suffix* matches one of the values of msDS-AllowedDNSSuffixes on the domain NC root (if any).

The requester must have the Validated-MS-DS-Additional-DNS-Host-Name validated write right.

3.1.1.5.3.1.1.4 servicePrincipalName

The object has class computer (or a subclass of computer).

In AD DS, the servicePrincipalName value satisfies the following constraints:

- The SPN is a syntactically correct two-part SPN, or it is a syntactically correct three-part SPN (see Mutual Authentication (section 5.1.1.4)) and the object is a DC's domain controller object (see sections 6.1.1.3.1 and 6.1.1.3.2).
- One of the following constraints:
  - The instance name matches one of the following: the dNSHostName of the machine, the sAMAccountName of the machine (without the terminating "$"), one of the msDS-AdditionalDnsHostName, or one of the msDS-AdditionalSamAccountName (without the terminating "$").
  - The object has class msDS-ManagedServiceAccount (or a subclass of msDS-ManagedServiceAccount), the domain behavior version is at least DS_BEHAVIOR_WIN2008R2, and the instance name matches one of the following: the dNSHostName, the sAMAccountName (without the terminating "$"), one of the msDS-AdditionalDnsHostName, or one of the msDS-AdditionalSamAccountName (without the terminating "$"), of an object that is referenced by the msDS-HostServiceAccountBL attribute on the object.
  - The SPN is a two-part SPN, and the service name is of the form <guid>._msdcs.<fqdn>, where <guid> is the objectGUID of the domain controller, and <fqdn> matches the msDS-DnsRootAlias of a crossRef object representing the forest.
- The SPN is a three-part SPN and the service name matches one of the following constraints:
  - The service class is "GC" and the service name matches one of the following: the dnsRoot, or the msDS-DnsRootAlias of the crossRef object representing the forest root domain NC.
  - The service class is "ldap" and the service name matches one of the following: the NetBIOSName, the dnsRoot, or the msDS-DnsRootAlias of a crossRef object representing the domain NCor one of the application NCs hosted by the DC.

The requester must have the Validated-SPN validated write right.

3.1.1.5.3.1.1.5 msDS-Behavior-Version

The functional level of the DC on which the modification is taking place is at least DS_BEHAVIOR_WIN2012.

The object is an nTDSDSA object.

The DC that the object represents is an RODC.

The object's parent is a server object.

The computer object specified by the serverReference attribute of the server object that is the parent of the object being modified represents the requester. In other words, it is allowed that an RODC itself can update the msDS-Behavior-Version attribute of its nTDSDSA object on a writable DC.

The requester must have the Validated-MS-DS-Behavior-Version validated write right.

3.1.1.5.3.1.2 FSMO Changes

If a write to the fSMORoleOwner attribute is performed, and the objectClass of the object being modified is one of the following classes, then the requester is required to have an additional control access right on the object. The following control access rights are checked, depending on the objectClass of the object being modified:

- infrastructureUpdate (domain infrastructure master FSMO, in AD DS only): Change-Infrastructure-Master
- dMD (schema FSMO): Change-Schema-Master
- rIDManager (domain RID FSMO, in AD DS only): Change-Rid-Master
- domainDNS (PDC emulator FSMO, in AD DS only): Change-PDC
- crossRefContainer (domain naming FSMO): Change-Domain-Master

3.1.1.5.3.2 Constraints

The following constraints are enforced for a modify operation performed as an originating update. These constraints are not enforced for replicated updates.

- The object resides in a writable NC replica; otherwise the modify returns *referral* / *ERROR_DS_REFERRAL*.
- In AD DS, if the object being modified is in the config NC or schema NC, and the RM control ([MS-DTYP] section 2.4.6) of the SD is present and contains the SECURITY_PRIVATE_OBJECT bit (section 6.1.3), the DC requires one of the following two conditions to be true:
  - The DC is a member of the root domain in the forest.
  - The DC is a member of the same domain to which the current object owner belongs.

  If neither condition is true, the modify returns *referral* / *ERROR_DS_REFERRAL*.
- If a **LostAndFound container** is being modified, the modify returns *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- If the **fschemaUpgradeInProgress** field is false on the LDAPConnection instance in dc.ldapConnections ([MS-DRSR] section 5.115) corresponding to the LDAP connection on which the operation is being performed and the object being modified has class subSchema, then only nTSecurityDescriptor modifications are allowed; otherwise, *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned.
- Modifying an object with isDeleted = true is allowed only if one of the following conditions is true:
  - The Recycle Bin optional feature is not enabled and the operation is an undelete operation. Note that the undelete operation is a special case of the modify operation. See section

3.1.1.9.1 for more details on the Recycle Bin optional feature. See section 3.1.1.5.3.7 for more details on the undelete operation.

- The Recycle Bin optional feature is enabled, the object does not have isRecycled = true, and the operation is an undelete operation. Note that the undelete operation is a special case of the modify operation. See section 3.1.1.9.1 for more details on the Recycle Bin optional feature. See section 3.1.1.5.3.7 for more details on the undelete operation.
- The object being modified is the Deleted Objects container (section 6.1.1.4.2).
- The DC functional level is DS_BEHAVIOR_WIN2008R2 or higher, the modification only affects the nTSecurityDescriptor attribute, and the requester has the Reanimate-Tombstones control access right on the NC root of the object's NC.

Any other modifications of these objects fail with *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.

- In AD DS, modifications to objects of LSA-specific object classes (section 3.1.1.5.2.3) fail with *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- It is disallowed to modify constructed attributes, with the exception of the entryTTL attribute. Such modifications fail with *undefinedAttributeType* / *ERROR_DS_ATT_NOT_DEF_IN_SCHEMA* if the DC functional level is DS_BEHAVIOR_WIN2000, and *constraintViolation* / *ERROR_DS_CONSTRUCTED_ATT_MOD* if the DC functional level is DS_BEHAVIOR_WIN2003 or greater.
- Updates to the name attribute, as well as updates to the object's naming attribute (the attribute named by the rdnTypeattribute), are disallowed and modification will return *notAllowedOnRDN* / *ERROR_DS_CANT_MOD_SYSTEM_ONLY*. Modify DN performs these updates.
- A modify of an object whose objectClass is defunct fails with *objectClassViolation* / *ERROR_DS_OBJECT_CLASS_REQUIRED*.
- If the forest functional level is less than DS_BEHAVIOR_WIN2003, a modify is allowed to remove all values of a defunct attribute. Any other modification that references a defunct attribute fails with *undefinedAttributeType* / *ERROR_DS_ATT_NOT_DEF_IN_SCHEMA*.
- If the forest functional level is greater than or equal to DS_BEHAVIOR_WIN2003, a modify that references a defunct attribute fails with *noSuchAttribute* / *ERROR_INVALID_PARAMETER*.
- If the **fschemaUpgradeInProgress** field is false on the LDAPConnection instance in dc.ldapConnections ([MS-DRSR] section 5.115) corresponding to the LDAP connection on which the operation is being performed, objectCategory modifications on classSchema objects that have FLAG_SCHEMA_BASE_OBJECT present in systemFlags fail with *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- If the domain functional level is less than DS_BEHAVIOR_WIN2003, then modifications of msDS-AdditionalDnsHostName fail with *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED*.
- If the DC functional level is DS_BEHAVIOR_WIN2003 or greater and the msDS-UpdateScript attribute is being modified:
  - IsEffectiveRoleOwner(RoleObject(default NC, RidAllocationMaster)) = true. Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.

- The connection is encrypted with at least 128-bit cipher. If the connection is not encrypted with at least 128-bit cipher, then *unwillingToPerform* / *ERROR_DS_CONFIDENTIALITY_REQUIRED* is returned.

The msDS-UpdateScript attribute is for server-to-server replication implementation only; the client does not interpret it. This attribute MAY have meaning to Windows Server operating system implementations, but the meaning is not significant to Windows clients.

- If the dSHeuristics attribute is being modified, the new value must satisfy the following constraints:
  - If the length of the value is 10 or more characters, then the tenth character must be "1";
  - If the length of the value is 20 or more characters, then the twentieth character must be "2";
  - If the length of the value is 30 or more characters, then the thirtieth character must be "3";
  - The same for "4" through "9".

  When this constraint is violated, the error returned depends on the DC functional level. If the DC functional level is DS_BEHAVIOR_WIN2000, no error is returned. If the DC functional level is DS_BEHAVIOR_WIN2003 or greater, then *constraintViolation* / ERROR_DS_CONSTRAINT_VIOLATION is returned.
- If the DC functional level is DS_BEHAVIOR_WIN2003 or greater and the nTMixedDomain attribute is modified, then the object being modified is the domain NC root. Modification of nTMixedDomain on any other object fails with *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- If the servicePrincipalName attribute is modified, then the values must be syntactically valid SPN values (note that additional constraints may apply if the requester did not have WRITE_PROPERTY access to the attribute; see the preceding Validated Writes section 3.1.1.5.3.1.1). Otherwise, *constraintViolation* / *ERROR_DS_NAME_REFERENCE_INVALID* is returned. See section 5.1.1.4, Mutual Authentication, for SPN syntax.
- If the servicePrincipalName or userPrincipalName attribute is modified, the values must meet the constraints specified in section 3.1.1.5.1.3.
- If the fSMORoleOwner attribute is modified, then the only allowed attribute value is the DN of the DSA object of the current DC; for all other values, *unwillingToPerform* / *ERROR_DS_INVALID_ROLE_OWNER* is returned. In other words, the FSMO role can only be "taken" or transferred to the current DC. It cannot be given away.
- System-only attribute modifications (including the case of adding an auxiliary class with a must-have system-only attribute) are disallowed, as well as modifications of all back link attributes; with the following exceptions:
  - If the **fschemaUpgradeInProgress** field is true on the LDAPConnection instance in dc.ldapConnections ([MS-DRSR] section 5.115) corresponding to the LDAP connection on which the operation is being performed.
  - If the DC functional level is DS_BEHAVIOR_WIN2003 or greater, then modifications of the objectClass attribute are permitted, subject to additional constraints (section 3.1.1.5.3.5).
  - If the DC functional level is DS_BEHAVIOR_WIN2003 or greater, then modifications of msDS-Behavior-Version are permitted, subject to additional constraints (section 3.1.1.5.3.4).

- Modifications of msDS-AdditionalDnsHostName are permitted.
- Modifications of systemFlags are permitted only in the following case: the modify is on an attributeSchema object in the schema container, and the change is to set (but not reset) the FLAG_ATTR_IS_RDN bit.
- Modifications of wellKnownObjects are permitted, subject to additional constraints. See section 3.1.1.5.3.6, wellKnownObjects Updates, for more details.
- Modifications of isDeleted and distinguishedName are permitted only when the modify operation is Undelete (section 3.1.1.5.3.7).
- Modifications of mAPIID are permitted, subject to the constraints described in section 3.1.1.2.3.

Otherwise *constraintViolation* / *ERROR_DS_CANT_MOD_SYSTEM_ONLY* is returned.

- The following constraints are enforced if the DC functional level is DS_BEHAVIOR_WIN2003 or greater and the requester is not passing the LDAP_SERVER_PERMISSIVE_MODIFY_OID control:
  - Inserting duplicate values into an attribute fails with *attributeOrValueExists* / *ERROR_DS_ATT_VAL_ALREADY_EXISTS*.
  - A modification that removes values that are not present from an attribute fails with *noSuchAttribute* / *ERROR_DS_CANT_REM_MISSING_ATT_VAL*.
  - Removing an attribute that is not currently present on the object by virtue of the attribute not having any value set on it fails with *noSuchAttribute* / *ERROR_DS_ATT_IS_NOT_ON_OBJ*.
- If the DC functional level is DS_BEHAVIOR_WIN2008 or greater, the following constraints are enforced on objects of class msDS-PasswordSettings:
  - The msDS-PasswordHistoryLength attribute is less than or equal to 1024.
  - The msDS-MinimumPasswordAge attribute is less than or equal to 0.
  - The msDS-MaximumPasswordAge attribute is less than or equal to 0.
  - The msDS-MaximumPasswordAge attribute is less than the value of the msDS-MinimumPasswordAge attribute on the same object after the modify would have completed.
  - The msDS-MinimumPasswordLength attribute is less than or equal to 256.
  - The msDS-LockoutDuration attribute is less than or equal to 0.
  - The msDS-LockoutObservationWindow attribute is less than or equal to 0.
  - The msDS-LockoutDuration attribute is less than or equal to the value of the msDS-LockoutObservationWindow attribute on the same object after the modify would have completed.

  Otherwise, *unwillingToPerform* / *ERROR_DS_SECURITY_ILLEGAL_MODIFY* is returned.
- In AD LDS, if the LDAP policy ADAMDisablePasswordPolicies does not equal 1, and a password value (either unicodePwd or userPassword) is specified in a modify, the password must satisfy the current password policy in effect on the AD LDS server as reported by SamrValidatePassword ([MS-SAMR] section 3.1.5.13.7). If the provided password value does not satisfy the password policy, the modify returns *constraintViolation* / *ERROR_PASSWORD_RESTRICTION*.

- In AD LDS, if the fAllowPasswordOperationsOverNonSecureConnection heuristic of the dSHeuristics attribute (see section 6.1.1.2.4.1.2) is not true, and a password value (either unicodePwd or userPassword) is specified in a modify, the LDAP connection must be encrypted with cipher strength of at least 128 bits. If the connection does not pass the test, the modify returns *operationsError* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- In AD LDS, if the userPrincipalName value is modified, then the new value must be unique within all NCs on this DC. If another object exists with the same userPrincipalName value, the modify returns *constraintViolation* / *ERROR_DS_NAME_NOT_UNIQUE*.
- In AD LDS, if the pwdLastSet attribute is modified, then the operation MUST replace the existing value with a new value of 0 or -1. Otherwise, *constraintViolation* / *ERROR_INVALID_PARAMETER* is returned.
- In AD LDS, if the lockoutTime attribute is modified, then the operation MUST replace the existing value with a new value of 0. Otherwise, *constraintViolation* / *ERROR_INVALID_PARAMETER* is returned.
- In AD LDS, if the msDS-UserAccountDisabled attribute is being set to false, then the operation succeeds if one of the following is true:
  - The LDAP policy ADAMDisablePasswordPolicies equals 1.
  - The ms-DS-UserPasswordNotRequired attribute equals true.
  - The current password value on the object satisfies the current password policy, as reported by SamrValidatePassword ([MS-SAMR] section 3.1.5.13.7).

  If this check fails, the modify returns *constraintViolation* / *ERROR_PASSWORD_RESTRICTION*.
- After the modify operation, the object must remain compliant with the schema as described in section 3.1.1.5.1.1.
- If the object being modified is a SAM-specific object (section 3.1.1.5.2.3), then additional constraints apply (specified in [MS-SAMR] section 3.1.1.6).
- If the modify operation affects the nTSecurityDescriptor attribute, then additional constraints apply (see section 6.1.3, "Security Descriptor Requirements", for more details).
- If the modify operation would require delayed link processing (section 3.1.1.1.16), and such processing is already underway for the object being modified due to a previous update, then the modify returns *busy* / *ERROR_DS_DATABASE_ERROR*.
- If the modify operation adds or replaces values of the description attribute on a SAM-specific object (section 3.1.1.5.2.3), and results in more than one value in the attribute, then the modification fails with *attributeOrValueExists* / *ERROR_DS_SINGLE_VALUE_CONSTRAINT*.
- In AD DS, the following attributes are disallowed in a Modify for an object of class user: badPasswordTime, badPwdCount, dBCSPwd, isCriticalSystemObject, lastLogoff, lastLogon, lastLogonTimestamp, lmPwdHistory, logonCount, memberOf, msDS-User-Account-Control-Computed, ntPwdHistory, objectSid, rid, sAMAccountType, and supplementalCredentials. If one of these attributes is specified in a Modify, the Modify returns *unwillingToPerform* / *ERROR_DS_ATTRIBUTE_OWNED_BY_SAM*.
- In AD DS, the following attributes are disallowed in a Modify for an object of class group: isCriticalSystemObject, memberOf, objectSid, rid, sAMAccountType, and userPassword. If one of

these attributes is specified in a Modify, the Modify returns *unwillingToPerform* / *ERROR_DS_ATTRIBUTE_OWNED_BY_SAM*.

- In AD DS, the following attributes are disallowed in a Modify for an object whose class is not a SAM-specific object class (see 3.1.1.5.2.3): isCriticalSystemObject, lmPwdHistory, ntPwdHistory, objectSid, samAccountName, sAMAccountType, supplementalCredentials, and unicodePwd. If one of these attributes is specified in a Modify, the Modify returns *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.

3.1.1.5.3.3 Processing Specifics

The following processing rules apply to the modify operation:

- If a value of the entryTTL attribute is specified in the modify request, it is processed as follows:
  - If the value of the entryTTL attribute is less than the DynamicObjectMinTTL LDAP setting, then the entryTTL attribute is set to the value of the DynamicObjectMinTTL setting.
  - The current system time, plus the entryTTL attribute interpreted as seconds, is written into the msDS-Entry-Time-To-Die attribute.
- If the modify assigns a value to an FPO-enabled attribute (section 3.1.1.5.2.3) of the existing object, and the DN value in the modify request has <SID=stringizedSid> format (section 3.1.1.3.1.2.4), then the DC creates a corresponding foreignSecurityPrincipal object in the Foreign Security Principals Container (section 6.1.1.4.10) and assigns a reference to the new foreignSecurityPrincipal object as the FPO-enabled attribute value. [MS-SAMR] section 3.1.1.8.9 specifies the creation of the foreignSecurityPrincipal object.
- If the msDS-UpdateScript attribute is changed in an originating update of the Partitions container, then the msDS-ExecuteScriptPassword value is removed from the Partitions container. The msDS-UpdateScript and msDS-ExecuteScriptPassword attributes are for server-to-server replication implementation only; the client does not interpret them. These attributes MAY have meaning to Windows Server operating system implementations, but the meaning is not significant to Windows clients.
- If the objectClass value is updated, then additional operations are performed (see ObjectClass Updates (section 3.1.1.5.3.5) for more details).
- In AD DS, if the wellKnownObjects value is updated, then additional operations are performed (see wellKnownObjects Updates (section 3.1.1.5.3.6) for more details).
- In AD LDS, if a password value (unicodePwd or userPassword) is modified on a bind proxy, then the password operation is "forwarded" to Windows as follows:
  - The objectSid on the bind proxy object is resolved to a Windows user object.
  - A DC hosting the Windows user's domain is discovered.
  - The currently bound user is impersonated.
  - For a change password operation, the NetUserChangePassword API is invoked with the new and old password values.
  - For a reset password operation, then NetUserSetInfo(level=1003) API is invoked with the new password value.
  - The currently bound user is unimpersonated.

If any of the operations above fail, then the modify returns *unwillingToPerform*. This processing rule is not supported by Active Directory Application Mode (ADAM) RTW DCs.

- In AD LDS, if the pwdLastSet attribute is set to -1 (that is, an unexpire-password operation is performed), then the current time is written as the value of the pwdLastSet attribute.
- For originating updates, additional operations may be performed if the object being modified is a SAM-specific object (section 3.1.1.5.2.3); [MS-SAMR] section 3.1.1.8 specifies these additional operations.
- Additional operations may be performed if the object being modified is a schema object (section 3.1.1.5.2.3); the additional operations are specified in section 3.1.1.2.5.
- If link attribute values that refer to deleted-objects are not visible to the update operation (section 3.1.1.3.4.1.25), and the update operation is a complete removal of a link attribute, all existing values of the attribute are removed, including values that refer to deleted-objects. Note that if the update operation is an explicit list of attributes to be removed rather than a directive to completely remove the attribute, then no values that refer to deleted-objects are removed.
- If link attribute values that refer to deleted-objects are not visible to the update operation (section 3.1.1.3.4.1.25), and the update operation is a complete replacement of a link attribute, all existing values of the attribute including values that refer to deleted-objects are removed before any new values specified by the replacement are added.
- If link attribute values that refer to deleted-objects are not visible to the update operation (section 3.1.1.3.4.1.25), and the update operation is the addition of a value to a single-valued attribute, and all existing values of the attribute refer to deleted-objects, then all existing values of the attribute (including values that refer to deleted-objects) are removed before the new value is added.
- In AD LDS, if an originating update is made to the unicodePwd or userPassword attribute on a bind proxy (section 3.1.1.8.2):
  - Let V be the value of the objectSid attribute from the bind proxy.
  - If the modify request specified a password reset (section 3.1.1.3.1.5), pass the password update operation to the host operating system as a request to update the password of a principal whose SID is V with the new password supplied in the modify request.
  - If the modify request specified a password change (section 3.1.1.3.1.5), pass the password update request operation to the host operating system as a request to update the password of a principal whose SID is V and whose current password is the old password specified in the modify request. That principal's password is to be changed to the new password specified in the modify request.

3.1.1.5.3.4 BehaviorVersion Updates

If the DC functional level is DS_BEHAVIOR_WIN2003 or greater and less than DS_BEHAVIOR_WIN2008R2, then originating updates of the msDS-Behavior-Version attribute are permitted, subject to the following additional constraints:

- The object being modified is the NC root of the domain NC (domain functional level) or the CN=Partitions child of the config NC (forest functional level); otherwise, *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned.
- The new value is greater than the current value; otherwise, *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned.

- The operation is performed on the FSMO (PDC for domain functional level updates, Schema Master FSMO for forest functional level updates); otherwise *referral* / *ERROR_DS_REFERRAL* is returned.
- If the domain functional level is being raised, then the domain MUST NOT contain a DC whose functional level is lower than the new value. This is determined by searching the config NC for objects with objectCategory nTDSDSA whose msDS-Behavior-Version attribute value is below the new value and whose hasMasterNCs attribute contains the DN of the domain NC root. If the search returns one or more results, then *unwillingToPerform* / *ERROR_DS_LOW_DSA_VERSION* is returned.
- If the forest functional level is being raised, then the forest MUST NOT contain a DC whose functional level is lower than the new value. This is determined by searching the config NC for objects with objectCategory nTDSDSA whose msDS-Behavior-Version attribute value is below the new value. If the search returns one or more results, then *unwillingToPerform* / *ERROR_DS_LOW_DSA_VERSION* is returned.
- If the domain functional level is being raised from a value below DS_BEHAVIOR_WIN2003 to a value of DS_BEHAVIOR_WIN2003 or greater, then the domain is not a mixed-mode domain. If the domain is a mixed-mode domain, then *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned.
- If the forest functional level is raised from a value below DS_BEHAVIOR_WIN2003 to a value of DS_BEHAVIOR_WIN2003 or greater, then the forest does not contain mixed-mode domains. If the forest does contain mixed-mode domains, then *unwillingToPerform* / *ERROR_DS_NO_BEHAVIOR_VERSION_IN_MIXED_DOMAIN* is returned.

If the DC functional level is DS_BEHAVIOR_WIN2008R2 or greater, then originating updates of the msDS-Behavior-Version attribute are permitted, subject to the following additional constraints:

- The object being modified is the nTDSDSA object of an RODC (DC functional level of an RODC), or NC root of the domain NC (domain functional level) or the CN=Partitions child of the config NC (forest functional level); otherwise, *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned.
- If the DC functional level of an RODC is being modified, the operation is performed on a writable DC that is a member of the same domain the RODC is a member of; otherwise, *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned.
- If the DC functional level of an RODC is being modified, the new value is greater than or equal to the domain functional level of the domain the RODC is a member of; otherwise, *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned.
- If the domain functional level is being modified, the operation is performed on the PDC FSMO; otherwise *referral* / *ERROR_DS_REFERRAL* is returned.
- If the domain functional level is being modified, the new value is greater than the current value or is greater than the forest functional level; otherwise, *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned.
- If the domain functional level is being modified, then the domain MUST NOT contain a DC whose functional level is lower than the new value. This is determined by searching the config NC for objects with objectCategory nTDSDSA or nTDSDSARO, whose msDS-Behavior-Version attribute value is below the new value and whose hasMasterNCs attribute contains the DN of the domain NC root. If the search returns one or more results, then *unwillingToPerform* / *ERROR_DS_LOW_DSA_VERSION* is returned.

- If the domain functional level is being raised from a value below DS_BEHAVIOR_WIN2003 to a value of DS_BEHAVIOR_WIN2003 or greater, then the domain is not a mixed-mode domain. If the domain is a mixed-mode domain, then *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned.
- If the forest functional level is being modified, the operation is performed on the Schema Master FSMO; otherwise *referral* / *ERROR_DS_REFERRAL* is returned.
- If the forest functional level is being modified, then the forest MUST NOT contain a DC whose functional level is lower than the new value. This is determined by searching the config NC for objects with objectCategory nTDSDSA or nTDSDSARO and whose msDS-Behavior-Version attribute value is below the new value. If the search returns one or more results, then *unwillingToPerform* / *ERROR_DS_LOW_DSA_VERSION* is returned.
- If the forest functional level is raised from a value below DS_BEHAVIOR_WIN2003 to a value of DS_BEHAVIOR_WIN2003 or greater, then the forest does not contain mixed-mode domains. If the forest does contain mixed-mode domains, then *unwillingToPerform* / *ERROR_DS_NO_BEHAVIOR_VERSION_IN_MIXED_DOMAIN* is returned.
- If the new value is less than or equal to the existing value, the new value is greater than or equal to DS_BEHAVIOR_WIN2008; otherwise, *unwillingToPerform* / *ERROR_DS_HIGH_DSA_VERSION* is returned.

  Note In Windows versions prior to Windows Server 2012 operating system, unwillingToPerform / ERROR_DS_ILLEGAL_MOD_OPERATION is returned.

3.1.1.5.3.5 ObjectClass Updates

If the DC functional level is DS_BEHAVIOR_WIN2003 or greater, then originating updates of the objectClass attribute are permitted, subject to the following additional constraints:

- If the forest functional level is less than DS_BEHAVIOR_WIN2003, objectClass updates can be performed only on objects in application NCs; otherwise *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED* is returned.
- The specified objectClass value(s) contains a single most specific structural object class; otherwise *objectClassViolation* / *ERROR_DS_OBJ_CLASS_NOT_SUBCLASS* is returned. If the set of object classes specified by an update contains "holes" (that is, classes are missing on the inheritance chain from the most specific structural object class to the distinguished class top), the server fills the "holes" during the update.
- The structural object class is not modified, with two exceptions:
  - It is permitted to convert a user object to an inetOrgPerson by the addition of inetOrgPerson to the objectClass attribute.
  - It is permitted to convert an inetOrgPerson object to a user by the removal of inetOrgPerson from the objectClass attribute.

  Otherwise, the error returned depends on the DC functional level. If the DC functional level is DS_BEHAVIOR_WIN2000, *constraintViolation* / *ERROR_DS_CONSTRAINT_VIOLATION* is returned. If the DC functional level is DS_BEHAVIOR_WIN2003, *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned. If the DC functional level is DS_BEHAVIOR_WIN2008 or greater, *objectClassViolation* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned.

Processing specifics:

- The set of values is updated to include the full inheritance chains of the structural object class as well as all auxiliary classes present in the value.
- The set of values is sorted according to the objectClass requirements (see section 3.1.1.2.4.3 for more information).
- A new value of nTSecurityDescriptor is computed and written based on the new objectClass values, according to the security descriptor requirements (see section 6.1.3).

3.1.1.5.3.6 wellKnownObjects Updates

In AD DS, when a wellKnownObjects value is modified by an originating update, the following additional constraints apply. These constraints are not enforced for replicated updates.

- The update is performed on the PDC FSMO; otherwise *referral* / *ERROR_DS_REFERRAL* is returned.
- The update is on the domain NC root object; otherwise, *unwillingToPerform* / *ERROR_DS_UNWILLING_TO_PERFORM* is returned.
- The domain functional level is at least DS_BEHAVIOR_WIN2003; otherwise *unwillingToPerform* / *ERROR_DS_NOT_SUPPORTED* is returned.
- Only the Users and Computers container wellKnownObjects references may be updated. This corresponds to the GUID_USERS_CONTAINER_W and GUID_COMPUTERS_CONTAINER_W well-known object (WKO) GUIDs, respectively; otherwise, *unwillingToPerform* / *ERROR_DS_UNWILLING_TO_PERFORM* is returned.
- Only add-value and remove-value LDAP verbs are supported; otherwise, *unwillingToPerform* / *ERROR_DS_UNWILLING_TO_PERFORM* is returned.
- If the DC functional level is DS_BEHAVIOR_WIN2008 or greater, then the object named by the new value must satisfy the possSuperiors schema constraint for the objectClass corresponding to the WKO reference being updated. For example, if the wellKnownObjects reference corresponding to the GUID_USERS_CONTAINER_W WKO GUID is updated, then it must be possible to create user objects as children of the object named by the new value. If this constraint is not satisfied, the server returns *unwillingToPerform* / *ERROR_DS_ILLEGAL_SUPERIOR*.
- The added value does not reside in the container identified by the DN of "CN=System,<domain NC DN>"; otherwise, *unwillingToPerform* / *ERROR_DS_DISALLOWED_IN_SYSTEM_CONTAINER* is returned.
- The object named by the new value MUST NOT have the following bits set in its systemFlags value: FLAG_DISALLOW_DELETE, FLAG_DOMAIN_DISALLOW_RENAME or FLAG_DOMAIN_DISALLOW_MOVE; otherwise *unwillingToPerform* / *ERROR_DS_WKO_CONTAINER_CANNOT_BE_SPECIAL* must be returned.
- The removed value matches the corresponding existing value of the WKO reference. If not, then *unwillingToPerform* / *ERROR_DS_UNWILLING_TO_PERFORM* is returned.

Processing specifics:

- The following bits MUST be set in the systemFlags of the new container: FLAG_DISALLOW_DELETE, FLAG_DOMAIN_DISALLOW_RENAME and FLAG_DOMAIN_DISALLOW_MOVE.

- The following bits MUST be reset in the systemFlags of the old container: FLAG_DISALLOW_DELETE, FLAG_DOMAIN_DISALLOW_RENAME and FLAG_DOMAIN_DISALLOW_MOVE.
- isCriticalSystemObject MUST be set to true on the new container.
- isCriticalSystemObject MUST be set to false on the old container.

3.1.1.5.3.7 Undelete Operation

The undelete operation is used to revert the effects of a delete operation; that is, to turn a tombstone or deleted-object into a regular object (see section 3.1.1.5.5 for more details). The undelete operation is represented by a regular LDAP modify operation, which contains special instructions that are used to distinguish it from a modify operation. These instructions (attribute modifications) are disallowed for regular modify operations.

The undelete operation is identified by the presence of the following attribute LDAPMods (both MUST be present):

- REMOVE isDeleted attribute
- REPLACE distinguishedName attribute with a new value

The undelete operation combines characteristics of both Modify and Modify DN operations. It modifies the object's attributes and moves it in the same transaction.

3.1.1.5.3.7.1 Undelete Security Considerations

In order to be able to perform the undelete operation as an originating update, the requester must have the following permissions. No permissions are required for replicated updates.

- The Reanimate-Tombstones control access right on the NC root of the NC where the operation is being performed.
- All the permissions required to rename an object (section 3.1.1.5.4).
- CREATE_CHILD on the new parent container for the objectClass of the object being undeleted.

Note Unlike with the Modify DN operation, the Delete/DeleteChild permission is not required.

3.1.1.5.3.7.2 Undelete Constraints

For originating updates, the following constraints are enforced for the Undelete operation; otherwise *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* is returned (unless specified otherwise). These constraints do not apply to replicated updates.

- All the modify constraints as they apply to the attributes being modified within the undelete processing (described in previous sections).
- All the Modify DN constraints as they apply to the "move" portion of the undelete operation, with the exception of the "disallowed to move in or out of the System container" constraint.
- If the Recycle Bin optional feature is not enabled, the target object is a tombstone; that is, the isDeleted attribute must be true. If the DC functional level is DS_BEHAVIOR_WIN2008R2 or higher, the error returned is *noSuchAttribute* / *ERROR_DS_ATT_IS_NOT_ON_OBJ*.

- If the Recycle Bin optional feature is enabled, the target object is a deleted-object; that is, the isDeleted attribute is true and the isRecycled attribute is not present on the object. If the DC functional level is DS_BEHAVIOR_WIN2008R2 or higher, the error returned is *noSuchAttribute* / *ERROR_DS_ATT_IS_NOT_ON_OBJ*.
- The target object is not the Deleted Objects container in its NC.
- The target object is not the user object of the currently connected user (that is, the user may not undelete his own object).
- After the modify attribute updates are applied, the object is checked for full schema compliance with regard to both mayContain and mustContain constraints.
- The new object DN is specified in string format (as opposed to <GUID=*stringized-guid*> or <SID=*stringized-sid*> format).
- The new parent container is in the same NC as the target tombstone object (that is, cross-NC undelete is not allowed).
- If the undelete operation would require delayed link processing (section 3.1.1.1.16), and such processing is already underway for the object being undeleted due to a previous update, then the undelete returns *busy* / *ERROR_DS_DATABASE_ERROR*.
- If the target object contains userPrincipalName or servicePrincipalName attribute values, those values must meet the uniqueness constraints specified in section 3.1.1.5.1.3.

3.1.1.5.3.7.3 Undelete Processing Specifics

The undelete operation comprises two suboperations: modifying the object and moving it to a new location. The destination of the move operation is obtained from the DN specified in the request.

- All the Modify operation processing specifics apply.
- All the Modify DN operation processing specifics apply.
- If the user did not specify the value for objectCategory attribute, and the target object did not have this value retained at the time of deletion, then the default objectCategory attribute is written, as obtained from the objectClass's defaultObjectCategory value (section 3.1.1.2.4.8).
- On originating updates, additional processing may apply if the object being reanimated is a SAM-related object (see [MS-SAMR] section 3.1.1.8).

3.1.1.5.4 Modify DN

References

- LDAP control LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID: see section 3.1.1.3.
- LDAP Modify DN operation: see [RFC2251] section 4.9.
- Concrete structure DRS_MSG_MOVEREQ: see [MS-DRSR] section 4.1.15.1.1.
- Concrete structure DRS_MSG_MOVEREQ_V2: see [MS-DRSR] section 4.1.15.1.3.
- Concrete structure DRS_SecBufferDesc: see [MS-DRSR] section 5.44.
- Concrete structure DRS_MSG_MOVEREPLY: see [MS-DRSR] section 4.1.15.1.4.

- Concrete structure DRS_MSG_MOVEREPLY_V2: see [MS-DRSR] section 4.1.15.1.6.
- Concrete method IDL_DRSInterDomainMove: see [MS-DRSR] section 4.1.15.
- Concrete method IDL_DRSBind: see [MS-DRSR] section 4.1.3.
- Function RoleObject: section 3.1.1.5.1.
- Function GetWellknownObject: section 3.1.1.1.6.
- Kerberos delegation: [MS-KILE].
- Glossary: global group, config NC, default NC, dsname, NC replica, prefix table, primary group, RID, schema NC, SID, structural class.
- Access control rights RIGHT_DELETE, RIGHT_DS_DELETE_CHILD.
- LDAP attributes: distinguishedName, groupType, instanceType, isCriticalSystemObject, isDeleted, lDAPDisplayName, member, msDS-NonMembers, name, nCName, objectSid, proxiedObjectName, systemFlags, systemOnly, userAccountControl, wellKnownObjects.
- State model attribues: parent, rdnType.
- LDAP classes: classSchema, crossRef, infrastructureUpdate.

Constants

- Access mask bits: RIGHT_DELETE, RIGHT_DS_DELETE_CHILD: see section 5.1.
- GROUP_TYPE_BUILTIN_LOCAL_GROUP, GROUP_TYPE_ACCOUNT_GROUP, GROUP_TYPE_RESOURCE_GROUP, GROUP_TYPE_SECURITY_ENABLED: see section 2.2.12.
- ADS_UF_WORKSTATION_TRUST_ACCOUNT, ADS_UF_INTERDOMAIN_TRUST_ACCOUNT: see [MS-DRSR] section 5.202, userAccountControl Bits.
- GUID_INFRASTRUCTURE_CONTAINER_W, GUID_SYSTEMS_CONTAINER_W: see section 6.1.1.4.

The Modify DN originating update operation modifies the DN of the object.

The requester supplies the following data:

- *OldDN*: DN of the object that is being modified by the Modify DN operation.
- *NewRDN*: RDN that will form the leftmost component of the new name of the object.
- *NewParentDN*: DN of the object that becomes the immediate superior of the object.
- *DeleteOldRDN*: Boolean value that says whether the old RDN value should be retained. True means that the old RDN value should NOT be retained.

Let *NewDN* be the DN of the renamed object. The value *NewDN* is *NewParentDN* preceded by *NewRDN*.

Definitions

Let *O* be the object such that O!distinguishedName = OldDN.

Let *P* be O!parent.

If *NewParentDN* = NULL then *NP* is O!parent.

Otherwise, let *NP* be an object such that NP!distinguishedName = NewParentDN.

The originating update is a rename operation if O!name ≠ *NewRDN*.

The originating update is a move operation if *P* ≠ *NP*.

3.1.1.5.4.1 Intra Domain Modify DN

For originating updates, if the requester does not specify LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID LDAP control in the Modify DN request, then the server interprets the update as an intradomain Modify DN operation. Replicated updates are always interpreted as intradomain Modify DN operations. The request must have the LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID control (see section 3.1.1.3.4.1.2) if the requester intends to perform a cross-domain move operation. Cross-domain move is not supported by AD LDS.

3.1.1.5.4.1.1 Security Considerations

For originating updates, the requester must have all the following permissions to perform a Modify DN operation. If the security check does not succeed, the server returns the error *insufficientAccessRights* / *ERROR_DS_INSUFF_ACCESS_RIGHTS*.

The security context of the requester must be granted rights RIGHT_DS_WRITE_PROPERTY permission on *O*!name to perform move or rename operation.

For a move operation, the requester must be granted right RIGHT_DS_CREATE_CHILD on *NP* for the objectClass of the object being added.

For a move operation, the requester must be granted rights RIGHT_DELETE on *O*, or must be granted right RIGHT_DS_DELETE_CHILD on *P*.

In AD DS, if *O* is within the config NC or schema NC and the RM control field of the security descriptor of the object has the SECURITY_PRIVATE_OBJECT bit set, the requester must be the owner of the object to perform this operation.

No access check is performed for replicated updates.

3.1.1.5.4.1.2 Constraints

For originating updates, the following constraints must be satisfied for the Modify DN operation. These constraints are not enforced for replicated updates.

- *DeleteOldRDN* = true. Otherwise, the server returns the error *unwillingToPerform* / *ERROR_INVALID_PARAMETER*.
- *OldDN* ≠ NULL. Otherwise, the server returns the error *noSuchObject* / *ERROR_DS_OBJ_NOT_FOUND*.
- *NewRDN* ≠ NULL. Otherwise, the server returns the error *protocolError* / *ERROR_INVALID_PARAMETER*.
- All naming constraints on *NewRDN* must be satisfied. This is explained in section 3.1.1.3.1.2.
- *O* is present. Otherwise, the server returns the error *noSuchObject* / *ERROR_DS_OBJ_NOT_FOUND*.

- *NP* is present. Otherwise, the server returns the error *other* / *ERROR_DS_NO_PARENT_OBJECT*.
- Both *O* and *NP* must be within the same NC Replica. Otherwise, the server returns the error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- *NP* is not equal to *O* or a descendant of *O*. If it is, then the server returns *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- (*O* is in the System container) if and only if (*NP* is the System container or an object inside the System container). Otherwise, the server returns the error *other* / *ERROR_DS_UNWILLING_TO_PERFORM* if the DC functional level is DS_BEHAVIOR_WIN2000, and the error *other* / *ERROR_DS_DISALLOWED_IN_SYSTEM_CONTAINER* if the DC functional level is DS_BEHAVIOR_WIN2003 or greater.
- *O* is not an LSA-specific object (section 3.1.1.5.2.3). Otherwise, the server returns the error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- *O*!isDeleted ≠ true. Otherwise, the server returns the error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- *O* must not be NC root. Otherwise, the server returns the error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* if the DC functional level is DS_BEHAVIOR_WIN2000, and *unwillingToPerform* / *ERROR_DS_MODIFYDN_DISALLOWED_BY_INSTANCE_TYPE* if the DC functional level is DS_BEHAVIOR_WIN2003 or greater.
- If (*O* is in config NC) and (operation is rename), then (*O*!systemFlags & FLAG_CONFIG_ALLOW_RENAME ≠ 0). Otherwise, the server returns the error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* if the DC functional level is DS_BEHAVIOR_WIN2000, and *unwillingToPerform* / *ERROR_DS_MODIFYDN_DISALLOWED_BY_FLAG* if the DC functional level is DS_BEHAVIOR_WIN2003 or greater.
- If (*O* is in config NC) and (operation is move), then either (*O*!systemFlags & FLAG_CONFIG_ALLOW_MOVE ≠ 0) or ((((*O*!parent)!parent)!parent before and after move is the same) and (*O*!systemFlags & FLAG_CONFIG_ALLOW_LIMITED_MOVE ≠ 0)). Otherwise, the server returns the error *unwillingToPerform* / *ERROR_DS_MODIFYDN_DISALLOWED_BY_FLAG*. The FLAG_CONFIG_ALLOW_LIMITED_MOVE flag is used to move server objects between site containers.
- If (operation is move) and (*O* is in schema NC), then the server returns the error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* if the DC functional level is DS_BEHAVIOR_WIN2000, and *unwillingToPerform* / *ERROR_DS_NO_OBJECT_MOVE_IN_SCHEMA_NC* if the DC functional level is DS_BEHAVIOR_WIN2003 or greater.
- If (*O* is a classSchema object) or (*O* is an attributeSchema object), then (*O*!systemFlags & FLAG_SCHEMA_BASE_OBJECT = 0). Otherwise, if the **fschemaUpgradeInProgress** field is false on the LDAPConnection instance in dc.ldapConnections ([MS-DRSR] section 5.115) corresponding to the LDAP connection on which the operation is being performed then the server returns the error *unwillingToPerform* / *ERROR_DS_ILLEGAL_BASE_SCHEMA_MOD*.
- If (*O* is in domain or schema NCs) and (operation is rename) and (attribute *O*!systemFlags is present), then (*O*!systemFlags & FLAG_DOMAIN_DISALLOW_RENAME = 0). Otherwise, the server returns the error *unwillingToPerform* / *ERROR_DS_MODIFYDN_DISALLOWED_BY_FLAG*.
- If (*O* is in domain NC) and (operation is move) and (attribute *O*!systemFlags is present), then (*O*!systemFlags & FLAG_DOMAIN_DISALLOW_MOVE = 0). Otherwise, the server returns the error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* if the DC functional level is

DS_BEHAVIOR_WIN2000, and *unwillingToPerform* / *ERROR_DS_MODIFYDN_DISALLOWED_BY_FLAG* if the DC functional level is DS_BEHAVIOR_WIN2003 or greater.

- The object class of *O* must satisfy the possSuperiors schema constraint for the objectClass of *NP*. Schema constraints are explained in Restrictions on schema extensions in section 3.1.1.2.
- There exists no object *CC* such that *CC*!parent = *NP*, *CC*!name = *O*!name, and *CC* ≠ *O*. Otherwise, the server returns the error *entryAlreadyExists* / *ERROR_DS_OBJ_STRING_NAME_EXISTS*.

3.1.1.5.4.1.3 Processing Specifics

- If the operation is move, set *O*!parent to the objectGUID of the new parent object *NP*.
- Let *A* be the attribute on *O* equal to *O*!rdnType. Set *O*!*A* to *newRDN*.
- Set *O*!name to *newRDN*.

3.1.1.5.4.2 Cross Domain Move

The Modify DN LDAP request must have LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID control to indicate that the requester intends to perform a cross-domain move operation. Cross-domain move is not supported by AD LDS.

The controlValue field of LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID control has the DNS hostname of the target DC that must be used as a helper to perform cross-domain move. If the DNS hostname is not specified in the controlValue field of the LDAP control, then the server will only perform constraint check as explained in section 3.1.1.3.

3.1.1.5.4.2.1 Security Considerations

The requester must have all the following permissions to perform a cross-domain move operation. If the security check does not succeed, the server returns the error *insufficientAccessRights* / *ERROR_DS_INSUFF_ACCESS_RIGHTS*.

For a move operation, the requester must be granted right RIGHT_DELETE on *O* or must be granted right RIGHT_DS_DELETE_CHILD on *P*.

The requester must have performed a Kerberos LDAP bind with delegation enabled (see [RFC4120] section 2.8). Delegation should be enabled because the server impersonates the requester when it contacts the target DC to perform cross-domain move. If Kerberos delegation is not enabled on the LDAP connection, the server returns the error *inappropriateAuthentication* / *ERROR_DS_INAPPROPRIATE_AUTH*.

3.1.1.5.4.2.2 Constraints

The following constraints must be satisfied for the Modify DN operation.

- *DeleteOldRDN* = true. Otherwise, the server returns error *unwillingToPerform* / *ERROR_INVALID_PARAMETER*.
- *OldDN* ≠ NULL and *NewParentDN* ≠ NULL. Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_XDOM_MOVE_OPERATION*.

- *NewRDN* ≠ NULL. Otherwise, the server returns error *protocolError* / *ERROR_INVALID_PARAMETER*.
- (*O*!systemFlags & FLAG_DISALLOW_DELETE = 0). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* if the DC functional level is DS_BEHAVIOR_WIN2000, and *unwillingToPerform* / *ERROR_DS_CANT_DELETE* if the DC functional level is DS_BEHAVIOR_WIN2003 or greater.
- IsEffectiveRoleOwner(RoleObject(default NC, RidAllocationMaster)) = true. Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_INCORRECT_ROLE_OWNER*. This constraint is enforced to avoid conflicting cross-domain move operations.
- Let C be the classSchema object of the most-specific structural class of O. *C*!systemOnly = false. Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_CANT_MOD_SYSTEM_ONLY*.
- *C*!lDAPDisplayName must not be any of the following. Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_XDOM_MOVE_OPERATION*.
  - addressBookContainer
  - attributeSchema
  - builtinDomain
  - certificationAuthority
  - classSchema
  - configuration
  - cRLDistributionPoint
  - crossRef
  - crossRefContainer
  - dMD
  - domain
  - dSA
  - foreignSecurityPrincipal
  - infrastructureUpdate
  - linkTrackObjectMoveTable
  - linkTrackOMTEntry
  - linkTrackVolEntry
  - linkTrackVolumeTable
  - lostAndFound
  - nTDSConnection
  - nTDSDSA

- nTDSSiteSettings
- rIDManager
- rIDSet
- samDomain
- samDomainBase
- samServer
- site
- siteLink
- siteLinkBridge
- sitesContainer
- subnet
- subnetContainer
- trustedDomain

- (*O*!systemFlags & FLAG_DOMAIN_DISALLOW_MOVE = 0). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- (*O*!isCriticalSystemObject ≠ true). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION*.
- (*O*!userAccountControl & ADS_UF_SERVER_TRUST_ACCOUNT = 0) and (*O*!userAccountControl & ADS_UF_INTERDOMAIN_TRUST_ACCOUNT = 0). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_XDOM_MOVE_OPERATION*.
- Let *K* be the RID of SID *O*!objectSid. (*K* > 1000). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_XDOM_MOVE_OPERATION*.
- (*O*!instanceType & IT_WRITE ≠ 0). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_XDOM_MOVE_OPERATION*.
- (*O*!instanceType & IT_NC_HEAD = 0). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_XDOM_MOVE_OPERATION*.
- (*O*!isDeleted ≠ true). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_CANT_MOVE_DELETED_OBJECT*.
- If (*O* is a group object), then (*O*!groupType & GROUP_TYPE_BUILTIN_LOCAL_GROUP = 0). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_XDOM_MOVE_OPERATION*.
- If (*O* is a group object) and ((attribute *O*!member is present) or (attribute *O*!msDS-NonMembers is present)), then (*O*!groupType & GROUP_TYPE_ACCOUNT_GROUP = 0). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_CANT_MOVE_ACCOUNT_GROUP*.

- If (O is a group object) and ((attribute O!member is present) or (attribute O!msDS-NonMembers is present)), then (O!groupType & GROUP_TYPE_RESOURCE_GROUP = 0). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_CANT_MOVE_RESOURCE_GROUP*.
- If (O is a group object) and ((attribute O!member is present) or (attribute O!msDS-NonMembers is present)), then (O!groupType & GROUP_TYPE_APP_BASIC_GROUP = 0). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_CANT_MOVE_APP_BASIC_GROUP*. This constraint is enforced only if the DC functional level is DS_BEHAVIOR_WIN2003 or greater.
- If (O is a group object) and ((attribute O!member is present) or (attribute O!msDS-NonMembers is present)), then (O!groupType = 0). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_CANT_MOVE_APP_QUERY_GROUP*. This constraint is enforced only if the DC functional level is DS_BEHAVIOR_WIN2003 or greater.
- If ((*O* is a user object) or (*O* is a group object)) and (*O* is a member of any global group), then (*O* is a member of only one global group and that group is its primary group). Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_CANT_WITH_ACCT_GROUP_MEMBERSHPS*.
- Let *N* be the root of NC replica where *OldDN* exists. Let *R* be a crossRef object such that *R*!nCName = *N*. *R* must exist and (*R*!systemFlags & FLAG_CR_NTDS_NC ≠ 0) and (*R*!systemFlags & FLAG_CR_NTDS_DOMAIN ≠ 0). Otherwise, the server returns error *noSuchObject* / *ERROR_DS_CANT_FIND_EXPECTED_NC*.
- Let *NN* be the root of NC replica where *NP* exists. Let *NR* be a crossRef object such that *NR*!nCName = *NN*!distinguishedName. *NR* must exist and (*NR*!systemFlags & FLAG_CR_NTDS_NC ≠ 0) and (*NR*!systemFlags & FLAG_CR_NTDS_DOMAIN ≠ 0). Otherwise, the server returns error *noSuchObject* / *ERROR_DS_CANT_FIND_EXPECTED_NC*.
- *R* ≠ *NR*. Otherwise, the server returns error *invalidDNSyntax* / *ERROR_DS_SRC_AND_DST_NC_IDENTICAL*.
- Let *WKS* be a set of all attribute values for *N*!wellKnownObjects. There is no attribute value *V* in *WKS* such that *V.object_DN* = *O*!distinguishedName. Otherwise, the server returns error *unwillingToPerform* / *ERROR_DS_ILLEGAL_XDOM_MOVE_OPERATION*.
- *O* has no child objects. Otherwise, the server returns error *notAllowedOnNonLeaf* / *ERROR_DS_CHILDREN_EXIST*.

3.1.1.5.4.2.3 Processing Specifics

Once the previously described constraint checking is done, the server performs the move operation on the target DC as specified below. The server then performs the cleanup operation as specified below. Constraint checking and cleanup operation are performed in two separate local transactions.

The caller specifies the DNS hostname of the target DC in the *controlValue* field of LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID LDAP control.

If the *controlValue* field is empty, then the server performs only constraints checking as mentioned previously. It returns *success* if it passes all the constraints.

Invoke move operation on target DC:

Let *S* be the nTDSDSA object of the server.

Let *NN* be the root of NC replica where *NP* exists.

Let *pmsgIn* be a reference to a structure of type DRS_MSG_MOVEREQ.

Set *pmsgIn->V2.pSrcDSA* to dsname of *S*.

pmsgIn->V2.pSrcObject is a reference to a structure of type ENTINF. Define ENTINF for *O* as described later in this section.

Set *pmsgIn->V2.pDstName* to dsname of *NewDN*.

Set *pmsgIn->V2.pExpectedTargetNC* to dsname of *NN*.

pmsgIn->V2.pClientCreds is a reference to DRS_SecBuffer structure. It is set to the GSS Kerberos authentication token (see [RFC1964]) derived from the security context of the caller.

Set *pmsgIn->V2.PrefixTable* to dc.prefixTable, as specified in section 3.1.1.1.9.

Set *pmsgIn->V2.ulFlags* to 0.

Let *H* be the bind handle derived by calling IDL_DRSBind method against target DC.

Let *pdwOutVersion* be a reference to *dwOutversion* of type integer.

Let *pmsgOut* be a reference to DRS_MSG_MOVEREPLY structure.

Call IDL_DRSInterDomainMove(*H*, 2, *pmsgIn*, *pdwOutVersion*, *pmsgOut*). If the method returns an error, then the server returns LDAP error *unavailable*.

If (*dwOutVersion* ≠ 2), then the server returns LDAP error *operationsError*.

If (*pmsgOut->v2.win32Error* ≠ 0), then the server returns LDAP error *unwillingToPerform*.

Create proxy object and perform cleanup

The proxiedObjectName attribute is present on the infrastructureUpdate object that is used to communicate the cross-domain move from the originating NC replica to other replicas of the NC. The proxiedObjectName attribute is also present on an object that has been moved across domain, as specified in [MS-DRSR] section 4.1.15.3.

The proxiedObjectName attribute has syntax Object(DN-Binary); see section 3.1.1.2.2.2.3 for the specification of this syntax, which contains the fields char_count, binary_value, and object_DN. The binary_value part of a proxiedObjectName value is 16 characters. Bytes 0 to 7 contain the character string "00000001" for a cross-domain move. Bytes 8 to 15 contain the hexadecimal representation of a number called the cross-domain move epoch.

The cross-domain move epoch *E* of the proxiedObjectName attribute on an infrastructureUpdate object is determined as follows:

- If *O*!proxiedObjectName is present, then let *B* be the binary_value of *O*!proxiedObjectName. Let *E* be value given by the least significant 32 bits of *B*.
- Otherwise, let *E* be 0.

Create an attribute value *K* of type Object (DN-Binary). Set *K.char_count* to 16. Let *J* be a string of eight characters that is the hexadecimal representation of value *E*. Set *K.binary_value* to the concatenation of the strings "00000001" and *J*. Set *object_DN* part of K to *NewDN*.

Expunge object *O* from *NC replica*.

Let *I* = GetWellknownObject(default NC, GUID_INFRASTRUCTURE_CONTAINER_W).

Create an infrastructureUpdate object *L* such that *L*!parent = *I* and *L*!name is any name unique among the children of *I* and *L*!proxiedObjectName = *K* and *L*!systemFlags = (FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DISALLOW_MOVE_ON_DELETE | FLAG_DOMAIN_DISALLOW_MOVE).

Delete L and turn it into a *tombstone* object.

Defining ENTINF structure for object O

Let *t* be the prefix table dc.prefixTable specified in section 3.1.1.1.9.

Let *AttsSet* be the set of all attributes (represented as ATTRTYP) of object *O*.

Let *Atts* be a sequence of ATTRTYP whose elements are elements of *AttsSet*.

Let *EntInf* be a structure of type ENTINF.

Set *EntInf.pName* to the dsname of *O*.

Set *EntInf.ulFlags* to 0.

Let *AttrBlock* be a structure of type ATTRBLOCK of length *Atts.length*.

Give *AttrBlock.pAttr[i]* a value determined by *Atts[i]* as follows, for all *i* in [0...*Atts.length*) (in any order)

- Let K be the attributeSchema object SchemaObj(*Atts[i]*). SchemaObj is specified in [MS-DRSR] section 5.179.
- Let *syntax* be *K*!attributeSyntax.
- Let *AttrBlock.pAttr[i].AttribTyp* be the value returned by MakeAttid(*t*, *oid*).
- Let *Vals* be the sequence of values *O.Atts[i]*.
- Let *AttrBlock.pAttr[i].AttrVal* be a structure of type ATTRVALBLOCK of length *Vals.length*.
- Set *AttrBlock.pAttr[i].AttrVal.valCount* = *Vals.length*.
- Give *AttrBlock.pAttr[i].AttrVal.pAVal[j]* a value determined by *Vals[j]* as follows, for all *j* in [0..*Vals.length*) (in any order).
  - Set *AttrBlock.pAttr[i].AttrVal.pAVal[j]* = ATTRVALFromValue(*Vals[j]*, *syntax*, *t*)

3.1.1.5.5 Delete Operation

References

LDAP attributes: distinguishedName, isDeleted, isRecycled, entryTTL, msDS-Entry-Time-To-Die, nTSecurityDescriptor, attributeID, attributeSyntax, dNReferenceUpdate, dNSHostName, flatName, governsID, groupType, instanceType, lDAPDisplayName, legacyExchangeDN, mS-DS-CreatorSID, msDS-LastKnownRDN, mSMQOwnerID, nCName, objectClass, objectGUID, objectSid, oMSyntax, proxiedObjectName, name, replPropertyMetaData, sAMAccountName, securityIdentifier, sIDHistory, subClassOf, systemFlags, trustPartner, trustDirection, trustType, trustAttributes, userAccountControl, uSNChanged, uSNCreated, whenCreated, searchFlags, isCriticalSystemObject, objectCategory, sAMAccountType, isDeleted, lastKnownParent.

State model attributes: rdnType

LDAP classes: dynamicObject, crossRef.

Constants

- Win32/status error codes: ERROR_DS_REFERRAL, ERROR_DS_ILLEGAL_MOD_OPERATION, ERROR_DS_CHILDREN_EXIST, ERROR_DS_TREE_DELETE_NOT_FINISHED
- Access mask bits, control access rights: SECURITY_PRIVATE_OBJECT, RIGHT_DELETE, RIGHT_DS_DELETE_CHILD, RIGHT_DS_DELETE_TREE
- Security privileges:
- systemFlags bits: FLAG_DISALLOW_DELETE, FLAG_DISALLOW_MOVE_ON_DELETE
- Schema bits: fPRESERVEONDELETE
- LDAP:

The delete operation results in the transformation of an existing object in the directory tree into some form of deleted-object. There are several modes of transformation, depending on whether the Recycle Bin optional feature is enabled or not. In all modes of transformation, the requester supplies the DN of the object to be transformed.

If the Recycle Bin optional feature is not enabled, the delete operation results in the transformation of an existing-object in the directory tree into a tombstone. If the Recycle Bin optional feature is enabled and the requester has specified an existing-object as the object to be transformed, the deletion operation results in transformation of the existing-object in the directory tree into a deleted-object.

If the Recycle Bin optional feature is enabled and the requester has specified a deleted-object as the object to be transformed, the operation results in transformation of a deleted-object in the directory tree into a recycled-object. Recycled-objects are created only by the transformation of a deleted-object, never directly from a normal object.

Tombstones, deleted-objects, and recycled-objects are special placeholder objects that replicate around, signaling replica partners that the original object was deleted. Tombstones, deleted-objects, and recycled-objects are invisible to LDAP searches by default, so for an LDAP application, it appears that the object was physically removed from the directory after a delete operation has taken place.

Tombstones are distinguished from existing-objects by the presence of the isDeleted attribute with the value true. The value of the isRecycled attribute may be true, or the isRecycled attribute may be absent. Tombstones exist only when the Recycle Bin optional feature is not enabled. After a time period at least as large as a tombstone lifetime, the tombstone is removed from the directory.

Deleted-objects are distinguished from existing-objects by the presence of the isDeleted attribute with the value true and the absence of the isRecycled attribute. Deleted-objects exist only when the Recycle Bin optional feature is enabled. After a time period at least as large as a deleted-object lifetime, the deleted-object is transformed into a recycled-object.

Recycled-objects are distinguished from existing-objects by the presence of the isRecycled attribute with the value true. Recycled-objects exist only when the Recycle Bin optional feature is enabled. After a time period at least as large as a tombstone lifetime, the recycled-object is removed from the directory.

Normally, only leaf objects (objects without descendants in the directory tree) may be deleted. There is also a special tree-delete operation, with which whole trees of objects are removed (see Tree-delete operation in section 3.1.1.5.5.7).

In most cases, upon deletion, a tombstone, deleted-object, or recycled-object is moved into the Deleted Objects container of its NC; for exceptions see section 3.1.1.5.5.6. The RDN of the object is changed to a "delete-mangled RDN"-an RDN that is guaranteed to be unique within the Deleted Objects container. If *O* is the object that is deleted, the delete-mangled RDN is the concatenation of *O*!name, the character with value 0x0A, the string "DEL:", and the dashed string representation ([RFC4122] section 3) of *O*!objectGUID. During this concatenation, if required, the O!name part is truncated to ensure that the length of the delete-mangled RDN does not violate the RDN size constraint in section 3.1.1.5.1.2. The RDN attribute of this object is also set to this delete-mangled RDN value. The illegal character constraint regarding a character with the value 0xA, as specified in section 3.1.1.5.1.2, is not enforced for this delete-mangled RDN. Also, the rangeUpper constraint for the RDN attribute of this object is not enforced. A "delete-mangled DN" is a DN such that the leaf RDN is a delete-mangled RDN.

An object whose class is defunct, or whose class is active but some of whose attributes are defunct, can still be deleted.

Linked attributes store references to other objects in the forest (see referential integrity in section 3.1.1.1.6). They are pairs of attributes for which the system calculates the values of one attribute (the back link) based on the values set on the other attribute (the forward link) throughout the forest. A back-link value on any object instance consists of the DNs of all the objects that have that object's DN set in the corresponding forward link. In addition to storing object references using linked attributes, objects can also store references to other objects in attributes that have an object reference syntax (see referential integrity in section 3.1.1.1.6). Such attributes are not considered to be linked attributes.

The direction of a linked attribute is determined by the directional flow of a forward link and the object from which this link is viewed. If this object has a forward link attribute containing a reference to another object, then its linked attribute is called an outgoing linked attribute. The link, as viewed from the referenced object, is called an incoming link. For example, if Object A has a forward link storing a reference to Object B (this implies that Object B has a backward link storing a reference to Object A), then the linked attribute on Object A is an outgoing linked attribute and accordingly, an incoming linked attribute on Object B.

3.1.1.5.5.1 Resultant Object Requirements

3.1.1.5.5.1.1 Tombstone Requirements

The following requirements apply to tombstones:

- The isDeleted attribute is set to true on tombstones.
- The tombstone does not have descendant objects.
- The tombstone remains in the database and is available for outbound replication for at least the tombstone lifetime time interval (see section 6.1.1) after its transformation into a tombstone, with the following exception:
  - The Deleted Objects container (which is considered a tombstone if the Recycle Bin optional feature is not enabled) always remains available.
- A tombstone does not retain the attribute values of the original object for any attributes except for the following:
  - The attribute that is the RDN, plus the objectGUID and objectSid attributes.
  - Attributes marked as being preserved on deletion (see section 2.2.9).

- Attributes on the following list:
  - attributeID, attributeSyntax, dNReferenceUpdate, dNSHostName, flatName, governsID, groupType, instanceType, lDAPDisplayName, legacyExchangeDN, isDeleted, isRecycled, lastKnownParent, msDS-LastKnownRDN, mS-DS-CreatorSID, mSMQOwnerID, nCName, objectClass, distinguishedName, objectGUID, objectSid, oMSyntax, proxiedObjectName, name, nTSecurityDescriptor, replPropertyMetaData, sAMAccountName, securityIdentifier, sIDHistory, subClassOf, systemFlags, trustPartner, trustDirection, trustType, trustAttributes, userAccountControl, uSNChanged, uSNCreated, whenCreated, msDS-PortLDAP

- A tombstone does not retain the attribute values of the original object for the attributes objectCategory and sAMAccountType or for any linked attributes even if these attributes would otherwise be retained according to the preceding bullet point. In other words, when an object is deleted and transformed into a tombstone, objectCategory values, sAMAccountType values, and any linked attribute values on it are always removed.
- NC replicas do not contain objects with linked attribute values referencing tombstones. In other words, when an object is deleted and transformed into a tombstone, any linked attribute values on other objects referencing it are also removed.
- If any NC replicas contain other objects with nonlinked attribute values referencing a tombstone, then those attribute values on those objects are retained. In other words, when an object is deleted and transformed into a tombstone, any nonlinked attribute values on other objects referencing it are not removed.
- Except as described in section 3.1.1.5.5.6, tombstones exist only in the Deleted Objects container of an NC.
- Except as described in section 3.1.1.5.5.6, tombstones have "delete-mangled RDNs".
- A protected object may not be deleted and transformed into a tombstone (see Protected Objects (section 3.1.1.5.5.3)).

3.1.1.5.5.1.2 Deleted-Object Requirements

The following requirements apply to deleted-objects:

- The isDeleted attribute is set to true on deleted-objects.
- The isRecycled attribute is not present.
- The deleted-object retains all of the attributes of the original object except for the attributes objectCategory and sAMAccountType.
- The deleted-object does not have descendant objects.
- The deleted-object remains in the database and is available for outbound replication for at least the deleted-object lifetime interval (see section 6.1.1) after its deletion, with the following exception:
  - The Deleted Objects container (which is itself a deleted-object if the Recycle Bin optional feature is enabled) always remains available, and is never transformed into a recycled-object.
- If a deleted-object has linked attribute values, then those attribute values are retained. For details, see LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID (section 3.1.1.3.4.1.25).

- If any NC replicas contain other objects with linked attribute values referencing deleted-objects, then those attribute values on those objects are retained. In other words, when an object is deleted and transformed into a deleted-object, any linked attribute values on other objects referencing it are not removed. For details, see LDAP_SERVER_SHOW_DEACTIVATED_LINK_OID (section 3.1.1.3.4.1.25).
- If any NC replicas contain other objects with nonlinked attribute values referencing a deleted-object, then those attribute values on those objects are retained. In other words, when an object is deleted and transformed into a deleted-object, any nonlinked attribute values on other objects referencing it are not removed.
- Except as described in section 3.1.1.5.5.6, deleted-objects exist only in the Deleted Objects container of an NC.
- Except as described in section 3.1.1.5.5.6, deleted-objects have "delete-mangled RDNs".
- A protected object may not be deleted and transformed into a deleted-object (see Protected Objects in section 3.1.1.5.5.3).

3.1.1.5.5.1.3 Recycled-Object Requirements

The following requirements apply to recycled-objects:

- The isDeleted attribute is set to true on recycled-objects.
- The isRecycled attribute is set to true on recycled-objects.
- The recycled-object does not have descendant objects.
- The recycled-object remains in the database and is available for outbound replication for at least the tombstone lifetime time interval (see section 6.1.1) after its transformation into a recycled-object.
- A recycled-object does not retain the attribute values of the deleted object for any attributes except for the following:
  - The attribute that is the RDN, plus the objectGUID and objectSid attributes
  - Attributes marked as being preserved on deletion (see section 2.2.9)
  - Attributes on the following list:
    - nTSecurityDescriptor, attributeID, attributeSyntax, dNReferenceUpdate, dNSHostName, flatName, governsID, groupType, instanceType, lDAPDisplayName, legacyExchangeDN, isDeleted, isRecycled, lastKnownParent, msDS-LastKnownRDN, mS-DS-CreatorSID, mSMQOwnerID, nCName, objectClass, distinguishedName, objectGUID, objectSid, oMSyntax, proxiedObjectName, name, replPropertyMetaData, sAMAccountName, securityIdentifier, sIDHistory, subClassOf, systemFlags, trustPartner, trustDirection, trustType, trustAttributes, userAccountControl, uSNChanged, uSNCreated, whenCreated, msDS-PortLDAP
- A recycled-object does not retain the attribute values of the original object for the attributes objectCategory, sAMAccountType, or for any linked attributes even if these attribute would otherwise be retained according to the preceding bullet point. In other words, when a deleted-object is transformed into a recycled-object, objectCategory values, sAMAccountType values, and any linked attribute values on it are always removed.

- NC replicas do not contain objects with linked attribute values referencing recycled-objects. In other words, when a deleted-object is transformed into a recycled-object, any linked attribute values on other objects referencing it are also removed.
- If any NC replicas contain other objects with nonlinked attribute values referencing a recycled-object, then those attribute values on those objects are retained. In other words, when a deleted-object is transformed into a recycled-object, any non-linked attribute values on other objects referencing it are not removed.
- Except as described in section 3.1.1.5.5.6, recycled-objects exist only in the Deleted Objects container of an NC.
- Except as described in section 3.1.1.5.5.6, recycled-objects have "delete-mangled RDNs".

3.1.1.5.5.2 dynamicObject Requirements

See section 6.1.7.

3.1.1.5.5.3 Protected Objects

The following objects are considered protected and may not be deleted:

- The DC's nTDSDSA object and all of its ancestors.
- The DC's rIDSet object and all of its ancestors. A DC's rIDSet object is the referent of the rIDSetReferences attribute of the DC's Domain Controller object (section 6.1.1.3.1).
- The crossRef objects corresponding to the DC's config, schema, and default domain NCs.

3.1.1.5.5.4 Security Considerations

No permissions are required for replicated updates.

For originating updates, the requester must have the following permissions.

To delete a regular object, at least one of the following permissions must be granted to the requester:

- RIGHT_DELETE on the object being deleted, or
- RIGHT_DS_DELETE_CHILD on the parent of the object being deleted, when the object is not an NC root.

For originating updates of transformations of deleted-objects to recycled-objects, all the same security requirements as those listed for a normal deletion must be met. In addition, the requester must have the permission RIGHT_DS_REANIMATE_TOMBSTONES on the NC root of the NC where the operation is being performed.

3.1.1.5.5.5 Constraints

For originating updates, the following constraints are enforced for the delete operation. These constraints are not enforced for replicated updates.

- The object being deleted resides in a writable NC replica; otherwise, the delete returns *referral* / *ERROR_DS_REFERRAL*.

- If the object being deleted is in the config NC or schema NC, and the RM control ([MS-DTYP] section 2.4.6) of the SD is present and contains the SECURITY_PRIVATE_OBJECT bit (section 6.1.3), additional requirements on the DC performing the operation are enforced (if neither is true, *referral* / *ERROR_DS_REFERRAL* must be returned):
  - The DC must be a member of the root domain in the forest, or
  - The DC must be a member of the same domain where the current object owner belongs.
- If the FLAG_DISALLOW_DELETE bit is set in the systemFlags attribute, *unwillingToPerform* / *ERROR_DS_CANT_DELETE* is returned.
- Deletions of tombstone objects fail with *unwillingToPerform* / *ERROR_DS_ILLEGAL_MOD_OPERATION* if the DC functional level is DS_BEHAVIOR_WIN2008 or lower, and with *unwillingToPerform* / *ERROR_DS_CANT_DELETE* if the DC functional level is DS_BEHAVIOR_WIN2008R2 or higher. However, if the object being deleted is a tombstone of a SAM-specific object (section 3.1.1.5.2.3), *noSuchObject* / *ERROR_DS_OBJ_NOT_FOUND* is returned instead.
- If the object being deleted is a recycled-object, *unwillingToPerform* / *ERROR_DS_CANT_DELETE* is returned.
- If the object being deleted has descendants, the delete operation fails with *notAllowedOnNonleaf* / *ERROR_DS_CHILDREN_EXIST*. This constraint is not effective if the requester is passing the LDAP_SERVER_TREE_DELETE_OID control (see section 3.1.1.5.5.7).
- If the **fschemaUpgradeInProgress** field is false on the LDAPConnection instance in dc.ldapConnections ([MS-DRSR] section 5.115) corresponding to the LDAP connection on which the operation is being performed and the object being deleted is in the schema NC, *unwillingToPerform* / *ERROR_DS_CANT_DELETE* is returned.
- If the object being deleted is a SAM-specific object (section 3.1.1.5.2.3), additional constraints apply (see [MS-SAMR] section 3.1.5.7).
- If the delete operation would require delayed link processing (section 3.1.1.1.16), and such processing is already underway for the object being deleted due to a previous update, then the delete returns *busy* / *ERROR_DS_DATABASE_ERROR*.
- If the object being deleted is the DC's nTDSDSA object or any of its ancestors, *unwillingToPerform* / *ERROR_DS_CANT_DELETE_DSA_OBJ* is returned.
- If the **object** being deleted is a **crossRef object** corresponding to the **DC's config**, **schema**, or **default domain NCs**, the returned error code depends on the following conditions:
  - If the **crossRef object** is a child of the CN=Partitions child of the **config NC** and the **nCName** attribute of the **crossRef object** is set to the value *DN1* and there exists another **crossRef object** with the same parent where the **nCName** attribute of the second **crossRef object** is set to the value *DN2*, and the **object** referred to by *DN1* is an ancestor of the **object** referred to by *DN2*, then *notAllowedOnNonLeaf* / *ERROR_DS_CANT_ON_NON_LEAF* is returned.
  - Else if the **crossRef object** is a child of the CN=Partitions child of the config NC, and the **crossRef object**'s NC is hosted by some domain controller, *unwillingToPerform* / *ERROR_DS_NC_STILL_HAS_DSAS* is returned.
  - Otherwise, *unwillingToPerform* / *ERROR_DS_CANT_DEL_MASTER_CROSSREF* is returned.

- If the object being deleted is protected (see section 3.1.1.5.5.3) and does not fall into the two categories above, *unwillingToPerform* / *ERROR_DS_CANT_DELETE* is returned.

3.1.1.5.5.6 Processing Specifics

3.1.1.5.5.6.1 Transformation into a Tombstone

When the delete operation results in the transformation of an object into a tombstone, the following processing rules apply to the delete operation:

- For originating updates:
  - The RDN for the tombstone is the object's delete-mangled RDN, as specified in Delete Operation in section 3.1.1.5. For replicated updates, the received RDN for the tombstone is set on the object.
  - The lastKnownParent attribute value is set to the DN of the current parent object.
  - Additional operations may be performed if the object being modified is a SAM-specific object (section 3.1.1.5.2.3); see [MS-SAMR] section 3.1.1.8).
- All attribute values are removed from the object, with the following exceptions:
  - nTSecurityDescriptor, attributeID, attributeSyntax, dNReferenceUpdate, dNSHostName, flatName, governsID, groupType, instanceType, lDAPDisplayName, legacyExchangeDN, mS-DS-CreatorSID, mSMQOwnerID, nCName, objectClass, distinguishedName, objectGUID, objectSid, oMSyntax, proxiedObjectName, name, replPropertyMetaData, sAMAccountName, securityIdentifier, sIDHistory, subClassOf, systemFlags, trustPartner, trustDirection, trustType, trustAttributes, userAccountControl, uSNChanged, uSNCreated, whenCreated attribute values are retained.
  - In AD LDS, the msDS-PortLDAP attribute is also retained.
  - The attribute that equals the rdnType of the object (for example, cn for a user object) is retained.
  - Any attribute that has fPRESERVEONDELETE flag set in its searchFlags is retained, except objectCategory and sAMAccountType, which are always removed, regardless of the value of their searchFlags.
- All outgoing linked attribute values are removed, but not as an originating update. These values are simply removed from the directory.
- All incoming linked attribute values are removed, but not as an originating update. These values are simply removed from the directory.
- The isDeleted attribute is set to true.
- The object is moved into the Deleted Objects container in its NC, except in the following scenarios, when it must remain in its current place:
  - The object is an NC root.
  - The object's systemFlags value has FLAG_DISALLOW_MOVE_ON_DELETE bit set.

3.1.1.5.5.6.2 Transformation into a Deleted-Object

When the delete operation results in the transformation of an object into a deleted-object, the following processing rules apply to the delete operation:

- For originating updates:
  - The RDN for the deleted-object is the object's delete-mangled RDN, as specified in Delete Operation in section 3.1.1.5. For replicated updates, the received RDN for the deleted-object is set on the object.
  - The lastKnownParent attribute value is set to the DN of the object's parent at the time of its deletion.
  - The msDS-LastKnownRDN attribute value is set to the RDN of the object before the deletion transformation.
  - Additional operations may be performed if the object being modified is a SAM-specific object (section 3.1.1.5.2.3); see [MS-SAMR] section 3.1.1.8).
- The attributes objectCategory and sAMAccountType are removed.
- The isDeleted attribute is set to true.
- The object is moved into the Deleted Objects container in its NC, except in the following scenarios, when it MUST remain in its current place:
  - The object is an NC root.
  - The object's systemFlags value has FLAG_DISALLOW_MOVE_ON_DELETE bit set.

3.1.1.5.5.6.3 Transformation into a Recycled-Object

When the delete operation results in the transformation of an object into a recycled-object, the following processing rules apply to the delete operation:

- For originating updates:
  - Additional operations may be performed if the object being modified is a SAM-specific object (section 3.1.1.5.2.3); see [MS-SAMR] section 3.1.1.8).
- All attribute values are removed from the object, with the following exceptions:
  - nTSecurityDescriptor, attributeID, attributeSyntax, dNReferenceUpdate, dNSHostName, flatName, governsID, groupType, instanceType, lDAPDisplayName, lastKnownParent, ms-DS-lastKnownRDN, legacyExchangeDN, mS-DS-CreatorSID, mSMQOwnerID, nCName, objectClass, distinguishedName, objectGUID, objectSid, oMSyntax, proxiedObjectName, name, replPropertyMetaData, sAMAccountName, securityIdentifier, sIDHistory, subClassOf, systemFlags, trustPartner, trustDirection, trustType, trustAttributes, userAccountControl, uSNChanged, uSNCreated, whenCreated attribute values are retained.
  - In AD LDS, the msDS-PortLDAP attribute is also retained.
  - The attribute that equals the rdnType of the object (for example, cn for a user object) is retained.

- Any attribute that has the fPRESERVEONDELETE flag set in its searchFlags is retained, except objectCategory and sAMAccountType, which are always removed, regardless of the value of their searchFlags.

- All outgoing linked attribute values are removed, but not as an originating update. These values are simply removed.
- All incoming linked attribute values are removed, but not as an originating update. These values are simply removed.
- The isDeleted attribute is set to true.
- The isRecycled attribute is set to true.
- The object is moved into the Deleted Objects container in its NC, except in the following scenarios, when it MUST remain in its current place:
  - The object is an NC root.
  - The object's systemFlags value has the FLAG_DISALLOW_MOVE_ON_DELETE bit set.

3.1.1.5.5.7 Tree-delete Operation

The tree-delete operation is a special mode of delete operation that simplifies the deletion of trees of objects. The regular delete operation can only delete leaf objects. The tree-delete operation processes a tree of objects one-by-one, deleting objects starting from the leaf objects and continuing up until the root can be deleted. The tree-delete operation is represented by a regular LDAP delete operation with the requester passing the LDAP_SERVER_TREE_DELETE_OID control.

A tree-delete operation is never performed as a replicated update.

3.1.1.5.5.7.1 Tree-delete Security Considerations

The requester must have the RIGHT_DS_DELETE_TREE on the object being deleted. Note that no additional permissions are required on the descendants of the object.

3.1.1.5.5.7.2 Tree-delete Constraints

- All regular delete operation constraints apply on each object being deleted.
- The tree-delete operation cannot be applied to an NC root.
- Objects with the isCriticalSystemObject attribute equal to true and which are not SAM-specific objects (as defined by section 3.1.1.5.2.3) cannot be deleted by the tree-delete operation. This constraint is checked object-by-object, and deletion stops at the first deletion attempt that violates the constraint. If deletion stops, the resultant tree might not be the same as the original tree because some objects might have been deleted prior to the failure.

3.1.1.5.5.7.3 Tree-delete Processing Specifics

- The tree-delete operation proceeds by removing the tree, starting from the leaf objects and making its way to the root of the tree. The order of processing is not important, as long as each node is only deleted after all of its descendants have been deleted and moved into a Deleted Objects container (section 6.1.1.4.2).
- Regular delete processing specifics apply to each object being deleted.

- The tree-delete operation is implemented using multiple transactions.
- It is allowed for the tree-delete operation not to delete the complete subtree. If the server failed to complete the tree-delete operation and the error is recoverable (that is, no user intervention is required), it returns a special error code *adminLimitExceeded* / *ERROR_DS_TREE_DELETE_NOT_FINISHED* to the user. However, it is required that at least one object in the subtree was deleted (that is, some progress was made). The clients continue repeating the tree-delete request until they either receive a *success* (indicating that the tree was successfully removed) or receive an error code other than ERROR_DS_TREE_DELETE_NOT_FINISHED (as specified in section 3.1.1.5.5.5).

3.1.1.6 Background Tasks

In AD DS, the server runs background tasks periodically to:

- Protect security principals that have elevated administrative privilege.
- Maintain referential integrity (see Referential integrity in section 3.1.1.1) on object references.
- Maintain security descriptor requirements (see Security Descriptor Requirements in section 6.1.3).

These tasks are specified in the following sections.

3.1.1.6.1 AdminSDHolder

References

- Special Objects in section 6.1: Windows NT operating system

Glossary Terms: Active Directory, security principal, privileges, PDC, FSMO, SD, **transitive membership**, RID

LDAP attributes: nTSecurityDescriptor, groupType, objectClass, member, objectSid, dSHeuristics

LDAP classes: container, user, group

Constants

- Access mask bits, CARs:
- groupType bits: GROUP_TYPE_SECURITY_ENABLED
- Constant RIDs: DOMAIN_ALIAS_RID_ADMINS, DOMAIN_ALIAS_RID_ACCOUNT_OPS, DOMAIN_ALIAS_RID_SYSTEM_OPS, DOMAIN_ALIAS_RID_PRINT_OPS, DOMAIN_ALIAS_RID_BACKUP_OPS, DOMAIN_ALIAS_RID_REPLICATOR, DOMAIN_GROUP_RID_SCHEMA_ADMINS, DOMAIN_GROUP_RID_ADMINS, DOMAIN_GROUP_RID_CONTROLLERS, DOMAIN_USER_RID_KRBTGT, DOMAIN_USER_RID_ADMIN

If a security principal object with elevated administrative privileges in Active Directory has a weak SD, Active Directory is vulnerable to straightforward attack. Therefore Active Directory protects the SDs of such objects from updates that might give them weak SDs.

Each security principal is represented as an object o in Active Directory. For every o there is an attribute o!nTSecurityDescriptor. The value is the SD that defines ownership, permissions, and audited operations for o.

Active Directory protects the SD on certain objects by periodically overwriting any changes. This mechanism loosely establishes an upper bound on the length of time that a protected object may have a weak SD.

3.1.1.6.1.1 Authoritative Security Descriptor

The security descriptor that is written to protected objects is stored in the nTSecurityDescriptor attribute on the AdminSDHolder object in Active Directory. The AdminSDHolder object is of class container and has a DN of "CN=AdminSDHolder,CN=System,<Domain NC DN>".

3.1.1.6.1.2 Protected Objects

In domain d, the set S of all security principal objects o that are protected is defined as follows:

- (o!objectClass = group AND attribute o!groupType & GROUP_TYPE_SECURITY_ENABLED ≠ 0) OR (o!objectClass = user)
- AND (o!objectSid = d!objectSid + RID)
- AND either
  - o is a member, directly or transitively, of any group in the set:
    - built-in well-known group with RID = DOMAIN_ALIAS_RID_ADMINS
    - built-in well-known group with RID = DOMAIN_ALIAS_RID_ACCOUNT_OPS
    - built-in well-known group with RID = DOMAIN_ALIAS_RID_SYSTEM_OPS
    - built-in well-known group with RID = DOMAIN_ALIAS_RID_PRINT_OPS
    - built-in well-known group with RID = DOMAIN_ALIAS_RID_BACKUP_OPS
    - built-in well-known group with RID = DOMAIN_ALIAS_RID_REPLICATOR
    - **account domain** well-known group with RID = DOMAIN_GROUP_RID_ADMINS
    - account domain well-known group with RID = DOMAIN_GROUP_RID_SCHEMA_ADMINS
    - account domain well-known group with RID = DOMAIN_GROUP_RID_ENTERPRISE_ADMINS
  - OR, is one of the following well-known security principals:
    - of class user with RID = DOMAIN_USER_RID_ADMIN
    - of class user with RID = DOMAIN_USER_RID_KRBTGT
    - of class group with RID = DOMAIN_GROUP_RID_CONTROLLERS
    - of class group with RID = DOMAIN_GROUP_RID_READONLY_CONTROLLERS

3.1.1.6.1.3 Protection Operation

Every object in the protected set is examined at least once every 120 minutes, every 60 minutes by default, at domain d's PDC FSMO role owner. For any object o where o!nTSecurityDescriptor ≠ AdminSDHolder!nTSecurityDescriptor an originating update is performed replacing o!nTSecurityDescriptor with the value of AdminSDHolder!nTSecurityDescriptor. Other replicas of domain d see the effects of this operation after a delay due to replication.

3.1.1.6.1.4 Configurable State

Let C be the object in the config NC identified by the DN of "CN=Windows NT,CN=Services,CN=Configuration,<forest root DN>". C!dSHeuristics (section 6.1.1.2.4.1.2) is a Unicode string attribute, in which the 16th character, **dwAdminSDExMask**, can optionally be set to cause the protection operation to exclude one or more protected objects.

The valid values of **dwAdminSDExMask** are the characters "0"–"9" and "a"–"f". The value is interpreted as a hex digit, of which each bit represents a specific set of security principals that is to be excluded from the AdminSDHolder protection operation.

The set of security principal objects that are excluded are a member, directly or transitively, of any group in the set defined by bits set in the list below:

- C!dSHeuristics[15] & 0x1 ≠ 0 then DOMAIN_ALIAS_RID_ACCOUNT_OPS
- C!dSHeuristics[15] & 0x2 ≠ 0 then DOMAIN_ALIAS_RID_SYSTEM_OPS
- C!dSHeuristics[15] & 0x4 ≠ 0 then DOMAIN_ALIAS_RID_PRINT_OPS
- C!dSHeuristics[15] & 0x8 ≠ 0 then DOMAIN_ALIAS_RID_BACKUP_OPS

3.1.1.6.2 Reference Update

References

- Variable: dsname
- LDAP attributes: dNReferenceUpdate.
- LDAP classes: infrastructureUpdate.
- Glossary: dsname, Infrastructure FSMO master, NC replica, tombstone, GC.
- IDL_DRSVerifyNames method: see [MS-DRSR] section 4.1.27.
- Well-known Objects

In AD DS, attributes of attribute syntax Object (DS-DN), Object(DN-String), Object(DN-Binary), Object(Access-Point) and Object(OR-Name) can have attribute values that reference objects in an NC for which no NC replica is present on the server. The server does not get a replicated update when an object in the NC replica not present on the server is modified or deleted. In such a case, references to such objects will remain to an old dsname on the server. In order to update these kinds of references, a background task called reference update is run at regular intervals. By default, each reference is examined every two days.

The reference update task is not run on a Global Catalog.

If the Recycle Bin optional feature is not enabled and the Infrastructure FSMO master is not a global catalog, then the reference update task is run only on the Infrastructure FSMO master.

If the Recycle Bin optional feature is enabled, every DC that is not also a global catalog runs the reference update task.

The reference update task does processing as follows:

For each object *P* in each NC replica on the server do the following:

- Let *S* be the set of all attributes of *P* with attribute syntax Object(DS-DN), Object(DN-String), Object(DN-Binary), Object(OR-Name) and Object(Access-Point).
- For each attribute *A* in set *S* and for each value *V* of *A* do the following:
  - If there exists an object with dsname *V* in any NC replica on this DC, then skip this value *V*.
  - If attribute syntax of *A* is Object(DS-DN) then let *G* be *P.A.V.guid_value*. Let *D* be *P.A.V.dn*.
  - Otherwise, let *G* be *P.A.V.object_DN.guid_value*. Let *D* be *P.A.object_DN.dn*.
  - If the Recycle Bin optional feature is not enabled:
    - Retrieve the dsname *N* of object with objectGUID *G* from a GC by calling method IDL_DRSVerifyNames. IDL_DRSVerifyNames is explained in [MS-DRSR] section 4.1.27.
    - If *N*!name ≠ *D* then create an infrastructureUpdate object *I* in the well-known infrastructure update container (see section 6.1.1.4). Set *I*!dNReferenceUpdate to *N*. Delete *I* immediately to turn it to a tombstone.

      Creation of an infrastructureUpdate object *K* with attribute dNReferenceUpdate will trigger an update of all references to dsnames corresponding to *K*!dNReferenceUpdate, as explained in section 3.1.1.5.2.4.
  - If the Recycle Bin optional feature is enabled:
    - Retrieve the dsname *N* and the value *Vgc* of the isRecycled attribute of object with objectGUID *G* from a GC by calling method IDL_DRSVerifyNames. IDL_DRSVerifyNames is explained in [MS-DRSR] section 4.1.27.
    - If *Vgc* is true and attribute *A* is a linked attribute, remove value *V* from attribute *A*. This removal is not replicated to any other DCs.
    - If *N*!name ≠ *D* then replace value *V* of attribute *A* with *N*!name. This replacement is not replicated to any other DCs.
    - If attribute *A* is a link value and the RDN of *N*!name is a delete-mangled RDN (see section 3.1.1.5.5), the value *V* is to be treated as a linked value to or from a deleted-object. That is, the value is not generally visible to LDAP clients unless the LDAP_SHOW_DEACTIVATED_LINK_OID control is used.
    - If attribute *A* is a link value and the RDN of *N*!name is not a delete-mangled RDN (see section 3.1.1.5.5), the value *V* is to be treated as a normal linked value. That is, the value is generally visible to LDAP clients.

3.1.1.6.3 Security Descriptor Propagator Update

References

- LDAP attributes: nTSecurityDescriptor
- Glossary: ACE, naming context (NC), security descriptor (SD)

In Active Directory, SDs can contain ACEs that are inheritable. Thus, modifying the SD on an object can imply a change in the SDs of descendant objects (either by adding or by removing such an inheritable ACE). In order to propagate the changes of inheritable ACEs to descendant objects, each DC runs a background task called the Security Descriptor Propagator Update task. By default, this task is triggered by the following conditions:

- Any modification (originating or replicated) of the nTSecurityDescriptor attribute of any object, except for those modifications done by the Security Descriptor Propagator Update task. Such an object is said to have caused a propagation event.
- Any modification of the DN of an object that results in the object having a different parent, except for those cases where the new parent is a Deleted Objects container. Such an object is said to have caused a propagation event.

The Security Descriptor Propagator Update task performs the following processing.

For each object *P* that has caused a propagation event, the server does the following:

- Initialize a set *S* with the single element *P*.
- While the set *S* is not empty, do the following:
  - Let *T* be an element of set *S*.
  - Enforce all SD requirements from section 6.1.3 on the SD of the object *T*. This might require that a new SD be written to the nTSecurityDescriptor attribute of object *T*. If this is the case, such a modification is not replicated to any other instances of Active Directory. Note that this modification of nTSecurityDescriptor is not a new propagation event; it is considered to be part of the original event that was triggered by the modification of the nTSecurityDescriptor attribute of object *P*.
  - If *T* is not a Deleted Objects container, as described in section 6.1.1.4.2, let *U* be the set of all children of *T* that are in the same naming context as *T*. Add all elements of *U* to the set *S*. The set *U* is said to contain qualifying children of object *T*. All objects that are ever elements of set *S* are said to be qualifying descendants of object *P*.
  - Remove *T* from set *S*.

The replication metadata values (see *AttributeStamp* and *LinkValueStamp* in section 3.1.1.1.9) MUST NOT be modified for any attributes that are updated during the processing shown in the preceding list.

There is no constraint on the number of transactions that the Security Descriptor Propagator Update task uses during processing. Therefore, there is no requirement that at any given time all of the objects that are qualifying descendants of an object whose SD has an inheritable ACE actually have the inheritable ACE. It is possible that there is a period of time during which an object that should contain an inheritable ACE from one of its ancestors will not have that inheritable ACE, pending completion of the Security Descriptor Propagator Update task. Likewise, it is possible there is a period of time during which an inheritable ACE that was removed from one of the object's ancestors is still present on the object. Although the protocol places no boundary or requirements on the length of this period of time, it is recommended that implementations minimize the length of this period of time to improve usability of the directory for clients.

The server MUST guarantee that all inheritable ACEs are eventually propagated to all qualifying descendants of an object that causes a propagation event.

3.1.1.7 NT4 Replication Support

AD DS supports the NT4 replication protocol as specified in [MS-NRPC] section 3.6 by maintaining two variables: *nt4ReplicationState* and *pdcChangeLog*. These variables are referenced by [MS-DRSR] section 4.1.11.3 in order to specify the IDL_DRSGetNT4ChangeLog method. This section normatively specifies the format of these variables and how they are maintained during state changes in AD DS.

This section also normatively specifies the format of the referent of the pmsgOut.V1.pLog field of the DRS_MSG_NT4_CHGLOG_REPLY_V1 response message of the IDL_DRSGetNT4ChangeLog method [MS-DRSR] section 4.1.11.3.

3.1.1.7.1 Format of nt4ReplicationState and pdcChangeLog

3.1.1.7.1.1 nt4ReplicationState

nt4ReplicationState is a tuple containing the following fields:

SamNT4ReplicationUSN: this field, a signed 64-bit value, is an update sequence number for updates that occur in AD DS that are relevant to the NT4 replication protocol. Relevant updates are described in section 3.1.1.7.2.2.

SamCreationTime: this field, a FILETIME, records the timestamp when SamNT4ReplicationUSN is set to one.

BuiltinNT4ReplicationUSN: this field, a signed 64-bit value, is an update sequence number for updates that occur in AD DS that are relevant to the NT4 replication protocol. It is different from SamNT4ReplicationUSN in that this value is used only to identify changes to objects whose objectSid has the **domain prefix** of the **built-in domain SID**.

BuiltinCreationTime: this field, a FILETIME, is used to record the timestamp when BuiltinNT4ReplicationUSN is set to one.

3.1.1.7.1.2 pdcChangeLog

The variable *pdcChangeLog* maintains a sequence of elements, each representing a unique update to Active Directory that is exposed through the NT4 replication protocol ([MS-NRPC] section 3.5).

Though *pdcChangeLog* is an internal variable, its contents are sent over the network.

The *pdcChangeLog* variable is a sequence of CHANGELOG_ENTRY elements. These CHANGELOG_ENTRY elements are defined in [MS-NRPC] section 3.5.4.6.4.

3.1.1.7.2 State Changes

This section describes state changes in AD DS that cause the *nt4ReplicationState* and *pdcChangeLog* variables to change values.

3.1.1.7.2.1 Initialization

nt4ReplicationState and *pdcChangeLog* are reset on Active Domain domain creation (for example, when the first DC in an AD DS domain is installed). See section 3.1.1.7.2.4 for information on resetting the *pdcChangeLog* for the specific values of the variables in this condition.

3.1.1.7.2.2 Directory Updates

Entries are added to the *pdcChangeLog* on select directory updates, specified here. The *pdcChangeLog* is maintained as a circular buffer—once an implementation-specific size limit (64K bytes) is exceeded, the least-recently-added entries are removed to make room for new entries.

If the following condition is true during a directory update, then the following action occurs:

1. Condition

1. The update, create, or delete occurs within the domain NC (both for an originating and replicated update).
2. The AD DS domain is in mixed mode.
3. A condition listed in the Trigger Condition Tables (below) matches the update.

2. Action
   - An entry is added to *pdcChangeLog* with the associated fields in the Trigger Condition Tables that satisfied condition (1.3). The remaining fields in the *pdcChangeLog* entry are as follows:
     1. If the objectSid attribute value of the object being updated has a domain prefix of the built-in domain SID, then DbIndex is 0x1; otherwise, DbIndex is 0x0.
     2. The SerialNumber field is set as follows:
        1. If DbIndex is 0x0, SamNT4ReplicationUSN is incremented by one and the resulting value is used for the SerialNumber field.
        2. If DbIndex is 0x1, BuiltinNT4ReplicationUSN is incremented by one and the resulting value is used for the SerialNumber field.
     3. The SID field is not specified.

Trigger Condition Tables: Database triggers for *pdcChangeLog* update.

- **Trigger Condition**: An update occurs to one or more of the attributes specified in Table A on a domain object or built-in domain object.

pdcChangeLog entry	
Field	**Value**
RelativeId	0x0
Flags	CHANGELOG_SID
DeltaType	AddOrChangeDomain

- **Trigger Condition**: A group object creation or update to one or more of the attributes specified in Table B occurs when the groupType attribute is GROUP_TYPE_ACCOUNT_GROUP.

pdcChangeLog entry	
Field	**Value**
RelativeId	RelativeId of the objectSid attribute value
Flags	CHANGELOG_SID
DeltaType	AddOrChangeGroup
Name	sAMAccountName attribute value

- **Trigger Condition**: A group object creation or update to one or more of the attributes specified in Table B occurs when the groupType attribute is GROUP_TYPE_RESOURCE_GROUP.

pdcChangeLog entry	
Field	**Value**
RelativeId	RelativeId of the objectSid attribute value
Flags	CHANGELOG_SID
DeltaType	AddOrChangeAlias
Name	sAMAccountName attribute value

- **Trigger Condition**: A user object creation or update to one of more of the attribute specified in Table C occurs.

pdcChangeLog entry	
Field	**Value**
RelativeId	RelativeId of the objectSid attribute value
Flags	CHANGELOG_SID
DeltaType	AddOrChangeUser
Name	sAMAccountName attribute value

- **Trigger Condition**: A group object deletion whose groupType attribute value is GROUP_TYPE_ACCOUNT_GROUP occurs.

pdcChangeLog entry	
Field	**Value**
RelativeId	RelativeId of the objectSid attribute value
Flags	0x8
DbType	DeleteGroup
Name	sAMAccountName attribute value

- **Trigger Condition**: A group object deletion whose groupType attribute value is GROUP_TYPE_RESOURCE_GROUP occurs.

pdcChangeLog entry	
Field	**Value**
RelativeId	RelativeId of the objectSid attribute value
Flags	CHANGELOG_SID
DeltaType	DeleteAlias
Name	sAMAccountName attribute value

- **Trigger Condition**: A user object deletion occurs.

pdcChangeLog entry	
Field	**Value**
RelativeId	RelativeId of the objectSid attribute value
Flags	CHANGELOG_SID
DeltaType	DeleteUser
Name	sAMAccountName attribute value

Table A: Domain Attributes for NT4 Replication

Attributes
nTSecurityDescriptor
oEMInformation
minPwdLength
pwdHistoryLength
pwdProperties
maxPwdAge
minPwdAge
lockoutDuration
lockOutObservationWindow
lockoutThreshold

Table B: Group Attributes for NT4 Replication

Attributes
nTSecurityDescriptor
sAMAccountName
description
member

Table C: User Attributes for NT4 Replication

Attributes
sAMAccountName
displayName

Attributes
primaryGroupID
description
comment
homeDirectory
homeDrive
scriptPath
profilePath
userWorkstations
logonHours
accountExpires
userAccountControl
userParameters
countryCode
codePage
pwdLastSet
unicodePwd
dBCSPwd
nTSecurityDescriptor
groupType

3.1.1.7.2.3 Acquiring the PDC Role

When the PDC role is acquired through a FSMO role transfer, one of the following two predicates is true following the transfer:

1. The new PDC's *pdcChangeLog* is in the reset state described in section 3.1.1.7.2.4
2. All of the following are true:
    1. The new PDC's *pdcChangeLog* has the same ordering of entries for all entries that existed in the *pdcChangeLog* on the old PDC during the PDC role transfer.
    2. All updates to the state of objects in the domain NC replica of the old PDC are reflected in the state of objects in the domain NC replica of the new PDC when the transfer is complete.
    3. All updates to the state of objects in domain NC replica on the new PDC that are not present on the old PDC have a corresponding entry in the *pdcChangeLog* on the new PDC, as described in section 3.1.1.7.2.2.

4. The SamNT4ReplicationUSN and BuiltNT4ReplicationUSN variables were increased by adding 0x1000000000 during the transfer.

When predicate (2) above is satisfied after a transfer, the transfer does not cause NT4 BDCs to perform a full synchronization (described in [MS-NRPC] section 3.6). The implementation satisfies predicate (2) above when possible.

Once the PDC role is acquired, the following two entries are added to the *pdcChangeLog*. This notifies NT4 BDCs that the PDC has changed. SamNT4ReplicationUSN and BuiltinNT4ReplicationUSN are updated prior to use in creating these entries.

pdcChangeLog entry	Field	Value
Entry 1	RelativeId Flags DbDelta DbIndex SerialNumber	0x0 CHANGELOG_SID AddOrChangeDomain 0x0 SamNT4ReplicationUSN
Entry 2	RelativeId Flags DbDelta DbIndex SerialNumber	0x0 CHANGELOG_SID AddOrChangeDomain 0x1 BuiltinNT4ReplicationUSN

3.1.1.7.2.4 Resetting the pdcChangeLog

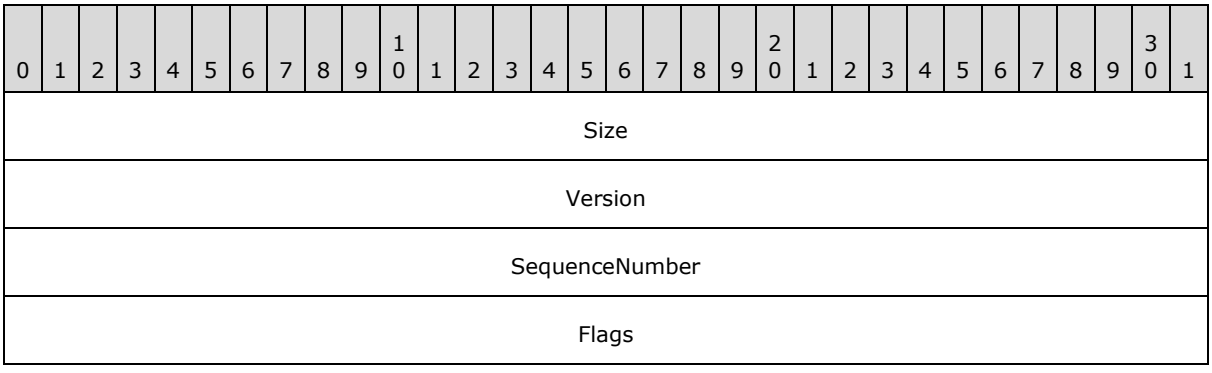
To reset the *pdcChangeLog*, set the array to have 0 elements, set SamCreationTime and BuiltinCreationTime to the current time and SamNT4ReplicationUSN and BuiltinNT4ReplicationUSN to one.

Resetting the *pdcChangeLog* has the effect of causing NT4 BDCs to perform a full sync.

3.1.1.7.3 Format of the Referent of pmsgOut.V1.pLog

The DRS_MSG_NT4_CHGLOG_REPLY_V1 ([MS-DRSR] section 4.1.11.1.4) response message to an IDL_DRSGetNT4ChangeLog request ([MS-DRSR] section 4.1.11) contains a BYTE *pLog field. The format of the referent of this field is not specified in [MS-DRSR] section 4.1.11; it is specified here.

The referent of this field is a CHANGE_LOG_ENTRIES structure:



ChangeLogEntries (variable)
...

Size (4 bytes): The size, in bytes, of the part of the buffer preceding the **ChangeLogEntries** field. Equals 0x00000010.

Version (4 bytes): The version of the message. Equals 0x00000001.

SequenceNumber (4 bytes): The sequence number for the buffer. Is set to 0x00000001 in a response to a IDL_DRSGetNT4ChangeLog request with pmsgIn.V1.pRestart = NULL. The value of pmsgOut.V1.pRestart in any IDL_DRSGetNT4ChangeLog response encapsulates SequenceNumber. In a response to an IDL_DRSGetNT4ChangeLog request with pmsgIn.V1.pRestart ≠ NULL, SequenceNumber is the value encapsulated in pmsgIn.V1.pRestart, plus one.

Flags (4 bytes): Equals 0x00000000. Ignored upon receipt.

ChangeLogEntries (variable): A sequence of CHANGELOG_ENTRY structures. Each CHANGELOG_ENTRY is followed by padding bytes with value zero such that the number of bytes in the CHANGELOG_ENTRY plus the padding is congruent to zero mod 8.

The server stores the total number of bytes in the fixed-length and variable-length portions of the CHANGE_LOG_ENTRIES structure in the DWORD cbLog field of the DRS_MSG_NT4_CHGLOG_REPLY_V1 response message. This field allows the client to determine the number of CHANGELOG_ENTRY structures contained in the CHANGE_LOG_ENTRIES structure.

3.1.1.8 AD LDS Special Objects

AD LDS NCs can contain the following special types of objects: AD LDS users and AD LDS bind proxies. Special processing applies to these types of objects.

3.1.1.8.1 AD LDS Users

An AD LDS user object is a security principal object in AD LDS that contains a password.

If at least one of the following statements applies to an object class within an AD LDS schema, then each instance of that object class functions as an AD LDS user:

1. The object class contains msDS-BindableObject as a static auxiliary class.
2. The object class contains a static auxiliary class that is a subclass of msDS-BindableObject.
3. The object class is a subclass of another object class that satisfies statement 1 or 2.

An AD LDS user object has these special properties and behavior:

- Its objectSid is assigned during Add as specified in section 3.1.1.5.2.4.
- It can be a member of group objects in its AD LDS forest, subject to the limitations on inter-NC references specified in section 3.1.1.2.2.3, Referential Integrity.
- It can be named in an LDAP bind; section 5.1.1.5 specifies the supported authentication mechanisms and protocols. If the bind succeeds, it creates a security context for the LDAP connection as specified in section 5.1.3.4.

- Its password can both be assigned an initial value and updated. Special processing is performed on both the initial assignment and on update. Sections 3.1.1.5.2.2, 3.1.1.5.2.4, 3.1.1.5.3.1, 3.1.1.5.3.2, and 3.1.1.5.3.3 specify this processing.
- Its objectSid can be written into an AD LDS security descriptor, subject to restrictions specified in section 6.1.3.3.

3.1.1.8.2 Bind Proxies

An AD LDS bind proxy is an object that represents a security principal of the underlying operating system; it is not a security principal itself. A bind proxy object does not contain a password.

If at least one of the following statements applies to an object class within an AD LDS schema, then each instance of that object class functions as an AD LDS bind proxy:

1. The object class contains msDS-BindProxy as a static auxiliary class.
2. The object class contains a static auxiliary class that is a subclass of msDS-BindProxy.
3. The object class is a subclass of another object class that satisfies statement 1 or 2.

An AD LDS bind proxy object has these special properties and behavior:

- Its objectSid is assigned during Add and is the SID of some Windows user in a security realm trusted by the machine running the AD LDS DC that performed the Add. For instance, if an AD LDS DC is running on a machine that is joined to an Active Directory domain D, then the objectSid of a bind proxy created by that DC can be a user within D or within the forest that contains D, or within any domain or forest trusted by D or the forest that contains D.
- It can be a member of group objects in its AD LDS forest, subject to the limitations on inter-NC references specified in section 3.1.1.2.2.3, Referential Integrity.
- It can be named in an LDAP bind; section 5.1.1.5 specifies the supported authentication mechanisms and protocols. If the bind succeeds, it creates a **security context** for the LDAP connection as specified in section 5.1.3.4.
- It does not contain a password. Special processing is performed on update to its password attribute, as specified in section 3.1.1.5.3.3, except on Active Directory Application Mode (ADAM) RTW DCs.

3.1.1.9 Optional Features

Beginning with Windows Server 2008 R2 operating system, Active Directory supports a set of optional features. An optional feature is a set of modifications to the Active Directory state model and the Directory Replication Service (DRS) Remote Protocol [MS-DRSR].

Optional features are enabled in some scope. A scope defines the set of DCs participating in the state-model changes that make up the optional feature. Optional features may be forest-wide, domain-wide, or server-wide in scope. A forest-wide optional feature affects the state model of all DCs in the forest when the optional feature is enabled. A domain-wide optional feature affects the state model of all DCs in the domain in which the optional feature is enabled. A server-wide optional feature affects the state model of the DCs in which the optional feature is enabled. AD LDS supports forest-wide and server-wide optional features. In AD LDS, a forest-wide optional feature affects the state model of all AD LDS instances in a configuration set. Domain-wide optional features are not supported in AD LDS.

Scopes are represented by objects in the directory information tree (DIT). The object that represents the forest-wide scope is the Cross-Ref-Container container (see section 6.1.1.2.1). The object that represents a domain-wide scope is the NC root object of the domain. The object that represents a server-wide scope is the nTDSDSA object.

Optional features are represented by instances of the object class msDS-OptionalFeature. Objects representing optional features are stored in the Optional Features container in the Config NC (see section 6.1.1.2.4.1.3).

Optional features are disabled in a scope via the disableOptionalFeature rootDSE modify operation (see section 3.1.1.3.3.27).

Optional features are enabled in a scope via the enableOptionalFeature rootDSE modify operation (see section 3.1.1.3.3.28).

The list of optional features enabled for a scope is stored in the msDS-EnabledFeature attribute on the object representing the scope. The value stored is a reference to the specific enabled optional feature.

The list of scopes in which an optional feature is enabled is stored in the msDS-EnabledFeatureBL attribute on the object representing the optional feature. The values stored are references to the objects representing the scopes where the feature is enabled.

If an optional feature is enabled in some scope, then, depending on the feature, it might be automatically enabled in another scope; for example, the Recycle Bin optional feature (section 3.1.1.9.1).

Optional features are uniquely identified by a GUID. The GUID is stored in the msDS-OptionalFeatureGUID attribute of the object representing the optional feature.

The following procedure determines whether an optional feature is enabled in a scope by using the msDS-EnabledFeature attribute:

```
procedure IsOptionalFeatureEnabled (
    scope: DSNAME, featureGuid: GUID): boolean

  Returns true if scope!msDS-EnabledFeature contains the DN of a
    msDS-optionalFeature object o such that o!msDS-optionalFeatureGUID
    equals featureGuid.

  Returns false otherwise.
```

Permissible scopes for optional features are specified in the msDS-OptionalFeatureFlags attribute on the object representing the optional feature. If an optional feature is permissible for a forest-wide scope, the attribute contains the bit flag FOREST_OPTIONAL_FEATURE (see section 2.2.16). If an optional feature is permissible for a domain-wide scope, the attribute contains the bit flag DOMAIN_OPTIONAL_FEATURE (see section 2.2.16). If an optional feature is permissible for a server-wide scope, the attribute contains the bit flag SERVER_OPTIONAL_FEATURE (see section 2.2.16). More than one flag may be specified, meaning that the optional feature can be enabled in more than one scope. If none of these flags is specified, an optional feature does not have a scope and, therefore, will not be enabled anywhere.

Whether an optional feature can be disabled is specified in the msDS-OptionalFeatureFlags attribute on the object representing the optional feature. If the feature can be disabled, the attribute contains the bit flag DISABLABLE_OPTIONAL_FEATURE. Absence of this flag means that the feature cannot be disabled once it has been enabled.

Optional features may require Active Directory to be at specific functional levels in order to be enabled.

If an optional feature requires a specific forest functional level before it can be enabled, the forest functional level required is stored in the msDS-RequiredForestBehaviorVersion attribute of the object representing the optional feature.

If an optional feature requires a specific forest functional level before it can be enabled in a domain-wide scope, the forest functional level required is stored in the msDS-RequiredDomainBehaviorVersion attribute of the object representing the optional feature.

The following table shows the optional features that are available in specific versions of Windows Server operating system.

Optional feature name	Windows Server 2008 R2 AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
Recycle Bin	X	X	X	X	X	X

3.1.1.9.1 Recycle Bin Optional Feature

The Recycle Bin optional feature is represented by the Recycle Bin Feature Object (see section 6.1.1.2.4.1.3.1).

The Recycle Bin optional feature modifies the DRS Remote Protocol and modifies the way Active Directory processes object deletion, object undeletion, and referential integrity. When the Recycle Bin optional feature is enabled, deleted-objects maintain virtually all of their state, and therefore may be undeleted without loss of information. When the Recycle Bin optional feature is enabled, link valued attributes are maintained both to and from deleted-objects. This is not possible in the unmodified state model. When the Recycle Bin optional feature is enabled, all tombstones are transformed to be recycled-objects, and all the requirements for recycled-objects in section 3.1.1.5.5.1.3 are maintained.

The state model modifications that implement the Recycle Bin optional feature are specified throughout this document, with specific details in sections 3.1.1.1.6, 3.1.1.4.5.37, 3.1.1.4.5.38, 3.1.1.5.3 (especially 3.1.1.5.3.7), 3.1.1.5.5, 3.1.1.6.2, and 6.1.5.5.

The Recycle Bin optional feature is identified by the feature GUID {766ddcd8-acd0-445e-f3b9-a7f9b6744f2a}.

The Recycle Bin optional feature requires a Forest Functional Level of DS_BEHAVIOR_WIN2008R2 or greater.

The Recycle Bin optional feature is forest-wide in scope; it cannot be enabled in only a domain-wide scope or server-wide scope. When the rootDSE modify operation enableOptionalFeature (section 3.1.1.3.3.28) is executed on a given DC to enable the Recycle Bin optional feature, in addition to being added to the list of forest-wide enabled features, the optional feature is also added to the list of server-wide enabled features (see section 3.1.1.9).

The Recycle Bin optional feature cannot be disabled once it is enabled.

Any DC with a behavior version of DS_BEHAVIOR_WIN2008R2 or greater MUST be capable of supporting the Recycle Bin optional feature.

3.1.1.10 Revisions

Sections 3.1.1.10.1, 3.1.1.10.2, and 3.1.1.10.3 apply only to AD DS, not to AD LDS.

3.1.1.10.1 Forest Revision

The forest revision represents the default state of the set of objects that are stored in the directory and required for the forest functionality.

The contents of a forest revision are established when the forest is created. Updates to the forest revision, if necessary (see below), are performed by an implementation-specific upgrade process.

The version of the forest revision consists of two integer parts that are separated by a period: major.minor. Assuming that a forest revision X has the version a.b, and forest revision Y has the version c.d, X has a higher or equal version compared to Y if a>c, or if a=c and b>=d.

See section 6.1.1.2.8 for the way in which the version of the forest revision is stored.

Introducing DCs into a forest is possible only if the version of the forest revision is higher than or equal to the minimum version of forest revision that is required for that DC functional level, as shown in the following table.

DC functional level	Minimum required forest revision
DS_BEHAVIOR_WIN2000	0.0
DS_BEHAVIOR_WIN2003	0.9
DS_BEHAVIOR_WIN2008	2.9
DS_BEHAVIOR_WIN2008R2	5.9
DS_BEHAVIOR_WIN2012	10.9
DS_BEHAVIOR_WIN2012R2	12.10

If the version of the forest revision is lower than the minimum version of forest revision for that DC, the forest revision must be upgraded to a newer version by an implementation-specific forest revision upgrade process before the DC can be added. The upgrade process updates the contents and the version of the forest revision.

3.1.1.10.2 RODC Revision

The RODC revision represents the default state of the set of objects that are stored in the directory and required for RODC functionality.

The contents of the RODC revision are established when the forest is created. Updates to the RODC revision, if necessary (see below), are performed by an implementation-specific upgrade process.

The version of the RODC revision is an integer. See section 6.1.1.2.8 for the way in which the version of the RODC revision is stored.

Introducing an RODC into a forest is possible only if the version of the RODC revision is higher than or equal to the minimum version of RODC revision that is required for the DC functional level of the RODC, as shown in the following table.

DC functional level of the RODC	Minimum required RODC revision
DS_BEHAVIOR_WIN2008	2
DS_BEHAVIOR_WIN2008R2	2
DS_BEHAVIOR_WIN2012	2
DS_BEHAVIOR_WIN2012R2	2

If the version of the RODC revision is lower than the minimum version of RODC revision for that RODC, the RODC revision must be upgraded to a newer version by an RODC revision upgrade process before the RODC can be added. The upgrade process updates the contents and version of the RODC revision.

3.1.1.10.3 Domain Revision

The domain revision represents the default state of the set of objects that are stored in the domain and required for its functionality.

The contents of a domain revision are established when the domain is created. Updates to the domain revision, if necessary (see below), are performed by an implementation-specific upgrade process.

The version of the domain revision consists of two integer parts that are separated by a period: major.minor. Assuming that a domain revision X has the version a.b, and a domain revision Y has the version c.d, X is said to have a higher or equal version compared to Y if a>c, or if a=c and b>=d.

See section 6.1.1.5.4 for the way in which the version of the domain revision is stored.

Introducing DCs into a domain is possible only if the version of the domain revision is higher than or equal to the minimum version of domain revision that is required for that DC functional level, as shown in the following table.

DC functional level	Minimum required domain revision
DS_BEHAVIOR_WIN2000	0.0
DS_BEHAVIOR_WIN2003	0.8
DS_BEHAVIOR_WIN2008	3.8
DS_BEHAVIOR_WIN2008R2	5.8
DS_BEHAVIOR_WIN2012	8.8
DS_BEHAVIOR_WIN2012R2	10.9

If the version of the domain revision is lower than the minimum version of domain revision for that DC, the domain revision must be upgraded to a newer version by a domain revision upgrade process before the DC can be added. The upgrade process updates the contents and the version of the domain revision.

3.1.1.11 Claims

3.1.1.11.1 Informative Overview

This section contains an informative overview of claims issuance and claims transformation in Active Directory. Refer to Claims Procedures (section 3.1.1.11.2) for the normative specification of claims issuance and claims transformation.

Note Claims issuance and claims transformation in Active Directory were introduced in Windows Server 2012 operating system. Constructed claims were introduced in Windows Server 2012 R2 operating system.

3.1.1.11.1.1 Claim

A claim is an assertion about a user's identity and is represented as the following n-tuple.

```
{Type, ValueType, m Values of type ValueType}
```

3.1.1.11.1.2 Claims Dictionary

The Claims Dictionary is a list of objects of type msDS-ClaimType placed in the "CN=Claim Types, CN=Claims Configuration, CN=Services" container in the config NC of Active Directory. The Claims Dictionary is configured by administrators in order to enable claims issuance.

3.1.1.11.1.3 Claim Source

Claims have two sources of values:

- **AD**: Active Directory is the default claim source.
- **Certificate**: Certificate sourced claims originate from the strings provided to the GetClaimsForPrincipal procedure (section 3.1.1.11.2.1) and are single-valued Boolean claims.

Constructed claims are generated dynamically according to a claim-specific algorithm, but are still considered to have **AD** as their source.

3.1.1.11.1.4 Claims Issuance

Active Directory generates claims for a principal using a configuration called the Claims Dictionary. The following is a high-level overview of claims issuance in Active Directory:

1. The claim **Type** of the claim is the value of the name attribute of the msDS-ClaimType object.
2. The claim **Value** or **Values** are retrieved from the source specified in the msDS-ClaimSourceType attribute of the msDS-ClaimType object (or computed dynamically in the case of constructed claims). At least one value must be present for this claim to be issued.
3. The claim **ValueType** is generated based on the claim **Values**.

Refer to the GetClaimsForPrincipal claims procedure (section 3.1.1.11.2.1) for a normative description of claims issuance.

3.1.1.11.1.5 Claims Transformation Rules

Claims transformation rules are stored in the msDS-TransformationRules attribute in the msDS-ClaimsTransformationPolicyType object. Refer to the GetTransformationRulesText claims procedure (section 3.1.1.11.2.13) for the processing details that describe how to obtain the transformation rules from the msDS-TransformationRules attribute.

For an msDS-ClaimsTransformationPolicyType object to be valid, it MUST be stored in the "CN=Claims Transformation Policies, CN=Claims Configuration, CN=Services" container in the config NC of Active Directory.

An msDS-ClaimsTransformationPolicyType object MUST be associated with a **TDO** for a given claims-traversal direction in order to apply the claims transformation rules in the msDS-ClaimsTransformationPolicyType object to sets of claims that traverse the TDO in the specified direction.

Claims transformation rules are configured by administrators.

3.1.1.11.1.6 Claims Transformation

Claims need to be examined, filtered, possibly modified, and reissued when traversing trusts. This process is known as *claims transformation*. Claims transformation is invoked only on certain types of trusts. Refer to [MS-PAC] section 4.1.2.2 for details about when claims transformation is invoked.

Claims transformation uses the trust name and the direction of the traversal of the trust to look up the corresponding msDS-ClaimsTransformationPolicyType object and obtain claims transformation rules from it.

The claims to be transformed and the transformation rules are passed to the Claims Transformation Algorithm [MS-CTA].

The output of the Claims Transformation Algorithm is further processed using the Claims Dictionary to produce claims that are relevant to the new forest in which they are used.

Refer to the TransformClaimsOnTrustTraversal claims procedure (section 3.1.1.11.2.11) for a normative description of claims transformation.

3.1.1.11.2 Claims Procedures

This section defines the logical processing for claim-related operations. The procedure definitions use the pseudocode language defined in [MS-DRSR] section 3.4. This section uses the data structures and types defined in section 2.2.17.

3.1.1.11.2.1 GetClaimsForPrincipal

```
procedure GetClaimsForPrincipal(
    pADPrincipal : ADDRESS OF DSNAME,
    pCertificateStringsArray : set of unicodestring,
    pClaimsBlob : ADDRESS OF CLAIMS_BLOB)
```

This procedure defines the process of generating claims for a principal in Active Directory and returning these claims as a blob in the wire format.

pADPrincipal: The Active Directory principal whose claims need to be generated.

pCertificateStringsArray: A set of Unicode strings.

pClaimsBlob: The output **CLAIMS_BLOB** structure that is filled with encoded claims.

Return Values: This procedure does not return a value.

Logical Processing:

```
principalClass: ObjectClass;
adSourcedClaims: CLAIMS_ARRAY;
certificateSourcedClaims: CLAIMS_ARRAY;
constructedClaims: CLAIMS_ARRAY
adSourcedAndConstructedClaims: CLAIMS_ARRAY
claimsSet : CLAIMS_SET;

principalClass := pADPrincipal^!ObjectClass.ClassId;
adSourcedClaims := null;
certificateSourcedClaims := null;
constructedClaims := null;
claimsSet := null;

GetADSourcedClaims (pADPrincipal, principalClass,
                    ADDRESS OF adSourcedClaims);
GetCertificateSourcedClaims(
    principalClass,
    pCertificateStringsArray,
    ADDRESS OF certificateSourcedClaims);
GetConstructedClaims (pADPrincipal, ADDRESS OF constructedClaims);

/*
  Merge AD-sourced claims and constructed claims into one CLAIMS_ARRAY
*/
adSourcedAndConstructedClaims.usClaimsSourceType := CLAIMS_SOURCE_TYPE_AD;
if (adSourcedClaims.ulClaimsCount > 0)
    adSourcedAndConstructedClaims.ClaimsEntry :=
        adSourcedClaims.ClaimsEntry
    adSourcedAndConstructedClaims.ulClaimsCount :=
        adSourcedClaims.ulClaimsCount;
endif

if (constructedClaims.ulClaimsCount > 0)
    adSourcedAndConstructedClaims.ClaimsEntry[adSourcedAndConstructedClaims.ulClaimsCount]
        := constructedClaims.ClaimsEntry;
    adSourcedAndConstructedClaims.ulClaimsCount :=
        adSourcedAndConstructedClaims.ulClaimsCount + constructedClaims.ulClaimsCount;
endif

if (adSourcedAndConstructedClaims.ulClaimsCount > 0)
    claimsSet.ulClaimsArrayCount := claimsSet.ulClaimsArrayCount + 1;
    claimsSet.ClaimsArrays.add (adSourcedAndConstructedClaims);
endif

if (certificateSourcedClaims.ulClaimsCount > 0)
    claimsSet.ulClaimsArrayCount := claimsSet.ulClaimsArrayCount + 1;
    claimsSet.ClaimsArrays.add (certificateSourcedClaims);
endif

if (claimsSet.ulClaimsArrayCount = 0)
    pClaimsBlob^ := NULL;
    return;
endif
```

```
EncodeClaimsSet(ADDRESS OF claimsSet, pClaimsblob);

return;
```

3.1.1.11.2.2 GetADSourcedClaims

```
procedure GetADSourcedClaims (
    pADPrincipal : ADDRESS OF DSNAME,
    principalClass : ObjectClass,
    pAdSourcedClaims : ADDRESS OF CLAIMS_ARRAY)
```

This procedure is a helper routine that retrieves Active Directory-sourced claims (section 3.1.1.11.1.3) for a given principal from Active Directory using the Claims Dictionary (section 3.1.1.11.1.2).

pADPrincipal: The principal whose Active Directory claims are to be retrieved.

principalClass: The object class of the principal.

pAdSourcedClaims: The address of a **CLAIMS_ARRAY** structure used for the output Active Directory-sourced claims.

Return Values: This procedure does not return a value.

Logical Processing:

```
bIssueClaim : boolean;
claim: CLAIM_ENTRY;
claimConfigContainer : DSName;
bIssueClaim := FALSE;
claim := null;
pAdSourcedClaims^ := null;
claimConfigContainer := DescendantObject( ConfigNC(),
    "CN=Claim Types, CN=Claims Configuration, CN=Services");

pAdSourcedClaims^.usClaimsSourceType := CLAIMS_SOURCE_TYPE_AD;
for (x in children claimConfigContainer)
    if (x!msDS-ClaimSourceType = "AD" &&
       x!msDS-ClaimTypeAppliesToClass in principalClass &&
       ValidateClaimDefinition(x))
         bIssueClaim := TRUE;
    endif

    if (bIssueClaim && pADPrincipal^!(x!msDS-ClaimAttributeSource) ≠ null)
         claim.Id := x!name;
         claim.Type := x!msDS-ClaimValueType;
         claim.ValueCount :=
           pADPrincipal^!(x!msDS-ClaimAttributeSource).count();

         if (x!msDs-ClaimAttributeSource.Syntax = 2.5.5.1)
            claim.Values :=
              pADPrincipal^!(x!msDS-ClaimAttributeSource)[].DN;
         else
            claim.Values := pADPrincipal^!(x!msDS-ClaimAttributeSource)[];
         endif
```

```
            pAdSourcedClaims^.ClaimEntries.Add(claim);
            pAdSourcedClaims^. ulClaimsCount :=
              pAdSourcedClaims^. ulClaimsCount + 1;
        endif

        claim := null;
        bIssueClaim := FALSE;
    endfor
    return;
```

3.1.1.11.2.3 GetCertificateSourcedClaims

```
procedure GetCertificateSourcedClaims (
    principalClass : ObjectClass,
    pCertificateStringsArray : set of unicodestring,
    pCertificateSourcedClaims : ADDRESS of CLAIMS_ARRAY)
```

This procedure is a helper routine that generates certificate-sourced claims (section 3.1.1.11.1.3) from given strings for a given principal type.

principalClass: The object class of the principal for whom the claims are being generated.

pCertificateStringsArray: A set of Unicode strings.

pCertificateSourcedClaims: The address of a **CLAIMS_ARRAY** structure used for the output certificate-sourced claims.

Return Values: This procedure does not return a value.

Logical Processing:

```
bIssueClaim : boolean;
claim : CLAIM_ENTRY;
claimConfigContainer : DSName;

bIssueClaim := FALSE;
claim := null;
pCertificateSourcedClaims^ := null;
claimConfigContainer := DescendantObject( ConfigNC(),
    "CN=Claim Types, CN=Claims Configuration, CN=Services");

pCertificateSourcedClaims^.usClaimsSourceType :=
    CLAIMS_SOURCE_TYPE_CERTIFICATE;

for (x in children claimConfigContainer)
    if (x!msDS-ClaimSourceType = "Certificate" &&
       x!msDS-ClaimTypeAppliesToClass in principalClass &&
       ValidateClaimDefinition(x))
      bIssueClaim := TRUE;
    endif

    if (bIssueClaim && x!msDS-ClaimSource in pCertificateStringsArray)
      claim.Id := x!msDS-ClaimSource;
      claim.Type := x!msDS-ClaimValueType;
      claim.ValueCount := 1;
```

```
        claim.Values := TRUE;
        pCertificateSourcedClaims^.ClaimEntries.Add(claim);
        pCertificateSourcedClaims^.ulClaimsCount :=
            pCertificateSourcedClaims^.ulClaimsCount + 1;
    endif

    claim := null;
    bIssueClaim := FALSE;
endfor

return;
```

3.1.1.11.2.4 GetConstructedClaims

```
procedure GetConstructedClaims (
    pADPrincipal : ADDRESS OF DSNAME,
    principalClass : ObjectClass,
    pConstructedClaims : ADDRESS OF CLAIMS_ARRAY)
```

This procedure is a helper routine that computes constructed claims (section 3.1.1.11.1.3) for a given principal from Active Directory by using the Claims Dictionary (section 3.1.1.11.1.2).

pADPrincipal: The principal whose Active Directory claims are to be retrieved.

principalClass: The object class of the principal.

pConstructedClaims: The address of a **CLAIMS_ARRAY (section 2.2.17.6)** structure that is used for the output constructed claims.

Return Values: This procedure does not return a value.

Logical Processing:

```
bIssueClaim : boolean;
claim: CLAIM_ENTRY;
claimConfigContainer : DSName;

bIssueClaim := FALSE;
claim := null;
pConstructedClaims^ := null;

claimConfigContainer := DescendantObject( ConfigNC(),
    "CN=Claim Types, CN=Claims Configuration, CN=Services");

/*
  Constructed claims use the CLAIMS_SOURCE_TYPE_AD source type.
*/
pConstructedClaims^.usClaimsSourceType := CLAIMS_SOURCE_TYPE_AD;
for (each x in children claimConfigContainer)
    if (x!msDS-ClaimSourceType = "Constructed" &&
       x!msDS-ClaimTypeAppliesToClass in principalClass &&
       ValidateClaimDefinition(x))
         bIssueClaim := TRUE;
    endif

    if (bIssueClaim)
```

```
            /*
              Currently only the AuthenticationSilo claim is supported
            */
            if (x.Name = "ad://ext/AuthenticationSilo")
                claim := GetAuthSiloClaim(pADPrincipal)
                if (claim != null)
                    pConstructedClaims^.ClaimEntries.Add(claim);
                    pConstructedClaims^.ulClaimsCount :=
                        pConstructedClaims^.ulClaimsCount + 1;
                 endif
            endif
        endif
    endfor

    return;
```

3.1.1.11.2.5 EncodeClaimsSet

```
procedure EncodeClaimsSet (
    pClaimsSet : ADDRESS OF CLAIMS_SET,
    pClaimsBlob : ADDRESS OF CLAIMS_BLOB)
```

This procedure is a helper routine that encodes a given claims set into a claims blob.

pClaimsSet: The address of the input **CLAIMS_SET** structure that is to be encoded.

pClaimsBlob: The address of the output **CLAIMS_BLOB** structure that receives the encoded claims set.

Return Values: This procedure does not return a value.

Logical Processing:

```
encodedClaimsSet: BYTE[];
encodedClaimsSetSize: ULONG;
claimsSetMetadata: CLAIMS_SET_METADATA;
encodedClaimsSet := null;
encodedClaimsSetSize := 0;
claimsSetMetadata := null;
pClaimsBlob^ := null;

NdrEncode (pClaimsSet, ADDRESS OF encodedClaimsSet,
           ADDRESS OF encodedClaimsSetSize);

FillClaimsSetMetadata(
    ADDRESS OF encodedClaimsSet,
    ADDRESS OF encodedClaimsSetSize,
    ADDRESS OF claimsSetMetadata);

NdrEncode (
    claimsSetMetadata,
    ADDRESS OF pClaimsBlob^.EncodedBlob,
    ADDRESS OF pClaimsBlob^.ulBlobSizeinBytes);

return;
```

3.1.1.11.2.6 FillClaimsSetMetadata

```
Procedure FillClaimsSetMetadata (
    pByteArray : BYTE ARRAY,
    ulBufferSizeinBytes : ULONG,
    pClaimsSetMetadata : ADDRESS OF CLAIMS_SET_METADATA)
```

This procedure is a helper routine that fills a **CLAIMS_SET_METADATA** structure using a given byte buffer after compressing the buffer based on its size.

pByteArray: A byte array of size *ulBufferSizeinBytes* that is used to fill in the **CLAIMS_SET_METADATA** structure.

ulBufferSizeinBytes: The size of the byte array.

pClaimsSetMetadata: The address of a **CLAIMS_SET_METADATA** structure, whose data is generated from the *pByteArray* parameter.

Return Values: This procedure does not return a value.

Logical Processing:

```
CompressionFormat : CLAIMS_COMPRESSION_FORMAT;
CompressionFormat := COMPRESSION_FORMAT_LZNT1;
pClaimsSetMetadata^ := null;

if (ulBufferSizeinBytes = 0)
    return;
endif

pClaimsSetMetadata^.ulUncompressedClaimsSetSize := ulBufferSizeinBytes;

if (ulBufferSizeinBytes < 0x100)
    pClaimsSetMetadata^.ulClaimsSetSize := ulBufferSizeinBytes;
    pClaimsSetMetadata^.ClaimsSet :=  pByteArray;
    return;
endif

pClaimsSetMetadata^.usCompressionFormat := CompressionFormat;

RunCompressionAlgorithm(
    TRUE,
    CompressionFormat,
    pByteArray,
    ulBufferSizeinBytes,
    ADDRESS OF pClaimsSetMetadata^.ClaimsSet,
    ADDRESS OF pClaimsSetMetadata^.ulClaimsSetSize)

return;
```

3.1.1.11.2.7 RunCompressionAlgorithm

```
procedure RunCompressionAlgorithm (
    compressData : boolean,
    compressionFormat : CLAIMS_COMPRESSION_FORMAT,
    pInByteArray : BYTE ARRAY,
```

```
    ulBufferSizeinBytes : ULONG,
    pOutByteArray : ADDRESS OF BYTE ARRAY,
    pOutByteArraySizeinBytes : ADDRESS OF ULONG)
```

This is a helper method that implements the compression and decompression algorithms listed in section 2.2.17.4. This method compresses or decompresses the given input data using the algorithm identified by the input *compressionFormat* parameter. If the compression algorithm encounters an error during its operation, the output byte array is cleared.

compressData: Specifies the compression direction. If set to TRUE, this method compresses the input data; otherwise, the method decompresses the input data.

compressionFormat: Specifies the compression or decompression algorithm.

pInByteArray: The input byte array of size *ulBufferSizeinBytes* that is to be compressed or decompressed.

ulBufferSizeinBytes: The size of the input byte array.

pOutByteArray: The address of the output byte array.

pOutByteArraySizeinBytes: The address of a **ULONG** that will contain the size of the output byte array.

Return Values: This procedure does not return a value.

Logical Processing:

```
pOutByteArray^ := null;
pOutByteArraySizeinBytes^ := null;

if (compressionFormat = COMPRESSION_FORMAT_LZNT1)
    if compressData
         pOutByteArray^ := CompressUsing_LZNT1;
    else
         pOutByteArray^ := UncompressUsing_LZNT1;
    endif
else if (compressionFormat = COMPRESSION_FORMAT_XPRESS)
    if compressData
         pOutByteArray^ := CompressUsing_XPRESS;
    else
         pOutByteArray^ := UncompressUsing_XPRESS;
    endif
else if (compressionFormat = COMPRESSION_FORMAT_XPRESS_HUFF)
    if compressData
         pOutByteArray^ := CompressUsing_XPRESS_HUFF;
    else
         pOutByteArray^ := UncompressUsing_XPRESS_HUFF;
    endif
else
    pOutByteArray^ := ADDRESS OF pInByteArray;
    pOutByteArraySizeinBytes^ := ulBufferSizeinBytes;
endif

return;
```

3.1.1.11.2.8 NdrEncode

```
procedure NdrEncode (
    pStructX : ADDRESS of struct X,
    pSerializedData : BYTE ARRAY,
    pDataLengthInBytes : ADDRESS OF ULONG)
```

This is a per-structure helper function that serializes a structure into an array of bytes using the NDR Type Serialization engine, as specified in [MS-RPCE] section 2.2.6. This function returns a null output buffer in case of errors.

pStructX: The address of a structure of some type (represented by "X") that needs to be serialized.

pSerializedData: A byte array of length *pDataLengthInBytes* that contains the output serialized data.

pDataLengthInBytes: The address of a **ULONG** that will contain the size of the output byte array.

Return Values: This procedure does not return a value.

3.1.1.11.2.9 NdrDecode

```
procedure NdrDecode (
    pSerializedData : BYTE ARRAY,
    dataLengthInBytes : ULONG,
    pStructX : ADDRESS of struct X)
```

This is a per-structure helper function that deserializes a byte array into a structure using the NDR type deserialization engine, as specified in [MS-RPCE] section 2.2.6. This function returns a null structure as output in case of errors.

pSerializedData: A byte array of length **dataLengthInBytes** that contains the input serialized data.

dataLengthInBytes: The length of the **pSerializedData** byte array.

pStructX: The address of a structure of some type (represented by "X") that receives the deserialized data.

Return Values: This procedure does not return a value.

3.1.1.11.2.10 DecodeClaimsSet

```
procedure DecodeClaimsSet (
    pClaimsBlob : ADDRESS OF CLAIMS_BLOB,
    pClaimsSet : ADDRESS OF CLAIMS_SET)
```

This method decodes the given **CLAIMS_BLOB** structure into a **CLAIMS_SET** structure and performs various validations on it. Upon successful validation, the output **CLAIMS_SET** structure is filled. In the case of errors, an empty output **CLAIMS_SET** structure is returned.

pClaimsBlob: The address of a **CLAIMS_BLOB** structure that is to be decoded.

pClaimsSet: The address of a **CLAIMS_SET** structure that receives the decoded output.

Return Values: This procedure does not return a value.

Logical Processing:

```
claimsSetMetaData : CLAIMS_SET_METADATA;
pByteArray : BYTE[];
ulBufferSizeinBytes: ULONG;

claimsSetMetaData := null;
pByteArray := null;
ulBufferSizeinBytes := 0;
pClaimsSet^ := null;

if (pClaimsBlob^.ulBlobSizeinBytes = 0)
    return;
endif
NdrDecode (
    pClaimsBlob^.EncodedBlob,
    pClaimsBlob^.ulBlobSizeinBytes,
    ADDRESS OF claimsSetMetadata);

if (claimsSetMetadata.ulClaimsSetSize = 0)
   return;
endif

RunCompressionAlgorithm(
              FALSE,
              claimsSetMetadata.usCompressionFormat,
              claimsSetMetadata.ClaimsSet,
              claimsSetMetadata.ulClaimsSetSize,
              ADDRESS OF pByteArray,
              ADDRESS OF ulBufferSizeinBytes);

if (ulBufferSizeinBytes = 0 ||
    ulBufferSizeinBytes ≠ claimsSetMetadata.ulUncompressedClaimsSetSize)
    return;
endif

NdrDecode (pByteArray, ulBufferSizeinBytes, pClaimsSet);

return;
```

3.1.1.11.2.11 TransformClaimsOnTrustTraversal

```
procedure TransformClaimsOnTrustTraversal (
    pInputClaimsBlob : ADDRESS OF CLAIMS_BLOB,
    trustName : unicodestring,
    fIncomingDirection : boolean,
    pOutputClaimsBlob : ADDRESS OF CLAIMS_BLOB) : ULONG
```

This procedure defines the logical processing for transforming a set of claims on trust traversal. This procedure uses the **Claim** data structure defined in [MS-CTA] section 2.1.2 and invokes the Claims Transformation Algorithm ([MS-CTA] section 2.1) for intermediate processing.

pInputClaimsBlob: The address of the **CLAIMS_BLOB** structure that contains the set of claims that are to be transformed.

trustName: The name of the trust that is being traversed.

fIncomingDirection: The direction of traversal. This parameter MUST be set to TRUE if the claims originated outside the trust boundary and are entering the trust boundary; otherwise, this parameter MUST be set to FALSE.

pOutputClaimsBlob: The address of a **CLAIMS_BLOB** structure that receives the transformed claims output.

Return Values: This procedure returns zero upon success or a nonzero result upon failure.

Logical Processing:

```
trustDsName : DSName;
claimsTransformRulesXml : string;
claimsTransformRulesText : string;
status : ULONG;
CTAInputClaims : Claim[];
CTAOutputClaims : Claim[];
outputClaimsUnfiltered : CLAIMS_ARRAY;
systemContainer : DSName;
trustDsName := null;
claimsTransformRulesXml := null;
claimsTransformRulesText := null;
status := 0;
CTAInputClaims := null;
CTAOutputClaims := null;
outputClaimsUnfiltered := null;
systemContainer := DescendantObject( DefaultNC(), "CN=System");

for (x in children systemContainer )
    if (x!name = trustName)
        trustDsName := x
        break;
    endif
endfor

if (trustDsName = null)
    return ERROR_INVALID_PARAMETER;
endif

status := GetClaimsTransformationRulesXml(trustDsName, fIncomingDirection,
                                          ADDRESS OF claimsTransformRulesXml)
if (status ≠ 0 and
     status ≠ ERROR_DS_OBJ_NOT_FOUND)
    pOutputClaimsBlob^ := 0;
    return 0;
endif

if (status = ERROR_DS_OBJ_NOT_FOUND)
     if (fIncomingDirection = FALSE)
           pOutputClaimsBlob^ := pInputClaimsBlob^;
     else
           pOutputClaimsBlob^ := 0;
     endif
endif

if (claimsTransformRulesXml ≠ null)
        status := GetTransformationRulesText (claimsTransformRulesXml,
                                        ADDRESS OF claimsTransformRulesText);
        if (status ≠ 0)
```

```
            pOutputClaimsBlob^ := 0;
        endif
    endif

    GetCTAClaims (pInputClaimsBlob^, ADDRESS OF CTAInputClaims);

    // Invoke the Claims Transformation Algorithm
    // specified in [MS-CTA] section 2.
    status := ClaimsTransformationAlgorithm( CTAInputClaims,
                                             claimsTransformRulesText,
                                             ADDRESS OF CTAOutputClaims);

    if (status ≠ 0)
          pOutputClaimsBlob^ := 0;
          return 0;
    endif

    CollapseMultiValuedClaims (CTAOutputClaims, ADDRESS OF outputClaimsUnfiltered);

    FilterAndPackOutputClaims(outputClaimsUnfiltered,
                              fIncomingDirection, pOutputClaimsBlob);

    return 0;
```

3.1.1.11.2.12 GetClaimsTransformationRulesXml

```
procedure GetClaimsTransformationRulesXml (
    trustDSName : DSNAME,
    fIncomingDirection : boolean,
    pClaimsTransformRulesXml : unicodestring) : ULONG
```

This is a helper procedure that retrieves the transformation rules (section 3.1.1.11.1.5) stored in the directory for a given trust and claims-traversal direction.

trustDSName: The **DSName** of the trust.

fIncomingDirection: The direction of traversal. This parameter MUST be set to TRUE if the caller requires transformation rules for claims that are entering the trust boundary; otherwise, this parameter MUST be set to FALSE.

pClaimsTransformRulesXML: The XML-encapsulated rules-text that is read directly from the directory.

Return Values: This procedure returns zero when it successfully returns the claims transformation rules. It returns ERROR_DS_OBJ_NOT_FOUND when no claims transformation rules are configured for the given input. Other errors are returned for all other conditions including invalid input parameters and the condition wherein the claims transformation is incorrectly configured.

Logical Processing:

```
claimsTransformObject : DSNAME;
status : ULONG;
allowedClaimsTransformPolicies : DSName;

pClaimsTransformRulesXml^ := NULL;
claimsTransformationObject := NULL;
```

```
status := 0;
allowedClaimsTransformPolicies := DescendantObject(ConfigNC(),
  "CN=Claims Transformation Policies, CN=Claims Configuration, CN=Services");

if (trustDSName = null)
    return ERROR_INVALID_PARAMETER;
endif
if (fIncomingDirection)
    claimsTransformObject :=
      trustDSName!msDS-IngressClaimsTransformationPolicy;
else
    claimsTransformObject :=
      trustDSName!msDS-EgressClaimsTransformationPolicy;
endif

if (claimsTransformObject = NULL)
    return ERROR_DS_OBJ_NOT_FOUND;
endif

if (claimsTransformObject not in children allowedClaimsTransformPolicies)
    return ERROR_INVALID_PARAMETER;
endif

pClaimsTransformRulesXml^ :=
  ClaimsTransformationObject!msDS-TransformationRules;

return 0;
```

3.1.1.11.2.13 GetTransformationRulesText

```
procedure GetTransformationRulesText (
    claimsTransformRulesXML : unicodestring,
    claimsTransformRulesText : unicodestring) : ULONG
```

This procedure validates the given string for the expected XML encapsulation of claims transformation rules stored in the directory and retrieves the plain-text claims transformation rules from the XML. This procedure assumes that generic XML validation and read routines are available based on [XMLSCHEMA2/2].

claimsTransformRulesXML: The XML-encapsulated rules text that was read from the directory.

claimsTransformRulesText: The rules text that is extracted from the given input.

Return Values: This procedure returns zero upon success along with the claims transformation rules text; otherwise, this procedure returns an error.

Logical Processing:

```
status : ULONG;
claimsTransformRulesText := NULL;
if (claimsTransformRulesXML = NULL)
    return 0;
endif

//Validate generic XML based on generic schema definition in [XMLSCHEMA2/2]
status := Validate_Generic_Xml( claimsTransformRulesXML: XML );
```

```
if (status ≠ 0)
    return status;
endif

//Search generic XML with given XPATH based on [XMLSCHEMA2/2]
status := Find_Matching_Attribute(claimsTransformRulesXML: XML,
                                  "ClaimsTransformationPolicy\Rules": XPATH,
                                  "Version": Attribute,
                                  1: AttributeValue);
if (status ≠ 0)
    return status;
endif

//Read generic XML element with given XPATH based on [XMLSCHEMA2/2]
status:= Read_Element_CDATA(claimsTransformRulesXML: XML,
                            "ClaimsTransformationPolicy\Rules": XPATH,
                            claimsTransformRulesText: Output Value);

return status;
```

3.1.1.11.2.14 GetCTAClaims

```
procedure GetCTAClaims (
    inputClaimsBlob : CLAIMS_BLOB,
    outputCTAClaims : set of Claim)
```

This is a helper procedure that converts a **CLAIMS_BLOB** into a set of Claim structures, which are defined in [MS-CTA] section 2.1.2.

inputClaimsBlob: The input **CLAIMS_BLOB** structure.

outputCTAClaims: The set of output CTA Claim structures.

Return Values: This procedure does not return a value.

Logical Processing:

```
inputClaimsSet : CLAIMS_SET;
valueType : string;
inputClaimsSet := null;
outputCTAClaims := null;

DecodeClaimsSet(ADDRESS OF inputClaimsBlob, ADDRESS OF inputClaimsSet);

for each array in inputClaimsSet.ClaimsArrays
    for each claim in array.ClaimEntries
        if (claim.Type = CLAIM_TYPE_INT64)
            valueType := "int64";
        else if (claim.Type = CLAIM_TYPE_UINT64)
            valueType := "uint64";
        else if (claim.Type = CLAIM_TYPE_BOOLEAN)
            valueType := "boolean";
        else if (claim.Type = CLAIM_TYPE_STRING)
            valueType := "string";
        endif
```

```
        for each value in claim.Values
          outputCTAClaims.Add(TYPE = claim.Id, VALUE_TYPE = valueType,
                              VALUE = value);
        endfor
    endfor
endfor
```

3.1.1.11.2.15 CollapseMultiValuedClaims

```
procedure CollapseMultiValuedClaims (
    cTAClaims : set of Claim,
    pOutputClaims : ADDRESS OF CLAIMS_ARRAY) : ULONG
```

This is a helper procedure that converts a given set of Claim structures (defined in [MS-CTA] section 2.1.2) into a **CLAIMS_ARRAY** structure. This procedure also aggregates more than one **single-valued claim** of the same type, removes any duplicates from each aggregate, and collapses the remaining single-valued claims in that aggregate into one **multi-valued claim**.

cTAClaims: The input set of Claim structures.

pOutputClaims: The address of the output **CLAIMS_ARRAY** structure.

Return Values: This procedure returns zero upon success or an error otherwise.

Logical Processing:

```
tempClaim : CLAIM_ENTRY;
valueType : USHORT;

tempClaim := null;
valueType := 0;

for each claim1 in cTAClaims
        if (claim1.VALUE_TYPE = "int64")
            valueType := CLAIM_TYPE_INT64;
        else if (claim1.VALUE_TYPE = "uint64")
            valueType := CLAIM_TYPE_UINT64;
        else if (claim1.VALUE_TYPE = "boolean")
            valueType := CLAIM_TYPE_BOOLEAN;
        else if (claim1.VALUE_TYPE = "string")
            valueType := CLAIM_TYPE_STRING;
        endif

       tempClaim := (Id = claim1.TYPE, Type = valueType,
           ValueCount = count of claim1.VALUE, Values = claim1.VALUE);
       for each claim2 in (cTAClaims - claim1)
         if (claim1.TYPE = claim2.TYPE and
             claim1.VALUE_TYPE = claim2.VALUE_TYPE and
             (claim2.VALUE_TYPE NOT in tempClaim.Values))
             tempClaim.Values := tempClaim.Values + Claim2.VALUE;
          endif
       endfor
    pOutputClaims^.claims := pOutputClaims^.claims + tempClaim;
endfor
```

```
return 0;
```

3.1.1.11.2.16 FilterAndPackOutputClaims

```
procedure FilterAndPackOutputClaims (
    inputClaims : CLAIMS_ARRAY,
    fIncomingDirection : boolean,
    pOutputClaimsBlob : ADDRESS OF CLAIMS_BLOB) : ULONG
```

This is a helper procedure that filters and packs the given **CLAIMS_ARRAY** structure using the Claims Dictionary (3.1.1.11.1.2) in the forest. Filtering is done only for claims in the incoming direction as indicated by the *fIncomingDirection* parameter, and involves the removal of any claims whose types are not defined in the dictionary. Packing of claims involves sorting them into **CLAIMS_ARRAY** structures based on the claims source type as listed in the Claims Dictionary, and packing them into a **PCLAIMS_BLOB** structure.

inputClaims: The input **CLAIMS_ARRAY** structure that is to be filtered.

fIncomingDirection: The direction of traversal. This parameter MUST be set to TRUE if the claims originated outside the trust boundary and are entering the trust boundary; otherwise, this parameter MUST be set to FALSE.

pOutputClaimsBlob: The address of a **CLAIMS_BLOB** structure for the output.

Return Values: This procedure returns zero upon success or an error otherwise.

Logical Processing:

```
status : ULONG;
claimConfigContainer : DSName
outputClaimsSet : CLAIMS_SET;
fMatchFound : boolean;
claimType : CLAIMS_SOURCE_TYPE;
status := 0;
claimConfigContainer := DescendantObject( ConfigNC(),
    "CN=Claim Types, CN=Claims Configuration, CN=Services");

fMatchFound := FALSE;
claimType := null;
pOutputClaimsBlob^ := null;
outputClaimsSet := null;

if (status ≠ 0)
    return status;
endif

outputClaimsSet.ClaimsArrays[0].ClaimsSourceType := CLAIMS_SOURCE_TYPE_AD;
outputClaimsSet.ClaimsArrays[1].ClaimsSourceType :=
  CLAIMS_SOURCE_TYPE_CERTIFICATE;

for each claim in inputClaims.ClaimEntries
     fMatchFound := FALSE;

     for (each claimdef in children claimConfigContainer &&
          NOT fMatchFound && ValidateClaimDefinition(claimdef))
          if (claimdef!msDS-ClaimSourceType = "Certificate")
```

```
            claimType := CLAIMS_SOURCE_TYPE_CERTIFICATE;
        else if (claimdef!msDS-ClaimSourceType = "AD")
            claimType := CLAIMS_SOURCE_TYPE_AD;

        else if (claimdef!msDS-ClaimSourceType = "TransformPolicy")
            claimType := CLAIMS_SOURCE_TYPE_AD;
        endif

        if (claimdef!Enabled AND
            claim.Id = claimdef!name AND
               claim.Type = claimdef!msDS-ClaimValueType)

             // Filter and sort claims in the incoming direction
             if (fIncomingDirection)
                   if (claimType = CLAIMS_SOURCE_TYPE_CERTIFICATE)
                       outputClaimsSet.ClaimsArrays[1].ClaimEntries =
                           outputClaimsSet.ClaimsArrays[1].ClaimEntries +
                               claim;
                   else if (claimType = CLAIMS_SOURCE_TYPE_AD)
                       outputClaimsSet.ClaimsArrays[0].ClaimEntries =
                           outputClaimsSet.ClaimsArrays[0].ClaimEntries +
                               claim;
                   endif
             endif
             fMatchFound := TRUE;
        endif
    endfor

    // Sort claims on the outgoing direction
    if (!fIncomingDirection)
        if (claimType = CLAIMS_SOURCE_TYPE_CERTIFICATE)
            outputClaimsSet.ClaimsArrays[1].ClaimEntries =
                outputClaimsSet.ClaimsArrays[1].ClaimEntries + claim;
         else
            outputClaimsSet.ClaimsArrays[0].ClaimEntries =
                outputClaimsSet.ClaimsArrays[0].ClaimEntries + claim;
        endif
    endif
endfor

EncodeClaimsSet(ADDRESS OF outputClaimsSet, pOutputClaimsBlob);

return 0;
```

3.1.1.11.2.17 ValidateClaimDefinition

```
procedure ValidateClaimDefinition (
    claimDefinition : DSNAME) : Boolean
```

This is a helper procedure that validates a claim definition defined in the Claims Dictionary (section 3.1.1.11.1.2) in the forest. The validation ensures that the correct attribute values are populated in the claim definition.

claimDefinition: The **DSNAME** of the claim definition in the Claims Dictionary that needs to be validated.

Return Values: This procedure returns TRUE if the claim definition is valid and FALSE otherwise.

Logical Processing:

```
status : Boolean;
status := FALSE;

if (claimDefinition = null ||
    claimDefinition!name = null ||
    NOT claimDefinition!Enabled ||
     claimDefinition!msDS-ClaimValueType = null)
    return status;
endif

if (claimDefinition!msDS-ClaimSourceType = "Certificate" &&
    claimDefinition!msDS-ClaimAttributeSource = null &&
    claimDefinition!msDS-ClaimSource ≠ null &&
    claimDefinition!msDS-ClaimValueType = CLAIM_TYPE_BOOLEAN)
    status := TRUE;

else if (claimDefinition!msDS-ClaimSourceType = "AD" &&
         claimDefinition!msDS-ClaimAttributeSource ≠ null &&
         claimDefinition!msDs-ClaimAttributeSource.Syntax in
             {2.5.5.1, 2.5.5.2, 2.2.5.8, 2.5.5.9, 2.5.5.12, 2.5.5.15, 2.5.5.16} &&
         claimDefinition!msDS-ClaimValueType ≠ null)

         if (claimDefinition!msDs-ClaimAttributeSource.Syntax in
             {2.5.5.1, 2.5.5.12, 2.5.5.15} &&
             claimDefinition!msDS-ClaimValueType = CLAIM_TYPE_STRING)
              status := TRUE;
         endif

         if (claimDefinition!msDs-ClaimAttributeSource.Syntax = 2.5.5.2 &&
             claimDefinition!msDS-ClaimValueType = CLAIM_TYPE_UINT64)
              status := TRUE;
         endif

         if (claimDefinition!msDs-ClaimAttributeSource.Syntax in {2.5.5.9, 2.5.5.16} &&
             claimDefinition!msDS-ClaimValueType = CLAIM_TYPE_INT64)
              status := TRUE;
         endif

         if (claimDefinition!msDs-ClaimAttributeSource.Syntax = 2.5.5.8 &&
             claimDefinition!msDS-ClaimValueType = CLAIM_TYPE_BOOLEAN)
              status := TRUE;
         endif

else if (claimDefinition!msDS-ClaimSourceType = "TransformPolicy" &&
         claimDefinition!msDS-ClaimAttributeSource = null &&
         claimDefinition!msDS-ClaimSource = null)

    status := TRUE;

else if (claimDefinition!msDS-ClaimSourceType = "Constructed" &&
         claimDefinition!msDS-ClaimAttributeSource = null &&
         claimDefinition!msDS-ClaimSource = null)

    status := TRUE;
```

```
endif

return status;
```

3.1.1.11.2.18 GetAuthSiloClaim

```
procedure GetAuthSiloClaim (
    pADPrincipal : ADDRESS OF DSNAME) : CLAIM_ENTRY
```

This is a helper procedure that computes the value of the ad://ext/AuthenticationSilo constructed claim type for the specified principal.

pADPrincipal: The Active Directory principal to return an AuthenticationSilo claim for, if applicable.

Return Values: This procedure returns a **CLAIM_ENTRY (section 2.2.17.5)** if the specified principal is a member of an authentication silo; otherwise NULL.

Logical Processing:

```
claim : CLAIM_ENTRY;
parentNC : DSName
siloMember : DSName
memberOfSilo : Boolean;
assignedSilo : DSName

/*
  AuthSiloClaim is not issued until the domain
  functional level is at DS_BEHAVIOR_WIN2012R2
  or higher.
*/
parentNC := GetObjectNC(pADPrincipal)
if (parentNC!msDS-BehaviorVersion < DS_BEHAVIOR_WIN2012R2)
  return NULL
endif

/*
  Check if user is assigned to an enforced silo.
*/
assignedSilo := pADPrincipal!msDS-AssignedAuthNPolicySilo
if (assignedSilo = NULL ||
    assignedSilo!msDS-AuthNPolicySiloEnforced = FALSE)
  return NULL
endif

/*
  Check if silo is configured with the user as a member.
*/
memberOfSilo := FALSE
foreach (siloMember in assignedSilo!msDS-AuthNPolicySiloMembers)
  if (siloMember = pADPrincipal)
      memberOfSilo := TRUE
      break
  endif
endfor

if (memberOfSilo = FALSE)
```

```
    return NULL
endif

/*
   Fill in the claim details and return the claim.
*/
claim.Id := "ad://ext/AuthenticationSilo";
claim.Type := CLAIM_TYPE_STRING
claim.ValueCount := 1
claim.Values := assignedSilo.name

return claim;
```

3.1.1.12 NC Rename

NC Rename is an operation that runs on a single domain controller (DC) and changes the identity and identity-related information of NC replicas hosted on the DC. Except where noted, these changes are strictly local to the abstract data of the DC (that is, the changes are not replicated). Because of this fact, NC Rename can result in multiple DCs wherein each DC hosts an NC replica of the same NC, but each DC has different values for the abstract data relating to that NC. If such diverging changes are performed, the protocol places no restriction on the behavior of the DCs that hold the divergent abstract data. No mechanism in the protocol prevents such diverging changes. It is recommended to users of the NC Rename operation that great care be taken to make such possibly diverging changes on every DC that is affected by the operation, thereby avoiding such divergence.

To accomplish an NC Rename, three general classes of change need to be made. First, attributes directly associated with the name of the NC need to be modified. These attributes include such things as the NetBIOS name and the fully qualified domain name (FQDN) of the NC. Second, objects and attributes associated with the interdomain trusts that a domain NC is a part of need to be modified. These objects and attributes include such things as trusted domain objects (TDOs) and **interdomain trust accounts**. Third, the crossRef objects associated with the NCs need to be modified. Additionally, some changes are made to reflect the fact that the preceding types of changes have been completed.

NC Rename can be used to rename both domain NCs and application NCs. In the case of application NCs, there are no interdomain trusts to update.

3.1.1.12.1 Abstract Data Types

An NC Rename operation is specified by an instance of the NCRenameDescription tuple. This section describes that tuple, including the tuple types that are included directly or indirectly in the NCRenameDescription tuple.

3.1.1.12.1.1 FlatName

```
type FlatName = A string composed of any alphanumeric characters except the quote character
(") and characters ',' and '<'.
```

Instances of the FlatName type exist as fields of tuples of types InterdomainTrustAccountDescription (section 3.1.1.12.1.4), TrustedDomainObjectDescription (section 3.1.1.12.1.5), NCDescription (section 3.1.1.12.1.6), DomainDescriptionElements (section 3.1.1.12.1.7), and NewTrustParentElements (section 3.1.1.12.1.9).

3.1.1.12.1.2 SPNValue

```
type SPNValue = A string that does not contain the quote character (").
```

Instances of the SPNValue type exist as members of the **SPNs** field of the ServerDescription (section 3.1.1.12.1.3) tuple.

3.1.1.12.1.3 ServerDescription

```
type ServerDescription = [
    serverGuid: GUID,
    ExistingDN: DN,
    SPNs: A set containing 1 or more SPNValue elements
 ]
```

An instance of a ServerDescription is a description of a specific object of class computer in the directory. Instances of ServerDescription exist as members of the **Servers** field of a DomainDescriptionElements tuple (section 3.1.1.12.1.7).

serverGuid: Holds the value of the objectGUID attribute on the object. The value of this field is unique across all instances of ServerDescription.

ExistingDN: Holds the DN of the object. The value of this field is unique across all instances of ServerDescription.

SPNs: A set of SPNValue elements (section 3.1.1.12.1.2) to be used in pre-process verification. Successful verification requires that values in the **SPNs** field also exist as values of the attribute SPN on the object. For more information, see section 3.1.1.12.1.4.

3.1.1.12.1.4 InterdomainTrustAccountDescription

```
type InterdomainTrustAccountDescription = [
    Guid: GUID,
    ParentDNFromDomainDN: DN,
    ExistingFlatName: FlatName
    NewFlatName: FlatName
 ]
```

An InterdomainTrustAccountDescription is a description of a specific interdomain trust account object (see section 6.1.6.8) stored in the directory and the changes to be performed as part of the NC Rename operation. Instances of InterdomainTrustAccountDescription exist as members of the **InterdomainTrustAccounts** field of the DomainDescription tuple (section 3.1.1.12.1.8).

GUID: Holds the value of the objectGUID attribute of the object. The value of this field is unique across all instances of InterdomainTrustAccountDescription.

ParentDNFromDomainDN: Holds the DN that, when prepended to the **ExistingDN** field of the instance of the DomainDescription tuple that contains this instance of an InterdomainTrustAccountDescription as an element of the **InterdomainTrustAccounts** field (section 3.1.1.12.1.7), results in the DN of the object that is the parent of the interdomain trust account object.

ExistingFlatName: Holds the value of the sAMAccountName attribute of the object. The value of this field is unique across all instances of InterdomainTrustAccountDescription.

NewFlatName: Holds the value to which the sAMAccountName attribute on the object is to be set as part of the NC Rename operation. The value of this field is unique across all instances of InterdomainTrustAccountDescription. This value is a valid SAM account name.

3.1.1.12.1.5 TrustedDomainObjectDescription

```
type TrustedDomainObjectDescription = [
    Guid: GUID,
    SID: SecurityIdentifier,
    ExistingTrustPartnerDNSName: DNSAddress,
    NewTrustPartnerDNSName: DNSAddress,
    NewTrustPartnerFlatName: FlatName
]
```

A TrustedDomainObjectDescription is a description of a specific interdomain trust account object that is stored in the directory and the changes to be performed as part of the NC Rename operation. Instances of TrustedDomainObjectDescription exist as members of the **TrustedDomainObjects** field of a DomainDescriptionElements tuple (section 3.1.1.12.1.7).

Guid: Holds the value of the objectGUID attribute on the object. The value of this field is unique across all instances of TrustedDomainObjectDescription.

SID: Holds the value of the objectSid attribute of the object. The value of this field is unique across all instances of TrustedDomainObjectDescription.

ExistingTrustPartnerDNSName: Holds the value of the trustPartner attribute on the object. The value of this field is unique across all instances of TrustedDomainObjectDescription.

NewTrustPartnerDNSName: Holds the value that the trustPartner attribute of the object is to be set to as part of the NC Rename operation. The value of this field is unique across all instances of TrustedDomainObjectDescription.

NewTrustPartnerFlatName: Holds the value that the flatName attribute of the object is to be set to as part of the NC Rename operation. The value of this field is unique across all instances of TrustedDomainObjectDescription. This value is a valid value for the flatName attribute.

3.1.1.12.1.6 NCDescription

```
type NCDescription = [
    Guid: GUID,
    ExistingDN: DN,
    NewDN: DN,
    CrossRefGuid: GUID,
    NewDNSName: DNSAddress,
    ExistingFlatName: FlatName
 ]
```

An NCDescription is a description of a specific NC replica and the changes to be performed as part of the NC Rename operation. Instances of NCDescription exist as members of the **AppNCs** field of an NCRenameDescription tuple (section 3.1.1.12.1.11), indirectly as the **RootDomain** field of an NCRenameDescription tuple, and indirectly as members of the **TrustTreeRootDomains** and **TrustTreeNonRootDomains** fields of an NCRenameDescription tuple.

Two objects in the directory are referenced by this tuple: the **NCRoot** and the **NCCrossRef**, as defined below.

Guid: Holds the value of the objectGUID attribute of the object that is the root of the NC replica. The value of this field is unique across all instances of NCDescription. This object is referred to here as the "**NCRoot** object".

ExistingDN: Holds the DN of the **NCRoot** object. The value of this field is unique across all instances of NCDescription.

NewDN: Holds the value that the DN of the **NCRoot** object should be set to as part of the NC Rename operation. The value of this field is unique across all instances of NCDescription.

CrossRefGuid: Holds the value of the objectGUID attribute on the object of class crossRef in the Partitions container whose nCName attribute holds the value of the **ExistingDN** field. The value of this field is unique across all instances of NCDescription. This object is referred to here as the "**NCCrossRef** object".

NewDNSName: Holds the value that the dnsRoot attribute of the **NCCrossRef** object is to be set to as part of the NC Rename operation. The value of this field is unique across all instances of NCDescription.

ExistingFlatName: Holds the value that the nETBIOSName attribute of the **NCCrossRef** object is to be set to as part of the NC Rename operation. The value of this field is unique across all instances of NCDescription. This field is a valid NetBIOS name.

3.1.1.12.1.7 DomainDescriptionElements

```
type DomainDescriptionElements = [
    ExistingDNSName: DNSAddress,
    NewFlatName: FlatName,
    TrustedDomainObjects: a set containing 1 or more
        TrustedDomainObjectDescription tuples,
    InterdomainTrustAccounts: A set containing 1 or more
        InterdomainTrustAccountDescription tuples,
    CountTrusts: A 32-bit integer that contains the number of elements in
        TrustedDomainObjects,
    Servers: a set containing 1 or more ServerDescription tuples
]
```

A DomainDescriptionElements tuple is a partial description of a specific domain NC and the changes to be performed as part of the NC Rename operation. Tuples of this type are never encountered. This type exists as a partial definition of the DomainDescription tuple (section 3.1.1.12.1.8). Since a DomainDescriptionElements tuple is always part of a DomainDescription tuple, and since a DomainDescription tuple implies an NCDescription tuple, an **NCRoot** and an **NCCrossRef** object are used in the following description.

ExistingDNSName: Holds the value of the dnsRoot attribute of the crossRef object. The value of this field is unique across all instances of DomainDescription.

NewFlatName: Holds the value to which the nETBIOSName attribute of the **NCCrossRef** object is to be set. The value of this field is unique across all instances of DomainDescription.

TrustedDomainObjects: Holds a set of TrustedDomainObjectDescription tuples (section 3.1.1.12.1.5). The value of this field is unique across all instances of DomainDescription. The TrustedDomainObjectDescription tuples are also unique across all instances of

TrustedDomainObjectDescription. Each element of this field is a TrustedDomainObjectDescription tuple describing an object that exists in the domain NC replica described by the DomainDescription tuple. This field contains one TrustedDomainObjectDescription for each trusted domain object (TDO) that is present in the NC replica.

InterDomainTrustAccounts: Holds a set of InterDomainTrustAccountDescription tuples (section 3.1.1.12.1.4). The value of this field is unique across all instances of DomainDescription. The InterDomainTrustAccountDescription tuples are also unique across all instances of InterDomainTrustAccountDescription. Each element of this field is an InterDomainTrustAccountDescription tuple describing an object that exists in the domain NC replica described by the DomainDescription tuple. This field contains one InterDomainTrustAccountDescription for each interdomain trust account object that is present in the NC replica.

CountTrusts: Contains the number of elements in the set for the **TrustedDomainObjects** field.

Servers: Holds a set of ServerDescription tuples (section 3.1.1.12.1.3). The value of this field is unique across all instances of DomainDescription. The ServerDescription tuples are also unique across all instances of ServerDescription. Each element of this field is a ServerDescription tuple describing an object that exists in the domain NC replica described by the DomainDescription tuple. This field contains one ServerDescription for each DC that holds a full replica of the domain NC.

3.1.1.12.1.8 DomainDescription

A DomainDescription is a tuple containing the union of all elements of an NCDescription tuple (section 3.1.1.12.1.6) and a DomainDescriptionElements tuple (section 3.1.1.12.1.7). It describes a domain NC in the forest and the changes to be performed as part of the NC Rename operation. Because a DomainDescription is a superset of an NCDescription, wherever a tuple of type NCDescription is specified in a production rule (see 3.1.1.12.2.1), a tuple of type DomainDescription can be used. A similar statement can be made for a tuple of type DomainDescriptionElements.

When used as an NCDescription, the elements from DomainDescriptionElements are ignored, and vice versa.

3.1.1.12.1.9 NewTrustParentElements

```
type NewTrustParentFlatName = [
    NewTrustParentFlatName: FlatName
]
```

A NewTrustParentElements tuple is a partial description of a specific domain NC that is to have a new trust parent as the result of an NC Rename operation, in addition to the changes to be performed as part of the NC Rename operation. Tuples of this type are never encountered. This type exists as a partial definition of the DomainWithNewTrustParentDescription tuple (section 3.1.1.12.1.10).

NewTrustParentFlatName: Holds the value that the trustParent attribute of the crossRef object is to be set to as part of the NC Rename operation.

3.1.1.12.1.10 DomainWithNewTrustParentDescription

A DomainWithNewTrustParentDescription is a tuple containing the union of all elements of a DomainDescription tuple (section 3.1.1.12.1.8) and a NewTrustParentElements tuple (section 3.1.1.12.1.9). It describes a domain NC replica that is to have a new trust parent as a result of an NC Rename operation, in addition to the changes to be performed as part of the NC Rename operation. Because a DomainWithNewTrustParentDescription tuple is a superset of a DomainDescription tuple, wherever a tuple of type DomainDescription is specified in a production rule, a tuple of type DomainWithNewTrustParentDescription can be used. When used as a DomainDescription, the elements from NewTrustParentElements are ignored. Similarly, because a DomainWithNewTrustParentDescription tuple is a superset of an NCDescription tuple (section 3.1.1.12.1.6), wherever a tuple of type NCDescription is specified in a production rule, a tuple of type DomainWithNewTrustParentDescription can be used. When used as an NCDescription, the elements from NewTrustParentElements and DomainDescriptionElements (section 3.1.1.12.1.7) are ignored.

3.1.1.12.1.11 NCRenameDescription

```
type NCRenameDescription = [
    NewReplicationEpoch: 32-bit integer,
    ConfigurationNCGuid: GUID,
    AppNCs: A set containing 0 or more NCDescription tuples,
    RootDomain: DomainDescription,
    TrustTreeRootDomains: A set containing 0 or more
        DomainDescription tuples,
    TrustTreeNonRootDomains: A set containing 0 or more
        DomainWithNewTrustParentDescription tuples,
    AllDomains: A set containing references to DomainDescription tuples
        and DomainWithNewTrustParentDescription tuples. This set has at least
        one element.
    DomainsCount: A 32-bit integer that contains the number of elements in
        the AllDomains field.
    AllNCs: A set containing references to NCDescription tuples,
        DomainDescription tuples, and DomainWithNewTrustParentDescription
        tuples. This set has at least one element.
]
```

An NCRenameDescription tuple describes an NC Rename operation. Tuples of this type are provided as input to an NC Rename operation.

NewReplicationEpoch: Holds the value to which the msDS-ReplicationEpoch attribute of the NTDS Settings object (section 6.1.1.2.2.1.2.1.1) of the DC performing the NC Rename operation is to be set. It is also used in preprocessing verification.

ConfigurationNCGuid: Holds the value of the objectGUID attribute of the root object of the config NC.

AppNCs: Holds a set of NCDescription tuples (section 3.1.1.12.1.6). This field contains one element for each non-domain NC replica in the forest. These elements describe the initial state of all such non-domain NC replicas and the changes to be performed as part of the NC Rename operation.

RootDomain: Holds a DomainDescription tuple (section 3.1.1.12.1.8) describing the root domain of the forest. This field describes the initial state of the root domain NC replica and the changes to be performed as part of the NC Rename operation.

TrustTreeRootDomains: A set of DomainDescription tuples. This field contains one element for each domain NC replica that is to have no values of the trustParent attribute on the **NCCrossRef** object. These elements describe the initial state of all such domain NC replicas and the changes to be performed as part of the NC Rename operation.

TrustTreeNonRootDomains: A set of DomainWithNewTrustParentDescription tuples. This field contains one element for each domain NC replica that is to have a new value for the trustParent attribute on the **NCCrossRef** object. These elements describe the initial state of all such domain NC replicas and the changes to be performed as part of the NC Rename operation.

AllDomains: Holds a set containing references to the elements in the union of the sets in the **TrustTreeRootDomains** field, the **TrustTreeNonRootDomains** field, and a set containing the value of the **RootDomain** field. This set holds references to both DomainDescription tuples and DomainWithNewTrustParentDescription tuples (section 3.1.1.12.1.10). At a minimum, this set contains one reference to a DomainDescription tuple, which is the DomainDescription in the **RootDomain** field. This field contains one reference to an object that describes each domain NC in the forest.

Note This field contains only references to tuples, not instances of the tuple, in order to formally preserve the uniqueness constraints of various tuple fields. Although it contains only references, it can be used in a production rule exactly as if it contained the instances themselves.

DomainsCount: Holds the number of elements in the **AllDomains** field.

AllNCs: Holds a set containing references to the elements in the union of the sets in the **AppNCs** field, the **TrustTreeRootDomains** field, the **TrustTreeNonRootDomains** field, and a set containing the value of the **RootDomain** field. At a minimum, this set contains a reference to one DomainDescription tuple, which is the DomainDescription in the **RootDomain** field.

Note This field contains only references to tuples, not instances of the tuple, in order to formally preserve the uniqueness constraints of various tuple fields. Although it contains only references, it can be used in a production rule exactly as if it contained the instances themselves.

3.1.1.12.2 Encoding/Decoding Rules

This section defines a notation for encoding and decoding a tuple to and from a string. An expression that describes the specific encoding/decoding for an NCRenameDescription tuple (section 3.1.1.12.1.11) is defined.

3.1.1.12.2.1 EBNF-M

Extended Backus-Naur Form (EBNF) [ISO/IEC-14977] is a notation used for expressing context-free grammars, describing all possible legal statements that match an expression. The syntax used to describe the encoding and decoding of an NCRenameDescription tuple to and from a string is a modified version of EBNF, hereafter called Extended Backus-Naur Form--Modified (EBNF-M). EBNF-M is defined here and is used to express an instance (or set of equivalent instances) of a legal statement based on an instance of a tuple. The elements defined in the following sections have been added to EBNF to produce EBNF-M.

3.1.1.12.2.1.1 Tuples as Parameters to Production Rules

An EBNF-M production rule can be defined such that it has access to one or more instances of tuples. The syntax for this is as follows.

```
productionRule(parameterList) = __expression__
```

Where productionRule and __expression__ are standard EBNF syntax and parameterList is a comma-delimited list of one or more tuple types. These parameters are accessible to __expression__, which can make use of them as described in the following sections.

3.1.1.12.2.1.2 Parameter Fields as Terminal Values

In EBNF, an __expression__ is a substitution rule that is made up of a set of operators and either terminal values or non-terminals. In EBNF-M, parameter fields are terminals. When a parameter field is used as a terminal, the meaning is to use the value of the field as a terminal value. The following is an example of this usage.

Given:

```
type tuple1 = [ field1: string ]
tuple1 Instance1 = [field1= "b"]

productionRule1(tuple1) = "a" , tuple1.field1, "c";
```

Then:

```
productionRule1(Instance1) == "abc"
```

3.1.1.12.2.1.3 Formatting of Non-String Parameter Fields as Terminal Values

Tuple fields are not limited to strings. In the case where a field is not a string, a specification for how to express the value as a string is necessary. The syntax for this is as follows.

```
<type> = Text description of how to format the type as a string
```

The following is an example of this usage.

Given:

```
type tuple2 = [ field1: integer ]
tuple2 Instance2 = [ field1 = 2]
<integer> = A base 10 integer with no leading zeros
productionRule2(tuple2) = "1" , tuple2.field1, "3";
```

Then:

```
productionRule2(Instance2) == "123"
```

3.1.1.12.2.1.4 Parameter Fields as Iterators

In EBNF, the standard way to define that a production rule results in one or more repetitions of another production rule is the following.

```
productionRuleX = productionRuleX | (productionRuleX , productionRuleY)
```

When describing how an instance of a tuple results in a legal expression, it is often necessary to constrain this basic repetition to invoke a production rule once for every element in a set stored as a field in a tuple. EBNF-M uses "foreach", a specialized non-terminal, to describe this. The syntax for this keyword is the following.

```
foreach(typeA in typeB.field) productionRule(typeA)
```

typeB.field must be a set of elements of type typeA. This non-terminal is equivalent to the following.

```
productionRule(typeB.field.elementX),
    productionRule(typeB.field.elementY),
...
    productionRule(typeB.field.elementZ);
```

Where the set in typeB.field comprises all typeB.field.element*. No ordering of element* is implied or required. Since the elements of a set are not ordered, this non-terminal results in more than one legal statement when typeB.field contains more than one element.

The following is an example of this usage.

Given:

```
type tuple3 = [ field1: set of integers ]
tuple3 Instance3 = [field1 = {1, 2}]

<integer> = A base 10 integer with no leading zeros
productionRule3(tuple1) =
    foreach(integer in tuple3.field1) productionRule4(string);
productionRule4(integer) = "<", integer, ">";
```

Then:

```
productionRule3(Instance3) == "<1><2>"
```

Or:

```
productionRule3(Instance3) == "<2><1>"
```

3.1.1.12.2.1.5 Reversed Production Rules

EBNF-M production rules can be reversed. That is, given a production rule with a tuple as a parameter and the result of the production rule, the instance of the tuple that produced the result can be recovered. The syntax for this is as follows.

```
Reversed::productionRule(result) = tuple
```

The following is an example of this usage.

Given:

```
type tuple4 = [ field1: stringVal; field2: integer ]

<stringVal> = A string containing no quotation marks.
<integer> = A base 10 integer with no leading zeros
productionRule5(tuple4) =
    "The string is \"", tuple4.field1, "\"",
    " and the integer is ", tuple4.field2, ".";
```

Then:

```
Reversed::productionRule5("The string is \"a\" and the integer is 1) =
    [ field1 = "a", field2 = 1 ]
```

Note that not all production rules can be deterministically reversed. The following is an example of such a production rule.

Given:

```
type tuple5 = [ field1: string; field2: string ]
productionRule6(tuple5) = tuple5.field1, tuple5.field2;
```

Then:

```
Reversed::productionRule("a,b") = Error
```

The error occurs because any of the following tuples produce the result.

```
[ field1: "a,b"; field2: "" ]
[ field1: "a,"; field2: "b" ]
[ field1: "a"; field2: ",b" ]
[ field1: ""; field2: "a,b" ]
```

Note that not all reversible production rules can be reversed in a context-free manner, although they can still be reversed. The following is an example of such a production rule.

Given:

```
type tuple6 = [ field1: alphabetic caracter;
                field2:alphabetic character;
                field3:alphabetic character]
field1 of all instances of tuple6 is unique across all instances of tuple6

type tuple7 = [ field4: a set of tuple6 ]
productionRule7(tuple7) =
    "[",
    foreach(tuple6 in tuple7.field4) productionRule8(tuple6),
    "]",
    "[",
    foreach(tuple6 in tuple7.field4) productionRule9(tuple6),
    "]";
```

```
productionRule8(tuple6) = "(", tuple6.field1, "," tuple6.field2, ")";
productionRule9(tuple6) = "(", tuple6.field1, "," tuple6.field3, ")";
```

Then:

```
Reversed::productionRule("[(a,b)(d,e)][(d,f),(a,c)") =
    [ field4: {(field1:a, field2:b, field3:c),(field1:d,field2:e,field3:f)]
```

Reversal is possible in this case because the use of tuple6.field1 is unique across all instances of tuple6 and is used in both productionRule8 and productionRule9, allowing field1 to be used as a "key" to combine the results from reversing productionRule8 and productionRule9.

3.1.1.12.2.2 CodedNCRenameDescription

This section defines an EBNF-M expression that is used to encode an NCRenameDescription tuple into a string and to decode strings to NCRenameDescription tuples. The given expression is a reversible EBNF-M expression.

3.1.1.12.2.2.1 Expression

```
CodedNCRenameDescription(NCRenameDescription):=
    ExpressionPrefix,
    Tests(NCRenameDescription),
    Flatten(NCRenameDescription),
    Rebuild(NCRenameDescription),
    CrossRefs(NCRenameDescription),
    Trusts(NCRenameDescription),
    ReplicationEpoch(NCRenameDescription),
    ExpressionSuffix;

ExpressionPrefix =
    ExpressionPrefixFragment01,
    WhiteSpace,
    ExpressionPrefixFragment02,
    WhiteSpace;

ExpressionPrefixFragment01 =
    "<?xml version =\"1.0\"?>";

ExpressionPrefixFragment02 =
    "<NTDSAscript opType=\"renamedomain\">"

ExpressionSuffix:=
    "</NTDSAscript>";
```

3.1.1.12.2.2.2 Common

```
<GUID> = Expressed in the form of a dashed-string UUID defined in ([RFC4122] section 3).
<SecurityIdentifier> = Expressed in the form of a Security Descriptor
    Definition Language (SDDL) SID string. The SID structure and the format
    of SDDL SID strings are defined in [MS-DTYP] sections 2.4.2 and 2.5.1.
<DNSAddress> = Expressed in the form defined in [RFC1035] section 2.3.1.
<DN> = Expressed in the form defined in [RFC2253] section 3.
```

```
<32-bit integer> = Expressed as a base 10 integer with no leading zeros.

ErrorReportNoEnd =
    ErrorMessage,
    Space,
    ReturnValue,
    ErrorReportNoEndFragment01,
    WhiteSpace;

ErrorReportNoEndFragment01 =
    ">";

ErrorReport =
    ErrorMessage,
    Space,
    ReturnValue,
    ErrorReportFragment01,
    WhiteSpace;

ErrorReportFragment01 =
    "/>";

ErrorMessage =
    Quote,
    Message,
    Quote;

Quote =
    "\"";

Message =
    A string composed strictly of spaces and alphanumerics.

ReturnValue =
    ReturnValueFragment01,
    Code;

ReturnValueFragment01 =
    "returnCode=";

Code =
    Quote,
    Number,
    Quote;

Number =
    A 32-bit integer.

Space =
    " ";

Comma =
    ",";

SystemRDN =
    ",CN=System,";

WhiteSpace =
    "" |
```

```
    WhiteSpaceChar |
    (WhiteSpaceChar, WhiteSpace);

WhiteSpaceChar =
    A space, a newline, or a tab.
```

3.1.1.12.2.2.3 Tests

```
Tests(NCRenameDescription) =
    TestsBegin,
    TestConfigurationNC(NCRenameDescription),
    TestReplicationEpoch(NCRenameDescription),
    TestAppNCs(NCRenameDescription),
    TestDomains(NCRenameDescription),
    TestPartitionCounts(NCRenameDescription),
    TestsEnd;

TestsBegin =
    TestsBeginFragment01,
    Message,
    TestsBeginFragment02,
    Message,
    TestsBeginFragment03,
    WhiteSpace;

TestsBeginFragment01 =
    "<action name=\"";

TestsBeginFragment02 =
    "\" stage=\"";

TestsBeginFragment03 =
    "\">";

TestsEnd =
    TestsEndFragment01,
    WhiteSpace;

TestsEndFragment01 =
    "</action>";
```

3.1.1.12.2.2.3.1 TestConfigurationNC

```
TestConfigurationNC(NCRenameDescription) =
    TestConfigurationNCFragment01,
    NCRenameDescription.ConfigurationNCGUID,
    TestConfigurationNCFragment02,
    ErrorReport;

TestConfigurationNCFragment01 =
    "<predicate test=\"instantiated\" instancetype=\"write\" path=\"guid:";

TestConfigurationNCFragment02 =
    "\" type=\"base\" ";
```

3.1.1.12.2.2.3.2 TestReplicationEpoch

```
TestReplicationEpoch(NCRenameDescription) =
    TestReplicationEpochFragment01,
    ErrorReportNoEnd,
    TestReplicationEpochFragment02,
    TestReplicationEpochFragment03,
    NCRenameDescription.NewReplicationEpoch,
    TestReplicationEpochFragment04,
    ErrorReport,
    TestReplicationEpochFragment05,
    WhiteSpace;

TestReplicationEpochFragment01 =
    "<predicate test=\"not\" ";

TestReplicationEpochFragment02 =
    "<predicate test=\"compare\" path=\"$LocalNTDSSettingsObjectDN$\"";

TestReplicationEpochFragment03 =
    " attribute=\"msDS-ReplicationEpoch\" attrval=\"";

TestReplicationEpochFragment04 =
    "\" defaultvalue=\"0\" type=\"base\" ";

TestReplicationEpochFragment05 =
    "</predicate>";
```

3.1.1.12.2.2.3.3 TestAppNCs

```
TestAppNCs(NCRenameDescription) =
    foreach(NCDescription in NCRenameDescription.AppNCs)
        TestAppNCCrossRef(NCDescription);

TestAppNCCrossRef(NCDescription) =
    TestAppNCCrossRefExists(NCDescription),
    TestAppNCCrossRefNCNameUnchanged(NCDescription);

TestAppNCCrossRefExists(NCDescription) =
    TestAppNCCrossRefExistsFragment01,
    NCDescription.CrossRefGuid,
    TestAppNCCrossRefExistsFragment02,
    ErrorReport;

TestAppNCCrossRefExistsFragment01 =
    "<predicate test=\"instantiated\" instancetype=\"write\" path=\"guid:";

TestAppNCCrossRefExistsFragment02 =
    "\" type=\"base\" ";

TestAppNCCrossRefNCNameUnchanged(NCDescription) =
    TestAppNCCrossRefNCNameUnchangedFragment01,
    NCDescription.CrossRefGuid,
    TestAppNCCrossRefNCNameUnchangedFragment02,
    NCDescription.ExistingDN,
    TestAppNCCrossRefNCNameUnchangedFragment03,
    ErrorReport;
```

```
TestAppNCCrossRefNCNameUnchangedFragment01 =
    "<predicate test=\"compare\" path=\"guid:";

TestAppNCCrossRefNCNameUnchangedFragment02 =
    "\" attribute=\"NcName\" attrval=\"";

TestAppNCCrossRefNCNameUnchangedFragment03 = "\" defaultvalue=\"0\" type=\"base\" ";
```

3.1.1.12.2.2.3.4 TestDomains

```
TestDomains(NCRenameDescription) =
    foreach(DomainDescription in NCRenameDescription.AllDomains)
        TestDomainDescription(NCRenameDescription, DomainDescription);

TestDomainDescription(NCRenameDescription, DomainDescription) =
    TestCrossRef(NCRenameDescription, DomainDescription),
    TestServersInstantiated(DomainDescription),
    TestTrustedDomainObjectDescriptions(DomainDescription),
    TestTrustCount(DomainDescription),
    TestInterdomainTrustAccountDescriptions(DomainDescription),
    TestDomainDescriptionFragment01,
    WhiteSpace,
    TestDomainDescriptionFragment02,
    WhiteSpace,
    TestDomainDescriptionFragment03,
    WhiteSpace,
    TestServerDescriptions(DomainDescription);

TestDomainDescriptionFragment01 =
    "</action>";

TestDomainDescriptionFragment02 =
    "</then>";

TestDomainDescriptionFragment03 =
    "</condition>";
```

3.1.1.12.2.2.3.4.1 TestCrossRef

```
TestCrossRef(NCRenameDescription, DomainDescription) =
    TestCrossRefExists(DomainDescription),
    TestCrossRefNCNameUnchanged(DomainDescription),
    TestCrossRefNewDNUnused(NCRenameDescription, DomainDescription);

TestCrossRefExists(DomainDescription) =
    CrossRefExistsFragment01,
    DomainDescription.CrossRefGuid,
    CrossRefExistsFragment02,
    ErrorReport;

CrossRefExistsFragment01 =
    "<predicate test=\"instantiated\" instancetype=\"write\" path=\"guid:";

CrossRefExistsFragment02 =
    "\" type=\"base\" ";
```

```
TestCrossRefNCNameUnchanged(DomainDescription) =
    TestCrossRefNCNameUnchangedFragment01,
    DomainDescription.CrossRefGuid,
    TestCrossRefNCNameUnchangedFragment02,
    DomainDescription.ExistingDN,
    TestCrossRefNCNameUnchangedFragment03,
    ErrorReport;

TestCrossRefNCNameUnchangedFragment01 =
    "<predicate test=\"compare\" path=\"guid:";

TestCrossRefNCNameUnchangedFragment02 =
    "\" attribute=\"NcName\" attrval=\"";

TestCrossRefNCNameUnchangedFragment03 =
    "\" defaultvalue=\"0\" type=\"base\" ";

TestCrossRefNewDNUnused(NCRenameDescription, DomainDescription) =
    TestCrossRefNewDNUnusedFragment01,
    ErrorReportNoEnd,
    TestCrossRefNewDNUnusedFragment02,
    DomainDescription.NewFlatName,
    TestCrossRefNewDNUnusedFragment03,
    NCRenameDescription.RootDomain.ExistingDN,
    TestCrossRefNewDNUnusedFragment04,
    ErrorReport,
    TestCrossRefNewDNUnusedFragment05,
    WhiteSpace;

TestCrossRefNewDNUnusedFragment01 =
    "<predicate test=\"not\" ";

TestCrossRefNewDNUnusedFragment02 =
    "<predicate test=\"instantiated\" instancetype=\"write\" path=\"CN=";

TestCrossRefNewDNUnusedFragment03 =
    ",CN=Partitions,CN=Configuration,";

TestCrossRefNewDNUnusedFragment04 =
    "\" type=\"base\" ";

TestCrossRefNewDNUnusedFragment05 =
    "</predicate>";
```

3.1.1.12.2.2.3.4.2 TestServersInstantiated

```
TestServersInstantiated(DomainDescription) =
    foreach(ServerDescription in DomainDescription.Servers)
        TestServerInstantiated(ServerDescription);

TestServerInstantiated(ServerDescription)
    TestServerInstantiatedFragment01,
    WhiteSpace,
    TestServerInstantiatedFragment02,
    WhiteSpace,
    TestServerInstantiatedFragment03,
    ServerDescription.serverGuid,
```

```
    TestServerInstantiatedFragment04,
    WhiteSpace,
    TestServerInstantiatedFragment05,
    WhiteSpace,
    TestServerInstantiatedFragment06,
    WhiteSpace,
    TestServerInstantiatedFragment07,
    WhiteSpace;

TestServerInstantiatedFragment01 =
    "<condition>";

TestServerInstantiatedFragment02 =
    "<if>";

TestServerInstantiatedFragment03 =
    "<predicate test=\"instantiated\" instancetype=\"write\" path=\"guid:";

TestServerInstantiatedFragment04 =
    "\" type=\"base\"/>";

TestServerInstantiatedFragment05 =
    "</if>";

TestServerInstantiatedFragment06 =
    "<then>";

TestServerInstantiatedFragment07 =
    "<action>";
```

3.1.1.12.2.2.3.4.3 TestTrustCount

```
TestTrustCount(DomainDescription) =
    TestTrustCountFragment01,
    DomainDescription.ExistingDN,
    TestTrustCountFragment02,
    DomainDescription.CountTrusts,
    TestTrustCountFragment03,
    ErrorReport;

TestTrustCountFragment01 =
    "<predicate test=\"cardinality\" type=\"subTree\" path=\"CN=System,";

TestTrustCountFragment02 =
    "\" filter=\"COUNT_TRUSTS_FILTER\" cardinality=\"";

TestTrustCountFragment03 =
    "\" ";
```

3.1.1.12.2.2.3.4.4 TestTrustedDomainObjectDescriptions

```
TestTrustedDomainObjectDescriptions(DomainDescription) =
    foreach(TrustedDomainObjectDescription in
      DomainDescription.TrustedDomainObjects)
        TestTrustedDomainObjectDescription(DomainDescription,
            TrustedDomainObjectDescription);
```

```
TestTrustedDomainObjectDescription(DomainDescription, TrustedDomainObjectDescription) =
    TestTrustedDomainObjectDescriptionFragment01,
    TrustedDomainObjectDescription.Guid,
    TestTrustedDomainObjectDescriptionFragment02,
    ErrorReport,
    TestTrustedDomainObjectDescriptionFragment03,
    TrustedDomainObjectDescription.Guid,
    TestTrustedDomainObjectDescriptionFragment04,
    TrustedDomainObjectDescription.SID,
    TestTrustedDomainObjectDescriptionFragment05,
    ErrorReport,
    TestTrustedDomainObjectDescriptionFragment06,
    ErrorReportNoEnd,
    TestTrustedDomainObjectDescriptionFragment07,
    TrustedDomainObjectDescription.NewTrustPartnerDNSName,
    SystemRDN,
    DomainDescription.ExistingDN,
    TestTrustedDomainObjectDescriptionFragment08,
    ErrorReport,
    TestTrustedDomainObjectDescriptionFragment09,
    WhiteSpace;

TestTrustedDomainObjectDescriptionFragment01 =
    "<predicate test=\"instantiated\" instancetype=\"write\" path=\"guid:";

TestTrustedDomainObjectDescriptionFragment02 =
    "\" type=\"base\" ";

TestTrustedDomainObjectDescriptionFragment03 =
    "<predicate test=\"compare\" path=\"guid:";

TestTrustedDomainObjectDescriptionFragment04 =
    "\" attribute=\"securityIdentifier\" attrval=\"";

TestTrustedDomainObjectDescriptionFragment05 =
    "\" defaultvalue=\"0\" type=\"base\" ";

TestTrustedDomainObjectDescriptionFragment06 =
    "<predicate test=\"not\" ";

TestTrustedDomainObjectDescriptionFragment07 =
    "<predicate test=\"instantiated\" instancetype=\"write\" path=\"CN=";

TestTrustedDomainObjectDescriptionFragment08 =
    "\" type=\"base\" ";

TestTrustedDomainObjectDescriptionFragment09 =
    "</predicate>";
```

3.1.1.12.2.2.3.4.5 TestInterdomainTrustAccountDescriptions

```
TestInterdomainTrustAccountDescriptions(DomainDescription) =
    foreach(InterdomainTrustAccountDescription in
      DomainDescription.InterdomainTrustAccounts)
        TestInterdomainTrustAccountDescription(DomainDescription,
            InterdomainTrustAccountDescription);
```

```
TestInterdomainTrustAccountDescription(DomainDescription,
  InterdomainTrustAccountDescription) =
    TestInterdomainTrustAccountDescriptionFragment01,
    InterdomainTrustAccountDescription.Guid,
    TestInterdomainTrustAccountDescriptionFragment02,
    ErrorReport,
    TestInterdomainTrustAccountDescriptionFragment03,
    InterdomainTrustAccountDescription.Guid,
    TestInterdomainTrustAccountDescriptionFragment04,
    InterdomainTrustAccountDescription.ExistingFlatName,
    TestInterdomainTrustAccountDescriptionFragment05,
    ErrorReport,
    TestInterdomainTrustAccountDescriptionFragment06,
    ErrorReportNoEnd,
    TestInterdomainTrustAccountDescriptionFragment07,
    InterdomainTrustAccountDescription.NewFlatName,
    Comma,
    InterdomainTrustAccountDescription.ParentDNFromDomainDN,
    Comma,
    DomainDescription.NewDN,
    TestInterdomainTrustAccountDescriptionFragment08,
    ErrorReport,
    TestInterdomainTrustAccountDescriptionFragment09,
    WhiteSpace;

TestInterdomainTrustAccountDescriptionFragment01 =
    "<predicate test=\"instantiated\" instancetype=\"write\" path=\"guid:";

TestInterdomainTrustAccountDescriptionFragment02 =
    "\" type=\"base\" ";

TestInterdomainTrustAccountDescriptionFragment03 =
    "<predicate test=\"compare\" path=\"guid:";

TestInterdomainTrustAccountDescriptionFragment04 =
    "\" attribute=\"samAccountName\" attrval=\"";

TestInterdomainTrustAccountDescriptionFragment05 =
    "\" defaultvalue=\"0\" type=\"base\" ";

TestInterdomainTrustAccountDescriptionFragment06 =
    "<predicate test=\"not\" ";

TestInterdomainTrustAccountDescriptionFragment07 =
    "<predicate test=\"instantiated\" instancetype=\"write\" path=\"CN=";

TestInterdomainTrustAccountDescriptionFragment08 =
    "\" type=\"base\" ";

TestInterdomainTrustAccountDescriptionFragment09 =
    "</predicate>";
```

3.1.1.12.2.2.3.4.6 TestServerDescriptions

```
TestServerDescriptions(DomainDescription) =
    foreach (ServerDescription in DomainDescription.Servers)
```

```
        TestServerSPNs(ServerDescription)

  TestServerSPNs(ServerDescription) =
      TestServerSPNsFragment01,
      WhiteSpace,
      TestServerSPNsFragment02,
      WhiteSpace,
      TestServerSPNsFragment03,
      ServerDescription.serverGuid,
      TestServerSPNsFragment04,
      WhiteSpace,
      TestServerSPNsFragment05,
      WhiteSpace,
      TestServerSPNsFragment06,
      WhiteSpace,
      TestServerSPNsFragment07,
      WhiteSpace,
      TestSPNs(ServerDescription),
      TestServerSPNsFragment08,
      WhiteSpace,
      TestServerSPNsFragment09,
      WhiteSpace,
      TestServerSPNsFragment10,
      WhiteSpace;

  TestServerSPNsFragment01 =
      "<condition>";

  TestServerSPNsFragment02 =
      "<if>";

  TestServerSPNsFragment03 =
      "<predicate test=\"instantiated\" instancetype=\"read\" path=\"guid:";

  TestServerSPNsFragment04 =
      "\" type=\"base\"/>";

  TestServerSPNsFragment05 =
      "</if>";

  TestServerSPNsFragment06 =
      "<then>";

  TestServerSPNsFragment07 =
      "<action>";

  TestServerSPNsFragment08 =
      "</action>";

  TestServerSPNsFragment09 =
      "</then>";

  TestServerSPNsFragment10 =
      "</condition>";


  TestSPNs(ServerDescription) =
      foreach(SPNValue in ServerDescription.SPNs)
          TestSPN(SPNValue, ServerDescription);
```

```
TestSPN(SPNValue, ServerDescription) =
    TestSPNFragment01,
    ServerDescription.ExistingDN,
    TestSPNFragment02,
    SPNValue,
    TestSPNFragment03,
    ErrorReport;

TestSPNFragment01 =
    "<predicate test=\"compare\" path=\"";

TestSPNFragment02 =
    "\" attribute=\"servicePrincipalName\" attrval=\"";

TestSPNFragment03 =
    "\" defaultvalue=\"0\" type=\"base\" ";
```

3.1.1.12.2.2.3.5 TestPartitionCounts

```
TestPartitionCounts(NCRenameDescription) =
    TestPartitionCountsFragment01,
    NCRenameDescription.RootDomain.ExistingDN,
    TestPartitionCountsFragment02,
    NCRenameDescription.DomainsCount,
    TestPartitionCountsFragment03,
    ErrorReport;

TestPartitionCountsFragment01 =
    "<predicate test=\"cardinality\" type=\"subTree\" path=\"CN=Partitons,CN=Configuration,";

TestPartitionCountsFragment02 =
    "\" filter=\"COUNT_DOMAINS_FILTER\" cardinality=\"";

TestPartitionCountsFragment03 =
    "\" ";
```

3.1.1.12.2.2.4 Flatten

```
Flatten(NCRenameDescription) =
    FlattenFragment01,
    Message,
    FlattenFragment02,
    WhiteSpace,
    FlattenNCs(NCRenameDescription),
    FlattenFragment03,
    WhiteSpace;

FlattenFragment01 =
    "<action name=\"";

FlattenFragment02 =
    "\">";

FlattenFragment03 =
```

```
    "</action>";


FlattenNCs(NCRenameDescription) =
    foreach(NCDescription in NCRenameDescription.AllNCs)
        FlattenNC(NCDescription);

FlattenNC(NCDescription) =
    FlattenNCFragment01,
    NCDescription.ExistingDN,
    FlattenNCFragment02,
    WhiteSpace,
    FlattenNCFragment03,
    NCDescription.Guid,
    FlattenNCFragment04,
    WhiteSpace,
    FlattenNCFragment05,
    WhiteSpace;

FlattenNCFragment01 =
    "<move path=\"dn:";

FlattenNCFragment02 =
    "\" metadata=\"0\">";

FlattenNCFragment03 =
    "<to path=\"dn:DC=";

FlattenNCFragment04 =
    ",DC=INVALID\"/>";

FlattenNCFragment05 =
    "</move>";
```

3.1.1.12.2.2.5 Rebuild

```
Rebuild(NCRenameDescription) =
    RebuildFragment01,
    Message,
    RebuildFragment02,
    WhiteSpace,
    RebuildNCs(NCRenameDescription),
    RebuildFragment03,
    WhiteSpace;

RebuildFragment01 =
    "<action name=\"";

RebuildFragment02 =
    "\">";

RebuildFragment03 =
    "</action>";


RebuildNCs(NCRenameDescription) =
    foreach(NCDescription in NCRenameDescription.AllNCs)
```

```
        RebuildNC(NCDescription);

RebuildNC(NCDescription) =
    RebuildNCFragment01,
    NCDescription.Guid,
    RebuildNCFragment02,
    WhiteSpace,
    RebuildNCFragment03,
    NCDescription.NewDN,
    RebuildNCFragment04,
    WhiteSpace,
    RebuildNCFragment05,
    WhiteSpace;

RebuildNCFragment01 =
    "<move path=\"dn:DC=";

RebuildNCFragment02 =
    ",DC=INVALID\" metadata=\"0\">";

RebuildNCFragment03 =
    "<to path=\"dn:";

RebuildNCFragment04 =
    "\"/>";

RebuildNCFragment05 =
    "</move>";
```

3.1.1.12.2.2.6 Trusts

```
Trusts(NCRenameDescription) =
    TrustsFragment01,
    Message,
    TrustsFragment02,
    WhiteSpace,
    DomainsTrusts(NCRenameDescription),
    TrustsFragment03,
    WhiteSpace;

TrustsFragment01 =
    "<action name=\"";

TrustsFragment02 =
    "\">";

TrustsFragment03 =
    "</action>";

DomainsTrusts(NCRenameDescription) =
    foreach(DomainDescription in NCRenameDescription.AllDomains)
        DomainTrust(DomainDescription);

DomainTrust(DomainDescription)
    DomainTrustFragment01,
    WhiteSpace,
    DomainTrustFragment02,
```

```
    WhiteSpace,
    DomainTrustFragment03,
    DomainDescription.Guid,
    DomainTrustFragment04,
    WhiteSpace,
    DomainTrustFragment05,
    WhiteSpace,
    DomainTrustFragment06,
    WhiteSpace,
    DomainTrustFragment07,
    WhiteSpace,
    DomainTrustSpecifications(DomainDescription),
    DomainTrustAccounts(DomainDescription),
    DomainTrustFragment08,
    WhiteSpace,
    DomainTrustFragment09,
    WhiteSpace,
    DomainTrustFragment10,
    WhiteSpace;

DomainTrustFragment01 =
    "<condition>";

DomainTrustFragment02 =
    "<if>";

DomainTrustFragment03 =
    "<predicate test=\"instantiated\" instancetype=\"write\" path=\"guid:";

DomainTrustFragment04 =
    "\" type=\"base\"/>";

DomainTrustFragment05 =
    "</if>";

DomainTrustFragment06 =
    "<then>";

DomainTrustFragment07 =
    "<action>";

DomainTrustFragment08 =
    "</action>";

DomainTrustFragment09 =
    "</then>";

DomainTrustFragment10 =
    "</condition>";
```

3.1.1.12.2.2.6.1 DomainTrustSpecifications

```
DomainTrustSpecifications(DomainDescription) =
    foreach(TrustedDomainObject in DomainDescription.TrustedDomainObjects)
        DomainTrustSpecification(DomainDescription, TrustedDomainObject);

DomainTrustSpecification(DomainDescription, TrustedDomainObject) =
```

```
      DomainTrustSpecificationFragment01,
      TrustedDomainObject.ExistingTrustPartnerDNSName,
      SystemRDN,
      DomainDescription.NewDN,
      DomainTrustSpecificationFragment02,
      WhiteSpace,
      DomainTrustSpecificationFragment03,
      TrustedDomainObject.NewTrustPartnerFlatName,
      DomainTrustSpecificationFragment04,
      WhiteSpace,
      DomainTrustSpecificationFragment05,
      TrustedDomainObject.NewTrustPartnerDNSName,
      DomainTrustSpecificationFragment06,
      WhiteSpace,
      DomainTrustSpecificationFragment07,
      WhiteSpace,
      DomainTrustSpecificationFragment08,
      TrustedDomainObject.ExistingTrustPartnerDNSName,
      SystemRDN,
      DomainDescription.NewDN,
      DomainTrustSpecificationFragment09,
      WhiteSpace
      DomainTrustSpecificationFragment10,
      TrustedDomainObject.NewTrustPartnerDNSName,
      SystemRDN,
      DomainDescription.NewDN,
      DomainTrustSpecificationFragment11,
      WhiteSpace,
      DomainTrustSpecificationFragment12,
      WhiteSpace;

  DomainTrustSpecificationFragment01 =
      "<update path=\"dn:CN=";

  DomainTrustSpecificationFragment02 =
      "\" metadata=\"1\">";

  DomainTrustSpecificationFragment03 =
      "<flatName op=\"replace\">";

  DomainTrustSpecificationFragment04 =
      "</flatName>";

  DomainTrustSpecificationFragment05 =
      "<trustPartner op=\"replace\">";

  DomainTrustSpecificationFragment06 =
      "</trustPartner>";

  DomainTrustSpecificationFragment07 =
      "</update>";

  DomainTrustSpecificationFragment08 =
      "<move path=\"dn:CN=";

  DomainTrustSpecificationFragment09 =
      "\" metadata=\"1\">";

  DomainTrustSpecificationFragment10 =
```

```
    "<to path=\"dn:CN=";

DomainTrustSpecificationFragment11 =
    "\"/>";

DomainTrustSpecificationFragment12 =
    "</move>";
```

3.1.1.12.2.2.6.2 DomainTrustAccounts

```
DomainTrustAccounts(DomainDescription) =
    foreach (InterdomainTrustAccountDescription in
      DomainDescription.InterdomainTrustAccounts)
        InterdomainTrustAccount(DomainDescription,
            InterdomainTrustAccountDescription);

InterdomainTrustAccount(DomainDescription,
  InterdomainTrustAccountDescription) =
    InterdomainTrustAccountFragment01,
    InterdomainTrustAccountDescription.ExistingFlatName,
    Comma,
    InterdomainTrustAccountDescription.ParentDNFromDomainDN,
    Comma,
    DomainDescription.NewDN,
    InterdomainTrustAccountFragment02,
    WhiteSpace,
    InterdomainTrustAccountFragment03,
    InterdomainTrustAccountDescription.NewFlatName,
    InterdomainTrustAccountFragment04,
    WhiteSpace,
    InterdomainTrustAccountFragment05,
    WhiteSpace,
    InterdomainTrustAccountFragment06,
    InterdomainTrustAccountDescription.ExistingFlatName,
    Comma,
    InterdomainTrustAccountDescription.ParentDNFromDomainDN,
    Comma,
    DomainDescription.NewDN,
    InterdomainTrustAccountFragment07,
    WhiteSpace,
    InterdomainTrustAccountFragment08,
    InterdomainTrustAccountDescription.NewFlatName,
    Comma,
    InterdomainTrustAccountDescription.ParentDNFromDomainDN,
    Comma,
    DomainDescription.NewDN,
    InterdomainTrustAccountFragment09,
    WhiteSpace,
    InterdomainTrustAccountFragment10,
    WhiteSpace;

InterdomainTrustAccountFragment01 =
    "<update path=\"dn:CN=";

InterdomainTrustAccountFragment02 =
    "\" metadata=\"1\">";
```

```
InterdomainTrustAccountFragment03 =
    "<samAccountName op=\"replace\">";

InterdomainTrustAccountFragment04 =
    "</samAccountName>";

InterdomainTrustAccountFragment05 =
    "</update>";

InterdomainTrustAccountFragment06 =
    "<move path=\"dn:CN=";

InterdomainTrustAccountFragment07 =
    "\" metadata=\"1\">";

InterdomainTrustAccountFragment08 =
    "<to path=\"dn:CN=";

InterdomainTrustAccountFragment09 =
    "\"/>";

InterdomainTrustAccountFragment10 =
    "</move>";
```

3.1.1.12.2.2.7 CrossRefs

```
CrossRefs(NCRenameDescription) =
    CrossRefsFragment01,
    Message,
    CrossRefsFragment02,
    WhiteSpace,
    ConfigurationCrossRef(NCRenameDescription),
    SchemaCrossRef(NCRenameDescription),
    NCRenameDescriptionRootCrossRef(NCRenameDescription),
    TrustTreeRootDomainCrossRefs(NCRenameDescription),
    TrustTreeNonRootDomainCrossRefs(NCRenameDescription),
    AppNCsCrossRefs(NCRenameDescription),
    CrossRefsFragment03,
    WhiteSpace;

CrossRefsFragment01 =
    "<action name=\"";

CrossRefsFragment02 =
    "\">";

CrossRefsFragment03 =
    "</action>";
```

3.1.1.12.2.2.7.1 ConfigurationCrossRef

```
ConfigurationCrossRef(NCRenameDescription) =
    ConfigurationCrossRefFragment01,
    NCRenameDescription.RootDomain.NewDN,
    ConfigurationCrossRefFragment02,
    WhiteSpace,
```

```
    ConfigurationCrossRefFragment03,
    NCRenameDescription.RootDomain.NewDNSName,
    ConfigurationCrossRefFragment04,
    WhiteSpace,
    ConfigurationCrossRefFragment05,
    WhiteSpace;

ConfigurationCrossRefFragment01 =
    "<update path=\"dn:CN=Enterprise Configuration,CN=Partitions,CN=Configuration,";

ConfigurationCrossRefFragment02 =
    "\" metadata=\"1\">";

ConfigurationCrossRefFragment03 =
    "<DnsRoot op=\"replace\">";

ConfigurationCrossRefFragment04 =
    "</DnsRoot>";

ConfigurationCrossRefFragment05 =
    "</update>";
```

3.1.1.12.2.2.7.2 SchemaCrossRef

```
SchemaCrossRef(NCRenameDescription) =
    SchemaCrossRefFragment01,
    NCRenameDescription.RootDomain.NewDN,
    SchemaCrossRefFragment02,
    WhiteSpace,
    SchemaCrossRefFragment03,
    NCRenameDescription.RootDomain.NewDNSName,
    SchemaCrossRefFragment04,
    WhiteSpace,
    SchemaCrossRefFragment05,
    WhiteSpace;

SchemaCrossRefFragment01 =
    "<update path=\"dn:CN=Enterprise Schema,CN=Partitions,CN=Configuration,";

SchemaCrossRefFragment02 =
    "\" metadata=\"1\">";

SchemaCrossRefFragment03 =
    "<DnsRoot op=\"replace\">";

SchemaCrossRefFragment04 =
    "</DnsRoot>";

SchemaCrossRefFragment05 =
    "</update>";
```

3.1.1.12.2.2.7.3 AppNCsCrossRefs

```
AppNCsCrossRefs(NCRenameDescription) =
    foreach(NCDescription in NCRenameDescription.AppNCs)
        AppNCCrossRef(NCRenameDescription, NCDescription);
```

```
AppNCCrossRef(NCRenameDescription, NCDescription) =
    AppNCCrossRefFragment01,
    NCDescription.ExistingFlatname,
    AppNCCrossRefFragment02,
    NCRenameDescription.RootDomain.NewDN,
    AppNCCrossRefFragment03,
    WhiteSpace,
    AppNCCrossRefFragment04,
    NCDescription.NewDNSName,
    AppNCCrossRefFragment05,
    WhiteSpace,
    AppNCCrossRefFragment06,
    WhiteSpace;

AppNCCrossRefFragment01 =
    "<update path=\"dn:CN=";

AppNCCrossRefFragment02 =
    ",CN=Partitions,CN=Configuration,";

AppNCCrossRefFragment03 =
    "\" metadata=\"0\">";

AppNCCrossRefFragment04 =
    "<DnsRoot op=\"replace\">";

AppNCCrossRefFragment05 =
    "</DnsRoot>";

AppNCCrossRefFragment06 =
    "</update>";
```

3.1.1.12.2.2.7.4 NCRenameDescriptionRootCrossRef

```
NCRenameDescriptionRootCrossRef(NCRenameDescription) =
    NCRenameDescriptionRootCrossRefFragment01,
    NCRenameDescription.RootDomain.ExistingFlatName,
    NCRenameDescriptionRootCrossRefFragment02,
    NCRenameDescription.RootDomain.NewDN,
    NCRenameDescriptionRootCrossRefFragment03,
    WhiteSpace,
    NCRenameDescriptionRootCrossRefFragment04,
    NCRenameDescription.RootDomain.NewDNSName,
    NCRenameDescriptionRootCrossRefFragment05,
    WhiteSpace,
    NCRenameDescriptionRootCrossRefFragment06,
    NCRenameDescription.RootDomain.ExistingDNSName,
    NCRenameDescriptionRootCrossRefFragment07,
    WhiteSpace,
    NCRenameDescriptionRootCrossRefFragment08,
    NCRenameDescription.RootDomain.NewFlatName,
    NCRenameDescriptionRootCrossRefFragment09,
    WhiteSpace,
    NCRenameDescriptionRootCrossRefFragment10,
    WhiteSpace,
    NCRenameDescriptionRootCrossRefFragment11,
```

```
    NCRenameDescription.RootDomain.ExistingFlatName,
    NCRenameDescriptionRootCrossRefFragment12,
    NCRenameDescription.RootDomain.NewDN,
    NCRenameDescriptionRootCrossRefFragment13,
    WhiteSpace,
    NCRenameDescriptionRootCrossRefFragment14,
    NCRenameDescription.RootDomain.NewFlatName,
    NCRenameDescriptionRootCrossRefFragment15,
    NCRenameDescription.RootDomain.NewDN,
    NCRenameDescriptionRootCrossRefFragment16,
    WhiteSpace,
    NCRenameDescriptionRootCrossRefFragment17,
    WhiteSpace;

NCRenameDescriptionRootCrossRefFragment01 =
    "<update path=\"dn:CN=";

NCRenameDescriptionRootCrossRefFragment02 =
    ",CN=Partitions,CN=Configuration,";

NCRenameDescriptionRootCrossRefFragment03 =
    "\" metadata=\"0\">";

NCRenameDescriptionRootCrossRefFragment04 =
    "<DnsRoot op=\"replace\">";

NCRenameDescriptionRootCrossRefFragment05 =
    "</DnsRoot>";

NCRenameDescriptionRootCrossRefFragment06 =
    "<msDS-DnsRootAlias op=\"replace\">";

NCRenameDescriptionRootCrossRefFragment07 =
    "</msDS-DnsRootAlias>";

NCRenameDescriptionRootCrossRefFragment08 =
    "<NetBiosName op=\"replace\">";

NCRenameDescriptionRootCrossRefFragment09 =
    "</NetBiosName>";

NCRenameDescriptionRootCrossRefFragment10 =
    "</update>";

NCRenameDescriptionRootCrossRefFragment11 =
    "<move path=\"dn:CN=";

NCRenameDescriptionRootCrossRefFragment12 =
    ",CN=Partitions,CN=Configuration,";

NCRenameDescriptionRootCrossRefFragment13 =
    "\" metadata=\"0\">";

NCRenameDescriptionRootCrossRefFragment14 =
    "<to path=\"dn:CN=";

NCRenameDescriptionRootCrossRefFragment15 =
    ",CN=Partitions,CN=Configuration,";
```

```
NCRenameDescriptionRootCrossRefFragment16 =
    "\"/>";

NCRenameDescriptionRootCrossRefFragment17 =
    "</move>";
```

3.1.1.12.2.2.7.5 TrustTreeNonRootDomainCrossRefs

```
TrustTreeNonRootDomainCrossRefs(NCRenameDescription) =
    foreach(DomainWithNewTrustParentDescription in
      NCRenameDescription.TrustTreeNonRootDomains)
        TrustTreeNonRootDomainCrossRef(NCRenameDescription,
          DomainWithNewTrustParentDescription);

TrustTreeNonRootDomainCrossRef(NCRenameDescription,
  DomainWIthNewTrustParentDescription) =
    TrustTreeNonRootDomainCrossRefFragment01,
    DomainWithNewTrustParentDescription.ExistingFlatName,
    TrustTreeNonRootDomainCrossRefFragment02,
    NCRenameDescription.RootDomain.NewDN,
    TrustTreeNonRootDomainCrossRefFragment03,
    WhiteSpace,
    TrustTreeNonRootDomainCrossRefFragment04,
    DomainWithNewTrustParentDescription.NewDNSName,
    TrustTreeNonRootDomainCrossRefFragment05,
    WhiteSpace,
    TrustTreeNonRootDomainCrossRefFragment06,
    DomainWithNewTrustParentDescription.NewFlatName,
    TrustTreeNonRootDomainCrossRefFragment07,
    WhiteSpace,
    TrustTreeNonRootDomainCrossRefFragment08,
    DomainWithNewTrustParentDescription.NewTrustParentFlatName,
    TrustTreeNonRootDomainCrossRefFragment09,
    NCRenameDescription.RootDomain.NewDN,
    TrustTreeNonRootDomainCrossRefFragment10,
    WhiteSpace,
    TrustTreeNonRootDomainCrossRefFragment11,
    WhiteSpace,
    TrustTreeNonRootDomainCrossRefFragment12,
    DomainWithNewTrustParentDescription.ExistingDNSName,
    TrustTreeNonRootDomainCrossRefFragment13,
    WhiteSpace,
    TrustTreeNonRootDomainCrossRefFragment14,
    WhiteSpace,
    TrustTreeNonRootDomainCrossRefFragment15,
    DomainWithNewTrustParentDescription.ExistingFlatName,
    TrustTreeNonRootDomainCrossRefFragment02,
    NCRenameDescription.RootDomain.NewDN,
    TrustTreeNonRootDomainCrossRefFragment16,
    WhiteSpace,
    TrustTreeNonRootDomainCrossRefFragment17,
    DomainWithNewTrustParentDescription.NewFlatName,
    TrustTreeNonRootDomainCrossRefFragment02,
    NCRenameDescription.RootDomain.NewDN,
    TrustTreeNonRootDomainCrossRefFragment18,
    WhiteSpace,
    TrustTreeNonRootDomainCrossRefFragment19,
```

```
    WhiteSpace;

  TrustTreeNonRootDomainCrossRefFragment01 =
    "<update path=\"dn:CN=";

  TrustTreeNonRootDomainCrossRefFragment02 =
    ",CN=Partitions,CN=Configuration,";

  TrustTreeNonRootDomainCrossRefFragment03 =
    "\" metadata=\"1\">";

  TrustTreeNonRootDomainCrossRefFragment04 =
    "<DnsRoot op=\"replace\">";

  TrustTreeNonRootDomainCrossRefFragment05 =
    "</DnsRoot>";

  TrustTreeNonRootDomainCrossRefFragment06 =
    "<NetBiosName op=\"replace\">";

  TrustTreeNonRootDomainCrossRefFragment07 =
    "</NetBiosName>";

  TrustTreeNonRootDomainCrossRefFragment08 =
    "<TrustParent op=\"replace\">CN=";

  TrustTreeNonRootDomainCrossRefFragment09 =
    ",CN=Partitions,CN=Configuration,";

  TrustTreeNonRootDomainCrossRefFragment10 =
    "</TrustParent>";

  TrustTreeNonRootDomainCrossRefFragment11 =
    "<RootTrust op=\"delete\"></RootTrust>";

  TrustTreeNonRootDomainCrossRefFragment12 =
    "<msDS-DnsRootAlias op=\"replace\">";

  TrustTreeNonRootDomainCrossRefFragment13 =
    "</msDS-DnsRootAlias>";

  TrustTreeNonRootDomainCrossRefFragment14 =
    "</update>";

  TrustTreeNonRootDomainCrossRefFragment15 =
    "<move path=\"dn:CN=";

  TrustTreeNonRootDomainCrossRefFragment16 =
    "\" metadata=\"0\">";

  TrustTreeNonRootDomainCrossRefFragment17 =
    "<to path=\"dn:CN=";

  TrustTreeNonRootDomainCrossRefFragment18 =
    "\"/>";

  TrustTreeNonRootDomainCrossRefFragment19 =
    "</move>";
```

3.1.1.12.2.2.7.6 TrustTreeRootDomainCrossRefs

```
TrustTreeRootDomainCrossRefs(NCRenameDescription) =
    foreach (TrustTreeRootDomainDescription in
      NCRenameDescription.TrustTreeRootDomains)
        TrustTreeRootDomainCrossRef(NCRenameDescription,
          TrustTreeRootDomainDescription);

TrustTreeRootDomainCrossRef(NCRenameDescription, TrustTreeRootDomainDescription) =
    TrustTreeRootDomainCrossRefFragment01,
    TrustTreeRootDomainDescription.ExistingFlatName,
    TrustTreeRootDomainCrossRefFragment02,
    NCRenameDescription.RootDomain.NewDN,
    TrustTreeRootDomainCrossRefFragment03,
    WhiteSpace,
    TrustTreeRootDomainCrossRefFragment04,
    TrustTreeRootDomainDescription.NewDNSName,
    TrustTreeRootDomainCrossRefFragment05,
    WhiteSpace,
    TrustTreeRootDomainCrossRefFragment06,
    TrustTreeRootDomainDescription.NewFlatName,
    TrustTreeRootDomainCrossRefFragment07,
    WhiteSpace,
    TrustTreeRootDomainCrossRefFragment08,
    WhiteSpace,
    TrustTreeRootDomainCrossRefFragment09,
    NCRenameDescription.RootDomain.NewFlatName,
    TrustTreeRootDomainCrossRefFragment10,
    NCRenameDescription.RootDomain.NewDN,
    TrustTreeRootDomainCrossRefFragment11,
    WhiteSpace,
    TrustTreeRootDomainCrossRefFragment12,
    TrustTreeRootDomainDescription.ExistingDNSName,
    TrustTreeRootDomainCrossRefFragment13,
    WhiteSpace,
    TrustTreeRootDomainCrossRefFragment14,
    WhiteSpace,
    TrustTreeRootDomainCrossRefFragment15,
    TrustTreeRootDomainDescription.ExistingFlatName,
    TrustTreeRootDomainCrossRefFragment16,
    NCRenameDescription.RootDomain.NewDN,
    TrustTreeRootDomainCrossRefFragment17,
    WhiteSpace,
    TrustTreeRootDomainCrossRefFragment18,
    TrustTreeRootDomainDescription.NewFlatName,
    TrustTreeRootDomainCrossRefFragment19,
    NCRenameDescription.RootDomain.NewDN,
    TrustTreeRootDomainCrossRefFragment20,
    WhiteSpace,
    TrustTreeRootDomainCrossRefFragment21,
    WhiteSpace;

TrustTreeRootDomainCrossRefFragment01 =
    "<update path=\"dn:CN=";

TrustTreeRootDomainCrossRefFragment02 =
    ",CN=Partitions,CN=Configuration,";

TrustTreeRootDomainCrossRefFragment03 =
```

```
    "\" metadata=\"1\">";

TrustTreeRootDomainCrossRefFragment04 =
    "<DnsRoot op=\"replace\">";

TrustTreeRootDomainCrossRefFragment05 =
    "</DnsRoot>";

TrustTreeRootDomainCrossRefFragment06 =
    "<NetBiosName op=\"replace\">";

TrustTreeRootDomainCrossRefFragment07 =
    "</NetBiosName>";

TrustTreeRootDomainCrossRefFragment08 =
    "<TrustParent op=\"delete\"></TrustParent>";

TrustTreeRootDomainCrossRefFragment09 =
    "<RootTrust op=\"replace\">CN=";

TrustTreeRootDomainCrossRefFragment10 =
    ",CN=Partitions,CN=Configuration,";

TrustTreeRootDomainCrossRefFragment11 =
    "</RootTrust>";

TrustTreeRootDomainCrossRefFragment12 =
    "<msDS-DnsRootAlias op=\"replace\">";

TrustTreeRootDomainCrossRefFragment13 =
    "</msDS-DnsRootAlias>";

TrustTreeRootDomainCrossRefFragment14 =
    "</update>";

TrustTreeRootDomainCrossRefFragment15 =
    "<move path=\"dn:CN=";

TrustTreeRootDomainCrossRefFragment16 =
    ",CN=Partitions,CN=Configuration,";

TrustTreeRootDomainCrossRefFragment17 =
    "\" metadata=\"0\">";

TrustTreeRootDomainCrossRefFragment18 =
    "<to path=\"dn:CN=";

TrustTreeRootDomainCrossRefFragment19 =
    ",CN=Partitions,CN=Configuration,";

TrustTreeRootDomainCrossRefFragment20 =
    "\"/>";

TrustTreeRootDomainCrossRefFragment21 =
    "</move>";
```

3.1.1.12.2.2.8 ReplicationEpoch

```
ReplicationEpoch(NCRenameDescription) =
    ReplicationEpochFragment01,
    Message,
    ReplicationEpochFragment02,
    WhiteSpace,
    ReplicationEpochFragment03,
    WhiteSpace,
    ReplicationEpochFragment04,
    NCRenameDescription.NewReplicationEpoch,
    ReplicationEpochFragment05,
    WhiteSpace,
    ReplicationEpochFragment06,
    WhiteSpace,
    ReplicationEpochFragment07,
    WhiteSpace;

ReplicationEpochFragment01 =
    "<action name=\"";

ReplicationEpochFragment02 =
    "\">";

ReplicationEpochFragment03 =
    "<update path=\"$LocalNTDSSettingsObjectDN$\" metadata=\"0\">";

ReplicationEpochFragment04 =
    "<msDS-ReplicationEpoch op=\"replace\">";

ReplicationEpochFragment05 =
    "</msDS-ReplicationEpoch>";

ReplicationEpochFragment06 =
    "</update>";

ReplicationEpochFragment07 =
    "</action>";
```

3.1.1.12.3 Decode Operation

To process an NC Rename operation, an instance of the NCRenameDescription tuple (section 3.1.1.12.1.11) describing the operation is required and is provided by the invoker of the NC Rename operation. The following EBNF-M operation is performed on the value.

```
Reversed::CodedNCRenameDescription(value) = NR
```

If the reverse operation returns an error (that is, the reversal does not result in a single instance of an NCRenameDescription (see section 3.1.1.12.2.1.5)), this protocol does not restrict what changes occur in the abstract data of the NC performing the NC Rename operation, nor what the return value from the operation is. Such changes can be nondeterministic, and no expectation can be made by the user of the NC Rename operation as to what the result of an operation using a malformed value will be. In order to improve the usability of this operation, it is suggested to implementers that an error be returned in this case.

3.1.1.12.4 Verify Conditions

Before an NC Rename operation is performed, the following conditions must be true for the abstract data of the DC performing the rename and the NCRenameDescription tuple (section 3.1.1.12.1.11) describing the operation, hereafter called *NR*.

- NR.ConfigurationNCGuid is the GUID of a writable object in an NC replica hosted on this DC.
- The value of the msDS-ReplicationEpoch attribute on the DC's NTDS Settings object (section 6.1.1.2.2.1.2.1.1) does not equal NR.NewReplicationEpoch.
- The number of crossRef objects that refer to domain NCs in the Partitions container (that is, the count of domain crossrefs) equals NR.DomainsCount.
- For every NCDescription *AppNC* in NR.AppNCs:
  - AppNC.CrossRefGuid is the GUID of a writable object in an NC replica hosted on this DC.
  - The DN of the object whose GUID is AppNC.CrossRefGuid equals AppNC.ExistingDN.
- For every DomainDescription *Domain* in the union of NR.AllDomains:
  - Domain.CrossRefGuid is the GUID of a writable object in an NC replica hosted on this DC.
  - The value of the nCName attribute on the object whose GUID is Domain.CrossRefGuid equals Domain.ExistingDN.
  - There does not exist an object in an NC replica hosted on this DC whose DN is "CN=*Domain.NewFlatName*,CN=Partitions,CN=Configuration,*NR.RootDomain.ExistingDN*".
  - For every ServerDescription *Server* in Domain.Servers:
    - Server.serverGuid is the GUID of a writable object in an NC replica hosted on this DC.
    - Every value in Server.SPNs exists as a value of the servicePrincipalName attribute on the object whose DN is Server.ExistingDN.
  - For every TrustedDomainObjectDescription *TrustedDomainObject* in Domain.TrustedDomainObjects:
    - TrustedDomainObject.Guid refers to a writable object in an NC replica hosted on this DC.
    - The value of the securityIdentifier attribute on the object whose GUID is TrustedDomainObject.Guid equals TrustedDomainObject.SID.
    - There does not exist an object whose DN is "CN=*TrustedDomainObject.NewTrustPartnerDNSName*".
  - For every InterdomainTrustAccountDescription *InterdomainTrustAccount* in Domain.InterdomainTrustAccounts:
    - InterdomainTrustAccount.Guid refers to a writable object in an NC replica hosted on this DC.
    - The value of the sAMAccountName attribute on the object whose GUID is InterdomainTrustAccount.Guid equals InterdomainTrustAccount.ExistingFlatName.

- There does not exist an object whose DN is "CN=*InterdomainTrustAccount.NewFlatName*,*InterdomainTrustAccount.ParentDNFromDomainDN*,*Domain.NewDN*".

- The number of objects of class trustedDomain that are children of the object whose DN is "CN=System,*Domain.ExistingDN*" equals Domain.CountTrusts.

If an NC Rename operation is attempted when any of these conditions are not met, the NC Rename operation is not performed and the operation returns an error. This protocol does not prescribe what error is to be returned; the value of the error is strictly for implementation debugging purposes, and clients cannot rely on consistent or meaningful return codes.

3.1.1.12.5 Process Changes

To perform the NC Rename operation, the following changes are completed. No ordering of these changes is implied or required. When an object is referred to by DN, the value of the DN is the value before any changes have been completed. Except where indicated, the metadata of changed objects is not updated to reflect the changes. Where the metadata is not updated, the changes are not replicated.

As in the previous section, *NR* is the NCRenameDescription tuple (section 3.1.1.12.1.11) describing the NC Rename operation.

- For the object referred to by the DN "CN=Enterprise Configuration,CN=Partitions,CN=Configuraiton,*NR.RootDomain.ExistingDN*":
  - The dnsRoot attribute is set to NR.RootDomain.NewDNSName.
  - The metadata of the object is updated to reflect this change.
- For the object referred to by the DN "CN=Enterprise Schema,CN=Partitions,CN=Configuration,*NR.RootDomain.ExistingDN*":
  - The dnsRoot attribute is set to NR.RootDomain.NewDNSName.
  - The metadata of the object is updated to reflect this change.
- For every NCDescription *AppNC* in NR.AppNCs:
  - The DN of the object whose GUID is AppNC.Guid is set to AppNC.NewDN.
  - For the object referred to by "CN=*AppNC.ExistingFlatName*,CN=Partitions,CN=Configuraiton,*NR.RootDomain.ExistingDN*":
    - The dnsRoot attribute on the object is set to AppNC.NewDNSName.
- For the root domain described by the DomainDescription tuple in NR.RootDomain:
  - The DN of the object referred to by NR.RootDomain.GUID is set to NR.RootDomain.NewDN.
  - For the object referred to by "CN=*NR.RootDomain.ExistingFlatName*,CN=Partitions,CN=Configuration,*NR.RootDomain.ExistingDN*":
    - The RDN is set to NR.RootDomain.NewFlatName.
    - The dnsRoot attribute is set to NR.RootDomain.NewDNSName.

        - The msDS-DnsRootAlias attribute is set to NR.RootDomain.ExistingDNSName.
        - The nETBIOSName attribute is set to NR.RootDomain.NewFlatName.
- For every DomainDescription *Domain* in NR.TrustTreeRootDomains:
    - The DN of the object referred to by Domain.Guid is set to Domain.NewDN.
    - For the object referred to by "CN=*Domain.ExistingFlatName*,CN=Partitions,CN=Configuration,*NR.RootDomain.ExstingDN*":
        - The RDN is set to Domain.NewFlatName.
        - The dnsRoot attribute is set to Domain.NewDNSName.
        - The nETBIOSName attribute is set to Domain.NewFlatName.
        - Any values of the trustParent attribute are removed.
        - The rootTrust attribute is set to Domain.NewDN.
        - The msDS-DnsRootAlias attribute is set to Domain.ExistingDNSName.
- For every DomainWithNewTrustParentDescription *Domain* in NR.TrustTreeNonRootDomains:
    - The DN of the object whose GUID is Domain.Guid is set to Domain.NewDN.
    - For the object referred to by "CN=*Domain.ExistingFlatName*,CN=Partitions,CN=Configuration,*NR.RootDomain.ExstingDN*":
        - The RDN is set to Domain.NewFlatName.
        - The dnsRoot attribute is set to Domain.NewDNSName.
        - The nETBIOSName attribute is set to Domain.NewFlatName.
        - The trustParent attribute is set to Domain.NewTrustParentFlatName.
        - Any values of the rootTrust attribute are removed.
        - The msDS-DnsRootAlias attribute is set to Domain.ExistingDNSName.
- For every DomainDescription *Domain* in NR.AllDomains, where Domain.Guid refers to a writable object in an NC replica hosted on this DC:
    - For every TrustedDomainObjectDescription *TrustedDomainObject* in Domain.TrustedDomainObjects:
        - For the object referred to by "CN=*TrustedDomainObject.ExistingTrustPartnerDNSName*,CN=System,*Domain.ExistingDN*" :
            - The RDN is set to TrustedDomainObject.NewTrustPartnerDNSName.
            - The flatName attribute is set to TrustedDomainObject.NewTrustPartnerFlatName.
            - The trustPartner attribute is set to TrustedDomainObject.NewTrustPartnerDNSName.
            - The metadata of the object is updated to reflect these changes.

- For every InterdomainTrustAccountDescription *InterdomainTrustAccount* in Domain.InterdomainTrustAccounts:
  - For the object referred to by "CN=*InterdomainTrustAccount.ExistingFlatName*,*InterdomainTrustAccount.ParentDNFromDomainDN*,*Domain.ExistingDN*"
    - The RDN is set to InterdomainTrustAccount.NewFlatName.
    - The sAMAccountName attribute is set to InterdomainTrustAccount.NewFlatName.
    - The metadata of the object is updated to reflect these changes.
- The msDS-ReplicationEpoch attribute on the DC's NTDS Settings object (section 6.1.1.2.2.1.2.1.1) is set to NR.NewReplicationEpoch.

When the changes have been successfully performed, the NC Rename operation returns a value of success. If some part of the NC Rename operation is not or cannot be performed, this protocol does not restrict what changes do occur in the abstract data of the NC performing the NC Rename operation, nor what the return value from the operation is. Such changes can be nondeterministic, and no expectation can be made by the user of the NC Rename operation as to what the result of a failed NC Rename operation will be. In order to improve the usability of this operation, it is suggested to implementers that, in this failure case, no changes be made and an error be returned.

4 Protocol Examples

Note To examine a sample scenario for joining a domain, see [MS-SYS] Appendix A.

The Active Directory Technical Specification (this document) does not specify a protocol, but rather a state model and a set of behaviors that must be followed such that protocols in the documentation set (for instance, the protocols specified in [MS-DRSR] and [MS-SAMR]) will expose the expected behavior to Windows clients. While this document includes a discussion of LDAP, it does so only to specify Active Directory's conformance with and extensions to that protocol, not to specify the protocol itself.

As a result, no protocol examples are appropriate for this document. This section is left in place to maintain section numbering consistency with the documentation template that is used throughout the protocol documentation set.

5 Security

5.1 LDAP Security

References

LDAP attributes userPassword, sAMAccountName, userPrincipalName, uPNSuffixes, supportedCapabilities, servicePrincipalName, nTSecurityDescriptor, schemaIDGUID, attributeSecurityGUID, dSHeuristics, validAccesses, rightsGuid, appliesTo: [MS-ADA1], [MS-ADA2], [MS-ADA3]

LDAP object class controlAccessRight: [MS-ADSC]

ACCESS_MASK structure and access right bits: [MS-DTYP] section 2.4.3

ACE structure: [MS-DTYP] section 2.4.4

ACL structure: [MS-DTYP] section 2.4.5

SECURITY_DESCRIPTOR structure: [MS-DTYP] section 2.4.6

5.1.1 Authentication

This section discusses the use of the LDAP bind mechanism in Active Directory to perform authentication, and the various authentication methods that are supported.

5.1.1.1 Supported Authentication Methods

[RFC2251] section 4.2 defines an AuthenticationChoice structure for a BindRequest that contains two alternatives: simple and SASL. [RFC1777] section 4.1 defines an authentication structure for a BindRequest that contains three alternatives: simple, krbv42LDAP, and krbv42DSA. Active Directory supports only simple and SASL authentication mechanisms. The former is for LDAP simple binds, while the latter is for LDAP SASL binds (as documented in [RFC2829]). In addition,Active Directory supports a third mechanism named "Sicily" that is primarily intended for compatibility with legacy systems. Sicily support adds three choices to the AuthenticationChoice structure, resulting in the following.

```
AuthenticationChoice ::= CHOICE {
    simple                  [0]    OCTET STRING,
    sasl                    [3]    SaslCredentials
    sicilyPackageDiscovery [9]    OCTET STRING
    sicilyNegotiate         [10]   OCTET STRING
    sicilyResponse          [11]   OCTET STRING  }
```

The relationship of the three authentication mechanisms, and the authentication protocols supported by each, is summarized in the following tables.

Authentication Mechanism: Simple

For the simple authentication mechanism, authentication is described entirely by the mechanism; no additional authentication protocols are used.

Authentication Mechanism: SASL

Authentication protocols	Comments
GSS-SPNEGO	GSS-SPNEGO, in turn, uses Kerberos or NTLM as the underlying authentication protocol.
GSSAPI	GSSAPI, in turn, always uses Kerberos as the underlying authentication protocol.
EXTERNAL	-
DIGEST-MD5	-

Authentication Mechanism: Sicily

Authentication protocols	Comments
NTLM	-

Each of the three authentication mechanisms supported by Active Directory is discussed in more detail in the following sections.

5.1.1.1.1 Simple Authentication

The support of simple bind in Active Directory is consistent with [RFC2251] section 4.2 and [RFC2829]. Active Directory does not require, but supports, the use of an SSL/TLS-encrypted or otherwise protected connection when performing a simple bind. Also, while section 6.2 of [RFC2829] specifies that an object possessing a userPassword attribute is a prerequisite to being able to perform a simple bind using that object's credentials, Active Directory does not use the userPassword attribute to store the user's password in most cases, and possession of such an attribute is not a prerequisite to performing a simple bind against an object. The password attributes used in Active Directory are discussed in more detail in "LDAP Password Modify Operations" in section 3.1.1.3.1.5. The simple bind uses the password policy settings described in the Group Policy: Security Protocol [MS-GPSB] section 2.2.1.2 and is applied using the policy described in [MS-GPSB] section 3.2.5.2.

When performing a simple bind, Active Directory accepts several forms of name in the name field of the BindRequest. Each name form is tried in turn. If the name field of the BindRequest maps to a single object using the attempted name form, the password on that object is checked, and the authentication succeeds or fails (with the error *invalidCredentials* / *<unrestricted>*) depending on the result. If the name field of the BindRequest maps to more than one object, the BindRequest fails with the error *invalidCredentials* / *ERROR_INVALID_PARAMETER*. If the name field of the BindRequest maps to no object, the next object name form is tried; if all forms have been tried, the BindRequest fails with the error *invalidCredentials* / *ERROR_INVALID_PARAMETER*.

For AD DS, the name forms are tried in the order they are listed below. For AD LDS, the name forms are tried in the order below, except that forms marked "Only for AD DS" are not tried, and the User Principal Name (UPN) mapping (the second form below) is tried last.

The name forms are:

1. The DN of the object.
2. The user principal name (UPN) of the object. The UPN of an object is either:
   - A value of the userPrincipalName attribute of the object, or

- Only for AD DS: The value of the sAMAccountName attribute of the object, followed by a "@" sign, followed by either:
  - The DNS name of a domain in the same forest as the object, or
  - A value in the uPNSuffixes attribute of the Partitions container in the config NC replica.

  When a name matches both the userPrincipalName attribute of one object and the UPN generated from the sAMAccountName of another object, the simple bind processing attempts to authenticate as the first object (that is, priority is given to the value of the userPrincipalName attribute) rather than failing the bind due to duplicate objects.

3. Only for AD DS: The NetBIOS domain name, followed by a backslash ("\"), followed by the value of the sAMAccountName attribute of the object.
4. The canonical name of the object.
5. The value of the objectGUID attribute of the object, expressed in dashed-string form ([RFC4122] section 3) and surrounded by curly braces (for example, "{ca2e693f-6280-4589-9376-b3707345d3ad}").
6. The value of the displayName attribute of the object.
7. Only for AD DS: A value of the servicePrincipalName attribute of the object.
8. Only for AD DS: A value V that, when the MapSPN(V, M) algorithm of [MS-DRSR] section 4.1.4.2.19 is applied to it, corresponds to a value of the servicePrincipalName attribute of the object. M is the value of the sPNMappings attribute of the nTDSService object.
9. The value of the objectSid attribute of the object, in SDDL SID string form ([MS-DTYP] section 2.4.2.1).
10. Only for AD DS: A value from the sIDHistory attribute of the object, in SDDL SID string form ([MS-DTYP] section 2.4.2.1).
11. The canonical name of the object in which the rightmost forward slash (/) is replaced with a newline character (\n).

5.1.1.1.2 SASL Authentication

The support of SASL bind in Active Directory is consistent with [RFC2251] section 4.2.1 and [RFC2829]. The following SASL mechanisms are supported by Active Directory. They are briefly described in "LDAP SASL Mechanisms", section 3.1.1.3.4.5:

- GSS_SPNEGO [MS-SPNG]
- GSSAPI [RFC2078]
- EXTERNAL [RFC2829]
- DIGEST-MD5 [RFC2831]

Active Directory supports the optional use of integrity verification or encryption that is negotiated as part of the SASL authentication. While Active Directory permits SASL binds to be performed on an SSL/TLS-protected connection, it does not permit the use of SASL-layer encryption/integrity verification mechanisms on such a connection. While this restriction is present in Active Directory on Windows 2000 Server operating system, Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012

operating system, and Windows Server 2012 R2 operating system, versions prior to Windows Server 2008 can fail to reject an LDAP bind that is requesting SASL-layer encryption/integrity verification mechanisms when that bind request is sent on a SSL/TLS-protected connection.

Once a SASL-layer encryption/integrity verification mechanism is in use on a connection, the client SHOULD not send an additional bind request on that connection (for example, to change the credentials with which the connection is authenticated), unless the LDAP_CAP_ACTIVE_DIRECTORY_LDAP_INTEG_OID capability is present in the supportedCapabilities attribute of the rootDSE for that DC (see "LDAP Capabilities" in section 3.1.1.3.4.3). If the client sends an additional bind to a DC on which that capability is not present, the DC returns the *unwillingToPerform* / *ERROR_DS_INAPPROPRIATE_AUTH* error.

Regarding [RFC2829] section 9: when using the EXTERNAL SASL mechanism, Active Directory supports the **authzId** field. However, it only supports the dnAuthzId form and not the uAuthzId form. Additionally, it does not permit an authorization identity to be established on the connection that is different from the authentication identity used on the connection. Violation of either of these rules causes the DC to return the *invalidCredentials* / <*unrestricted*> error.

Regarding [RFC2829] section 6.1: when using the DIGEST-MD5 mechanism:

- On Windows 2000 operating system, Windows Server 2003, Windows Server 2003 R2 operating system, Windows Server 2008, and Windows Server 2008 R2, Active Directory does not support subsequent authentication, although the credentials field contains the string defined by "response-auth" in [RFC2831] section 2.1.3.
- On Windows Server 2008 R2 SP1, Windows Server 2012, and Windows Server 2012 R2, Active Directory also does not support subsequent authentication, but will respond to such requests with an initial authentication challenge (see [RFC2831] section 2.1.1).

5.1.1.1.3 Sicily Authentication

Sicily is a combination of a package discovery mechanism and an authentication mechanism. Unlike SASL, Sicily includes package discovery in the authentication mechanism itself. The package discovery mechanism permits a client to discover the authentication protocols (also known as packages) that a DC supports. The authentication mechanism then permits a client to authenticate using one of those protocols. The authentication mechanism is independent of the package discovery mechanism in that a client may skip the package discovery mechanism entirely and proceed directly to the authentication mechanism (for example, if the client has some out-of-band knowledge of which authentication protocols the server supports).

Windows 2000 Server operating system, Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system versions of Active Directory expose and support only the NTLM authentication protocol, as specified in [MS-NLMP], via Sicily.

The package discovery mechanism is performed by the client sending a BindRequest to the DC in which the name field of the BindRequest is empty and the authentication field contains the sicilyPackageDiscovery choice. The octet string contained in the sicilyPackageDiscovery choice is not used and is empty.

The DC responds to a sicilyPackageDiscovery by returning a SicilyBindResponse. A SicilyBindResponse is similar to an [RFC2251] BindResponse, but some of the fields differ. The SicilyBindResponse is defined as follows.

```
SicilyBindResponse ::= [APPLICATION 1] SEQUENCE {
```

```
resultCode   ENUMERATED {
                 success                      (0),
                 protocolError                (2),
                 adminLimitExceeded           (11),
                 inappropriateAuthentication (48),
                 invalidCredentials           (49),
                 busy                         (51),
                 unavailable                  (52),
                 unwillingToPerform           (53),
                 other                        (80) },

serverCreds  OCTET STRING,
errorMessage LDAPString }
```

Note that resultCode is a subset of the enumeration present in LDAPResult. If the sicilyPackageDiscovery request is successful, the DC sets the resultCode to *success* in its SicilyBindResponse, and returns a string in serverCreds consisting of the semicolon-separated names of the authentication protocols it supports via the Sicily authentication mechanism. Active Directory supports NTLM, and returns the string "NTLM" in the package discovery response. The names of the authentication protocols are ordered in the server's preferred order, starting with the most-preferred authentication protocol. If the sicilyPackageDiscovery request is not successful, the DC returns an error in the **resultCode** field of the SicilyBindResponse. If the sicilyPackageDiscovery request fails because the DC does not support any authentication protocols via Sicily, the DC returns the error *inappropriateAuthentication* / *ERROR_DS_INAPPROPRIATE_AUTH*. The **errorMessage** field of the SicilyBindResponse may contain additional implementation-specific details indicating why the request failed.

Once the client has determined which authentication protocol it will use, it uses the Sicily authentication mechanism to authenticate the connection. The Sicily authentication mechanism consists of two requests, both of which take the form of an LDAP BindRequest. The first request is the sicilyNegotiate request. If successful, this is followed by the sicilyResponse request.

The authentication begins when the client sends the sicilyNegotiate request to the DC. This constitutes a BindRequest in which the name field is set to "NTLM" and the authentication field contains the sicilyNegotiate choice. The sicilyNegotiate choice contains an octet string consisting of binary data supplied by and dependent on the authentication protocol that is used, and which serves as a representation of the credentials with which the client wishes to authenticate the connection. If successful, the DC responds with a SicilyBindResponse in which the resultCode is set to *success* and the serverCreds contains binary data supplied by the authentication protocol on the server side. The client is expected to pass this binary data, whose content is authentication protocol–specific, to its implementation of the authentication package. If not successful, the DC returns an error in the **resultCode** field of the SicilyBindResponse, indicating that the sicilyNegotiate request was not successful. If the credentials supplied by the client are invalid, the DC returns the *invalidCredentials* / *<unrestricted>* error. If the client requests an authentication protocol that is not supported by the DC, it returns the *inappropriateAuthentication* / *ERROR_DS_INAPPROPRIATE_AUTH* error. The errorMessage field of the SicilyBindResponse may contain additional implementation-specific details indicating why the request failed.

If the sicilyNegotiate request is successful, the client then sends the sicilyResponse request to the DC by sending a BindRequest in which the name field is empty and the authentication field contains the sicilyResponse choice. The octet string in the sicilyResponse choice contains authentication protocol–specific data, generated in response to the data received in the serverCreds field of the SicilyBindResponse. The DC responds to this sicilyResponse request by sending a SicilyBindResponse. The serverCred field is not used in this response, and is empty. If successful, the DC sets the resultCode field to *success*, and the connection is now authenticated as the client-

supplied credentials. If the bind fails, the DC sets resultCode to an error and the connection is not authenticated. As in the previous case, the DC uses the error *invalidCredentials* / <*unrestricted*> to indicate that the client presented incorrect credentials, and the error *inappropriateAuthentication* / *ERROR_DS_INAPPROPRIATE_AUTH* to indicate that the client requested an unsupported protocol. The errorMessage field of the SicilyBindResponse may contain additional implementation-specific details indicating why the request failed.

As with SASL, integrity verification or encryption can be negotiated as part of the Sicily authentication. The support for, and means of implementation of, such mechanisms is dependent on the particular authentication protocol used (for example, NTLM). As with SASL, such mechanisms cannot be used on a connection that is protected by SSL/TLS mechanisms, and once such a mechanism is in use, the connection cannot be rebound unless the LDAP_CAP_ACTIVE_DIRECTORY_LDAP_INTEG_OID capability is present in the supportedCapabilities attribute of the rootDSE of the DC.

5.1.1.2 Using SSL/TLS

Active Directory permits two means of establishing an SSL/TLS-protected connection to a DC. The first is by connecting to a DC on a protected LDAPS port (TCP ports 636 and 3269 in AD DS, and a configuration-specific port in AD LDS). The second is by connecting to a DC on a regular LDAP port (TCP ports 389 or 3268 in AD DS, and a configuration-specific port in AD LDS), and later sending an LDAP_SERVER_START_TLS_OID extended operation [RFC2830]. In both cases, the DC will request (but not require) the client's certificate as part of the SSL/TLS handshake [RFC2246]. If the client presents a valid certificate to the DC at that time, it can be used by the DC to authenticate (bind) the connection as the credentials represented by the certificate.

If the client establishes the SSL/TLS-protected connection by means of connecting on a protected LDAPS port, then the connection is considered to be immediately authenticated (bound) as the credentials represented by the client certificate. An EXTERNAL bind is not required but is permitted. If the client does not present a certificate during the SSL/TLS handshake, the connection is not authenticated and is treated as anonymous. In that case, the DC rejects any attempt to perform an EXTERNAL bind with the error *invalidCredentials* / <*unrestricted*>.

If the client establishes the SSL/TLS-protected connection by means of an LDAP_SERVER_START_TLS_OID operation, the authentication state of the connection remains the same after the operation as it was before the operation. The DC authenticates the connection as the credentials represented by the client's certificate only if an EXTERNAL SASL bind is subsequently performed. This is similar to the "implicit assertion" of [RFC2830] section 5.1.2.1, except that neither the authentication identity nor the authorization identity is established on the connection until the EXTERNAL bind takes place. If the client includes the authzId field in the EXTERNAL bind, in accord with the "explicit assertion" of [RFC2830] section 5.1.2.2, then as described in section 5.1.1.1.2 the authzId field contains the DN of the object that the EXTERNAL bind is authenticating the connection as; in other words, the object associated with the credentials represented by the certificate. Therefore, the implicit assertion and explicit assertion are functionally identical. If the client performs an EXTERNAL bind but does not supply a certificate during the SSL/TLS handshake, the EXTERNAL bind fails with the error *invalidCredentials* / <*unrestricted*>.

Alternatively, the client can perform any other form of LDAP bind that is permissible on an SSL/TLS-protected connection, or the client can perform no bind to continue to use any authentication and authorization identity that was previously established on the connection.

5.1.1.3 Using Fast Bind

Active Directory supports a mode of operation known as "fast bind" that can be enabled for each LDAP connection. Fast bind mode allows a client to use the LDAP bind request to simply validate

credentials and authenticate the client without the overhead of establishing the authorization information. Fast bind mode is enabled on a connection by sending the LDAP_SERVER_FAST_BIND_OID LDAP extended operation on the connection, documented in "LDAP Extended Operations" in section 3.1.1.3.4.2.

Once fast bind mode is enabled on a connection, it cannot be disabled on that connection. This mode cannot be enabled on a connection on which a successful bind was previously performed, and the server returns *unwillingToPerform* / *ERROR_DS_INAPPROPRIATE_AUTH* if such an attempt is made.

When fast bind mode is enabled on an LDAP connection, the DC accepts bind requests and validates the credentials presented, returning an error code that indicates a success or failure. However, on successful binds, the DC does not perform authorization steps, and the connection is treated as if it was authorized as the anonymous user.

While [RFC2251] section 4.2.1 specifies that a bind request causes all operations currently in progress on a connection to be abandoned, when the connection is in fast bind mode, multiple independent binds (for example, using different credentials) can simultaneously be in progress on the same connection without any of them being abandoned. This permits a client to validate multiple sets of credentials at the same time, while the DC always considers the connection to be authenticated and authorized as the anonymous user.

Only simple binds are accepted on a connection in fast bind mode. The client can use SSL/TLS protection on a connection in fast bind mode.

5.1.1.4 Mutual Authentication

[MS-DRSR] sections 2.2.2 and 2.2.4 specify the mutual authentication requirements for client-to-DC interactions over the RPC interfaces documented in [MS-DRSR]. The requirements are the same for mutual authentication in an LDAP connection.

Therefore, by registering its SPNs for the RPC interfaces documented in [MS-DRSR], a DC also satisfies its SPN registration requirements for LDAP.

5.1.1.5 Supported Types of Security Principals

For AD DS, the concept of "security principal" is straightforward: a security principal is an object in the directory that possesses an objectSid attribute. But for AD LDS, the notion of security principal is more complex, because AD LDS recognizes three distinct types of security principals, any of which can authenticate via an LDAP Bind request:

- AD LDS security principals that are created in an AD LDS NC.
- Principals that are defined by the operating system of the computer on which AD LDS is running.
- Principals that are defined in an Active Directory domain to which the computer on which AD LDS is running is joined, or principals that are in domains that are trusted by the joined domain.

In addition to these three types of security principals, AD LDS also supports bind proxies, which are not security principals but which can be authenticated via an LDAP Bind request. This section will discuss each of the three types of security principals in turn, and follow that with a discussion of bind proxies. Finally, it will conclude with an explanation of which types of LDAP Binds an AD LDS server must support for each type of principal and bind proxy.

The first type of security principal in AD LDS is unchanged from AD DS: an object in the directory that possesses an objectSid attribute. However, while AD DS restricts security principals to the

domain NC, AD LDS (which has no domain NCs) permits security principals to be stored in an application NC. Additionally, if the ADAMAllowADAMSecurityPrincipalsInConfigPartition configuration setting is supported and equals 1 (section 3.1.1.3.4.7), AD LDS permits security principals to be created in the config NC.

In AD DS, the set of security principal object classes is fixed. In AD LDS, any object class that statically links (section 3.1.1.2.4.6) to the msDS-BindableObject auxiliary class is a security principal object class. Dynamically instantiating the msDS-BindableObject auxiliary class does not have the same effect.

The second and third types of principals are similar to each other in that both are means for AD LDS to "pass through" the Authentication to the underlying operating system on which it is running. AD LDS recognizes as a security principal those security principals (users and groups) that are stored locally on the computer on which AD LDS is running. Additionally, if the computer is a member of a domain, then AD LDS recognizes as security principals any security principals that are in that domain or which are in a domain trusted by that domain. Such security principals may be included in the security descriptors of objects in the AD LDS directory in the same fashion as security principals of the first type. Additionally, such security principals may be included in the membership of group objects in AD LDS, and in the msDS-ServiceAccount attribute of nTDSDSA objects in AD LDS, via the automatic creation of foreignSecurityPrincipal objects (sections 3.1.1.5.2.4 and 3.1.1.5.3.3).

Note that, except for the creation of foreignSecurityPrincipal objects as needed to represent group members or service accounts, the second and third types of principals are not represented as objects in AD LDS. Instead, upon receipt of an LDAP Bind request for such a principal, AD LDS provides the credentials it receives in the Bind request to the host operating system and relies on the host operating system to validate those credentials. The means of passing the received credentials to the host operating system, as well as the method that the host operating system uses to validate those credentials, is implementation-specific.

Bind proxies are objects in AD LDS that contain the msDS-BindProxy auxiliary class. A bind proxy contains an objectSid attribute but is not a security principal. Rather, it is a means of associating an object in AD LDS with a security principal of the underlying operating system (that is, the second or third type of security principal). The objectSid attribute contains the SID of a security principal of the second or third type. When an LDAP Bind request is received in which the object identified in the name field of the BindRequest is an msDS-BindProxy object, the server performs the following actions:

- Retrieve the value V of the objectSid attribute from the named object.
- Pass through the Authentication request to the host operating system as a request to authenticate a principal whose SID is V and whose password is as supplied in the LDAP Bind request.

An LDAP Bind request that targets an msDS-BindProxy object O has nearly the same effect as an LDAP Bind request for a security principal S of the second or third type. Instead of directly naming S in the LDAP Bind request, the client names an object O such that O!objectSid equals the SID of S. The security context generated by the two requests is slightly different, as specified in section 5.1.3.4.

In order for an object class to be usable in an LDAP Bind request in AD LDS, that object class must either contain the msDS-BindableObject class or the msDS-BindProxy class.

AD LDS servers restrict the authentication mechanisms and protocols that can be used to authenticate different types of security principal and bind proxies. The authentication mechanisms

and protocols supported by AD LDS for each type of principal or proxy are specified in the following table.

Type of principal/proxy	Supported authentication mechanism	Supported authentication protocol
First type (AD LDS principal)	Simple SASL	- DIGEST-MD5*
Second or third type (computer or domain principal)	SASL SASL SASL SASL Sicily	GSSAPI GSS-SPNEGO DIGEST-MD5 EXTERNAL NTLM
Bind proxy	Simple	-

* DIGEST-MD5 authentication for AD LDS security principals is supported only when the ADAMDisableSSI configurable setting (section 3.1.1.3.4.7) is supported and is equal to 0. If the ADAMDisableSSI configurable setting is not supported, then DIGEST-MD5 authentication for AD LDS security principals is not supported.

In particular, note that simple bind is not supported for principals of the second or third type, and that DIGEST-MD5 is the only SASL protocol supported for all types of security principals in AD LDS.

5.1.2 Message Security

5.1.2.1 Using SASL

Active Directory supports the optional use of an LDAP message security layer that provides message integrity and/or confidentiality protection services that are negotiated as part of the SASL authentication. Support for such mechanisms and their implementation is dependent on the specific authentication protocol used (for example, Kerberos or Digest), and is documented in the SASL specification for each authentication protocol.

Once a SASL-negotiated security layer is in effect in the LDAP data stream, it remains in effect until either a subsequently negotiated security layer is installed or the underlying transport connection is closed. When in effect, the security layer processes protocol data into buffers of protected data as per [RFC2222].

While Active Directory permits SASL binds to be performed on an SSL/TLS-protected connection, it does not permit the use of SASL-layer confidentiality/integrity protection mechanisms on such a connection. Active Directory can also be configured to require that SASL layer integrity protection services be used on a LDAP connection (the way in which the configuration can be done is outside the scope of the state model and is implementation-dependent).

On Windows 2000 Server operating system, Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 operating system with Service Pack 2 (SP2), Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, Active Directory treats a request for SASL-layer integrity protection and SASL-layer confidentiality protection distinctly. Therefore, if a client does not request SASL-layer integrity protection or requests SASL-layer confidentiality protection without requesting integrity protection when sending a bind request to a DC which is configured to require SASL-layer integrity protection, the DC will reject such a bind and return the error *strongAuthRequired* /

ERROR_DS_STRONG_AUTH_REQUIRED. On Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2, Active Directory treats a request for SASL-layer confidentiality protection as also requesting SASL-layer integrity protection; therefore, a DC that is configured to require SASL-layer integrity protection will accept a bind from a client that requests SASL-layer confidentiality protection but does not explicitly request SASL-layer integrity protection. A DC configured to require SASL-layer integrity protection will accept a bind request from a client sent on a SSL/TLS-protected connection even if the client does not request SASL-layer integrity because it will accept the SSL/TLS-encryption in lieu of SASL-layer integrity.

5.1.2.2 Using SSL/TLS

Active Directory supports LDAP message security on an SSL/TLS-protected connection to a DC in accordance with [RFC2246].

As indicated in the previous section, Active Directory does not permit SASL-layer message confidentiality/integrity protection mechanisms to be employed on an SSL/TLS-protected LDAP connection.

5.1.3 Authorization

Although the LDAP security model does not include mechanisms for access control, Active Directory provides access control in the form of **access control lists (ACLs)** on directory objects.

If the fLDAPBlockAnonOps heuristic of the dSHeuristics attribute (see section 6.1.1.2.4.1.2) is true, anonymous (unauthenticated) users are limited to performing rootDSE searches and binds. If fLDAPBlockAnonOps is false, anonymous users can perform any LDAP operation, subject to access checks that use the ACL mechanisms described in this section.

5.1.3.1 Background

The security context of a requester (see security context in the Glossary) requesting access to an Active Directory object represents the authorization information that is associated with the requester. A DC performs an access check to determine whether the security context, and thus the requester, is authorized for the type of access that has been requested before allowing any further processing to continue. Access control information associated with an object is contained in the security descriptor of the object.

Every object in Active Directory has an nTSecurityDescriptor attribute whose value is the security descriptor that contains access control information for the object.

An access check compares information in the thread's security context with information in the object's security descriptor:

- The security context contains a SID that identifies the principal associated with the thread, and SIDs that identify the groups of which the principal is a member.
- The security descriptor contains a DACL that specifies the access rights that are allowed or denied to specific principals or groups. It also identifies the owner of the object. The structure of a security descriptor is described in [MS-DTYP] section 2.4.6.

A DACL in a security descriptor is an ordered list of access control entries (ACEs) that define the protections that apply to an object and its properties. Each ACE identifies a security principal (that is, a user, group, and so on) and specifies a set of access rights that are allowed, denied, or audited for that security principal. The data structures for an ACE and a DACL are described in [MS-DTYP] sections 2.4.4 and 2.4.5.

There are two types of ACEs: simple and object-specific. A simple ACE applies to an entire object. If a simple ACE gives a particular user read access, the user can read all information associated with the object. An object-specific ACE, on the other hand, can apply to any individual attribute of an object or to a set of attributes. It makes it possible to place independent access controls on each attribute of an Active Directory object.

During an access check, the server steps through the ACEs in the order in which they appear in the object's DACL, looking for ACEs that apply to the principal and group SIDs from the thread's security context. It steps through each ACE until it finds one that either allows or denies access to the principal or one of the principal's groups, or until there are no more ACEs to check. If it comes to the end of the DACL, and the thread's desired access is still not explicitly allowed or denied, the server denies access to the object.

The order in which ACEs are listed in a DACL is important. For example, an object's DACL might contain one ACE that allows access to a group and another ACE that denies access to a principal who is a member of the group. If the access check process encounters the ACE that allows access to the group before the ACE that denies access to the principal, the principal is allowed access to the object. If the ACEs are encountered in the reverse order, then the principal is denied access to the object.

AD LDS security principals cannot appear in an AD DS ACE. Section 6.1.3.3 specifies a restriction on the AD LDS security principals that can be used in an AD LDS ACE.

5.1.3.2 Access Rights

The following diagram specifies access rights that can be assigned to or requested for an Active Directory object. The access mask in an ACE contains a combination of these values.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
GR	GW	GX	GA	X	X	X	X	X	X	X	X	WO	WD	RC	DE	X	X	X	X	X	X	X	CR	LO	DT	WP	RP	VW	LC	DC	CC

Note The values are presented in big-endian byte order.

CC (RIGHT_DS_CREATE_CHILD, 0x00000001): The right to create child objects of the object. The ObjectType member of an ACE can contain a GUID that identifies the objectClass of child object whose creation is controlled. If ObjectType does not contain a GUID, the ACE controls the creation of all child object classes allowed by the schema.

DC (RIGHT_DS_DELETE_CHILD, 0x00000002): The right to delete child objects of the object. The ObjectType member of an ACE can contain a GUID that identifies the objectClass of the child object whose deletion is controlled. If ObjectType does not contain a GUID, the ACE controls the deletion of all child object classes.

LC (RIGHT_DS_LIST_CONTENTS, 0x00000004): The right to list child objects of this object. For more information about this right, see section 3.1.1.4.

VW (RIGHT_DS_WRITE_PROPERTY_EXTENDED, 0x00000008): The right to perform an operation controlled by a validated write access right. The ObjectType member of an ACE can contain a GUID that identifies the validated write. If ObjectType does not contain a GUID, the ACE controls the rights to perform all validated write operations associated with the object. For a list of

validated write rights, see section 5.1.3.2.2. For specifics of validated write processing, see the Modify operation in section 3.1.1.5.3.

RP (RIGHT_DS_READ_PROPERTY, 0x00000010): The right to read properties of the object. The ObjectType member of an ACE can contain a GUID that identifies a property set or an attribute. If ObjectType does not contain a GUID, the ACE controls the right to read all attributes of the object.

WP (RIGHT_DS_WRITE_PROPERTY, 0x00000020): The right to write properties of the object. The ObjectType member of an ACE can contain a GUID that identifies a property set or an attribute. If ObjectType does not contain a GUID, the ACE controls the right to write all attributes of the object.

DT (RIGHT_DS_DELETE_TREE, 0x00000040): The right to perform a Delete-Tree operation on this object. See the Delete operation in section 3.1.1.5.5 for more details.

LO (RIGHT_DS_LIST_OBJECT, 0x00000080): The right to list a particular object. If the user is not granted this right, and the user is not granted the RIGHT_DS_LIST_CONTENTS right on the object's parent, the object is hidden from the user. Note that LIST_OBJECT rights are not enforced by Active Directory by default. In order to enable LIST_OBJECT enforcement, the fDoListObject heuristic of the dSHeuristics attribute (see section 6.1.1.2.4.1.2) must be true.

CR (RIGHT_DS_CONTROL_ACCESS, 0x00000100): The right to perform an operation controlled by a control access right. The ObjectType member of an ACE can contain a GUID that identifies the control access right. If ObjectType does not contain a GUID, the ACE controls the right to perform all control access right controlled operations associated with the object. For a list of control access rights, see section 5.1.3.2.1.

DE (RIGHT_DELETE, 0x00010000): The right to delete the object.

RC (RIGHT_READ_CONTROL, 0x00020000): The right to read data from the security descriptor of the object, not including the data in the SACL.

WD (RIGHT_WRITE_DAC, 0x00040000): The right to modify the DACL in the object security descriptor.

WO (RIGHT_WRITE_OWNER, 0x00080000): The right to modify the owner of an object in the object's security descriptor. A user can only take ownership of an object, but cannot transfer ownership of an object to other users.

GA (RIGHT_GENERIC_ALL, 0x10000000): The right to create or delete child objects, delete a subtree, read and write properties, examine child objects and the object itself, add and remove the object from the directory, and read or write with an extended right.

GX (RIGHT_GENERIC_EXECUTE, 0x20000000): The right to read permissions on, and list the contents of, a container object.

GW (RIGHT_GENERIC_WRITE, 0x40000000): The right to read permissions on this object, write all the properties on this object, and perform all validated writes to this object.

GR (RIGHT_GENERIC_READ, 0x80000000): The right to read permissions on this object, read all the properties on this object, list this object name when the parent container is listed, and list the contents of this object if it is a container.

X: Ignored. These bits are ignored in Active Directory DACLs.

The four generic rights are presented, along with the specific access rights which they represent. The mapping for access to objects in Active Directory is as follows:

GR = (RC | LC | RP | LO)

GW = (RC | WP | VW)

GX = (RC | LC)

GA = (DE | RC | WD | WO | CC | DC | DT | RP | WP | LC | LO | CR | VW)

Note that the preceding "GENERIC" access mask bits are never stored in Active Directory security descriptor values. They can be present in an SD value sent by a user in an add or modify request. When the SD value is stored in the database, the GENERIC access bits are mapped according to the specific access rights that they represent, using the mapping described above. See section 6.1.3 and [MS-DTYP] section 2.4.3 for more information.

5.1.3.2.1 Control Access Rights

In Active Directory, the implementer can control which users have the right to perform a particular operation on an object or its attributes by using standard access rights. However, there are certain operations that have semantics that are not tied to specific properties, or where it is desirable to control access in a way that is not supported by the standard access rights. For example, the implementer can grant users a "Reanimate tombstones" right so that they are able to perform tombstone reanimation on any object in a naming context. Active Directory allows the standard access control mechanism to be extended for controlling access to custom actions or operations, using a mechanism called control access rights.

A control access right is not identified by a specific bit in an access mask as the standard access rights are. Instead, each control access right is identified by a GUID. An ACE that grants or denies a control access right specifies the RIGHT_DS_CONTROL_ACCESS (CR) bit in the **ACCESS_MASK** field and the GUID identifying the particular control access right in the ObjectType field of the ACE. If the **ObjectType** field does not contain a GUID, the ACE is deemed to control the right to perform all operations associated with the objects that are controlled by control access rights. For convenience and easy identification by Active Directory administrative tools facilitating access control, each control access right is represented by an object of class controlAccessRight in the Extended-Rights container. Note that these objects are not integral to evaluating access to an operation and, therefore, their presence is not required for the proper functioning of the access control mechanism. There are a number of predefined control access rights in Active Directory, and that list can be extended by application developers by adding controlAccessRight objects to the Extended-Rights container.

The pertinent attributes on the controlAccessRight object that defines the use of the control access right for the administrative tools are as follows:

- validAccesses: The type of access right bits in the **ACCESS_MASK** field of an ACE with which the control access right can be associated. The only permitted access right for control access rights is RIGHT_DS_CONTROL_ACCESS (CR).
- rightsGuid: The GUID that is used to identify the control access right in an ACE. The GUID value is placed in the **ObjectType** field of the ACE.
- appliesTo: This multivalue attribute has a list of object classes that the control access right applies to. Each object class in the list is represented by the schemaIDGUID attribute of the classSchema object that defines the object class in the Active Directory schema. The appliesTo values on the controlAccessRight are not enforced by the directory server; that is, the

controlAccessRight can be included in security descriptors of objects of classes not specified in the appliesTo attribute.

The following table summarizes the predefined control access rights, and the corresponding GUID value identifying each right, that can be specified in an ACE that is supported by each Windows Server operating system version.

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
Abandon-Replication	ee914b82-0a98-11d1-adbb-00c04fd8d5cd	X									
Add-GUID	440820ad-65b4-11d1-a3da-0000f875ae0d	X	X	X	X	X	X	X	X	X	X
Allocate-Rids	1abd7cf8-0a99-11d1-adbb-00c04fd8d5cd	X	X	X		X		X		X	
Allowed-To-Authenticate	68b1d179-0d15-4d4f-ab71-46152e79a7bc		X	X		X		X		X	

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
Apply-Group-Policy	edacfd8f-ffb3-11d1-b41d-00a0c968f939	X	X	X		X		X		X	
Certificate-Enrollment	0e10c968-78fb-11d2-90d4-00c04f79dc55	X	X	X		X		X		X	
Certificate-AutoEnrollment	a05b8cc2-17bc-4802-a710-e7c15ab866a2							X		X	
Change-Domain-Master	014bf69c-7b3b-11d1-85f6-08002be74fab		X	X		X		X		X	
Change-Infrastructure-Master	cc17b1fb-33d9-11d2-97d4-00c04fd8d5	X	X	X		X		X		X	

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
	cd										
Change-PDC	bae50096-4752-11d1-9052-00c04fc2d4cf	X	X	X		X		X		X	
Change-Rid-Master	d58d5f36-0a98-11d1-adbb-00c04fd8d5cd	X	X	X		X		X		X	
Change-Schema-Master	e12b56b6-0a95-11d1-adbb-00c04fd8d5cd	X	X	X	X	X	X	X	X	X	X
Create-Inbound-Forest-Trust	e2a36dc9-ae17-47c3-b58b-be34c55ba633		X	X		X		X		X	
Do-Garbage-Collection	fec364e0-0a98-11d1-adbb-00c04	X	X	X	X	X	X	X	X	X	X

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
	fd8d5cd										
Domain-Administer-Server	ab721a52-1e2f-11d0-9819-00aa0040529b	X	X	X		X		X		X	
DS-Check-Stale-Phantoms	69ae6200-7f46-11d2-b9ad-00c04f79f805	X	X	X		X		X		X	
DS-Execute-Intentions-Script	2f16c4a5-b98e-432c-952a-cb388ba33f2e		X	X	X	X	X	X	X	X	X
DS-Install-Replica	9923a32a-3607-11d2-b9be-0000f87a36b2	X	X	X	X	X	X	X	X	X	X
DS-Query-Self-Quota	4ecc03fe-ffc0-4947-b630-		X	X	X	X	X	X	X	X	X

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
	eb672a8a9dbc										
DS-Replication-Get-Changes	1131f6aa-9c07-11d1-f79f-00c04fc2dcd2	X	X	X	X	X	X	X	X	X	X
DS-Replication-Get-Changes-All	1131f6ad-9c07-11d1-f79f-00c04fc2dcd2		X	X	X	X	X	X	X	X	X
DS-Replication-Get-Changes-In-Filtered-Set	89e95b76-444d-4c62-991a-0facbeda640c			X		X		X		X	
DS-Replication-Manage-Topology	1131f6ac-9c07-11d1-f79f-00c04fc2dcd2	X	X	X	X	X	X	X	X	X	X
DS-Replication-Monitor-	f98340fb-7c5b-4cdb-		X	X	X	X	X	X	X	X	X

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
Topology	a00b-2ebdfa115a96										
DS-Replication-Synchronize	1131f6ab-9c07-11d1-f79f-00c04fc2dcd2	X	X	X	X	X	X	X	X	X	X
Enable-Per-User-Reversibly-Encrypted-Password	05c74c5e-4deb-43b4-bd9f-86664c2a7fd5		X	X		X		X		X	
Generate-RSoP-Logging	b7b1b3de-ab09-4242-9e30-9980e5d322f7		X	X		X		X		X	
Generate-RSoP-Planning	b7b1b3dd-ab09-4242-9e30-9980e5d322f7		X	X		X		X		X	
Manage-Optional-	7c0e2a7c-a419-					X	X	X	X	X	X

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
Features	48e4-a995-10180aad54dd										
Migrate-SID-History	ba33815a-4f93-4c76-87f3-57574bff8109		X	X		X		X		X	
msmq-Open-Connector	b4e60130-df3f-11d1-9c86-006008764d0e	X	X	X		X		X		X	
msmq-Peek	06bd3201-df3e-11d1-9c86-006008764d0e	X	X	X		X		X		X	
msmq-Peek-computer-Journal	4b6e08c3-df3c-11d1-9c86-006008764d0e	X	X	X		X		X		X	
msmq-Peek-	4b6e08c1-	X	X	X		X		X		X	

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
Dead-Letter	df3c-11d1-9c86-006008764d0e										
msmq-Receive	06bd3200-df3e-11d1-9c86-006008764d0e	X	X	X		X		X		X	
msmq-Receive-computer-Journal	4b6e08c2-df3c-11d1-9c86-006008764d0e	X	X	X		X		X		X	
msmq-Receive-Dead-Letter	4b6e08c0-df3c-11d1-9c86-006008764d0e	X	X	X		X		X		X	
msmq-Receive-journal	06bd3203-df3e-11d1-9c86-006008764d0e	X	X	X		X		X		X	

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
msmq-Send	06bd3202-df3e-11d1-9c86-006008764d0e	X	X	X		X		X		X	
Open-Address-Book	a1990816-4298-11d1-ade2-00c04fd8d5cd	X	X	X		X		X		X	
Read-Only-Replication-Secret-Synchronization	1131f6ae-9c07-11d1-f79f-00c04fc2dcd2			X		X		X		X	
Reanimate-Tombstones	45ec5156-db7e-47bb-b53f-dbeb2d03c40f		X	X	X	X	X	X	X	X	X
Recalculate-Hierarchy	0bc1554e-0a99-11d1-adbb-00c04fd8d5	X	X	X		X		X		X	

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
	cd										
Recalculate-Security-Inheritance	62dd28a8-7f46-11d2-b9ad-00c04f79f805	X	X	X	X	X	X	X	X	X	X
Receive-As	ab721a56-1e2f-11d0-9819-00aa0040529b	X	X	X		X		X		X	
Refresh-Group-Cache	9432c620-033c-4db7-8b58-14ef6d0bf477		X	X		X		X		X	
Reload-SSL-Certificate	1a60ea8d-58a6-4b20-bcdc-fb71eb8a9ff8			X	X	X	X	X	X	X	X
Run-Protect_Admin_Groups-Task	7726b9d5-a4b4-4288-a6b2-dce95					X		X		X	

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
	2e80a7f										
SAM-Enumerate-Entire-Domain	91d67418-0135-4acc-8d79-c08e857cfbec		X	X		X		X		X	
Send-As	ab721a54-1e2f-11d0-9819-00aa0040529b	X	X	X		X		X		X	
Send-To	ab721a55-1e2f-11d0-9819-00aa0040529b	X	X	X		X		X		X	
Unexpire-Password	ccc2dc7d-a6ad-4a7a-8846-c04e3cc53501		X	X	X	X	X	X	X	X	X
Update-Password-Not-Required-	280f369c-67c7-438e-ae98-		X	X		X		X		X	

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
Bit	1d46f3c6f541										
Update-Schema-Cache	be2bb760-7f46-11d2-b9ad-00c04f79f805	X	X	X	X	X	X	X	X	X	X
User-Change-Password	ab721a53-1e2f-11d0-9819-00aa0040529b	X	X	X	X	X	X	X	X	X	X
User-Force-Change-Password	00299570-246d-11d0-a768-00aa006e0529	X	X	X	X	X	X	X	X	X	X
DS-Clone-Domain-Controller	3e0f7e18-2c7a-4c10-ba82-4d926db99a3e							X		X	
DS-Read-Partition-Secrets	084c93a2-620d-4879-									X	X

Control access right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
	a836-f0ae47de0e89										
DS-Write-Partition-Secrets	94825a8d-b171-4116-8146-1e34d8f5401									X	X
DS-Set-Owner	4125c71f-7fac-4ff0-bcb7-f09a413252 86									X	X
DS-Bypass-Quota	4125c71f-7fac-4ff0-bcb7-f09a413252 86									X	X

5.1.3.2.2 Validated Writes

In Active Directory, write access to an object's attributes is controlled by using the RIGHT_DS_WRITE_PROPERTY (WP) access right. However, that would allow any value that is permissible by the attribute schema to be written to the attribute with no value checking performed. There are cases where validation of the attribute values being written, beyond that required by the schema, is necessary before writing them to an object in order to maintain integrity constraints. Active Directory extends the standard access control mechanism to allow such additional validation semantics to be incorporated by using a mechanism called "validated write rights". The attributes to

which the validated write rights apply, and the specific validations performed, are specified in section 3.1.1.5.3.1.

A validated write right is not identified by a specific bit in an access mask as the standard access rights are. Instead, each validated write right is identified by a GUID. This GUID is the value of the schemaIDGUID attribute from the attributeSchema object of the attribute where the validated write is defined. An ACE that grants or denies a validated write right specifies the RIGHT_DS_WRITE_PROPERTY_EXTENDED (VW) bit in the **ACCESS_MASK** field and the GUID identifying the particular validated write right in the **ObjectType** field of the ACE. If the **ObjectType** field does not contain a GUID, the ACE is deemed to control the right to perform all validated write operations associated with the object. As with control access rights, each validated write right is represented by an object of class controlAccessRight in the Extended-Rights container for convenience and easy identification by Active Directory administrative tools. Note that these objects are not integral to evaluating access to an update operation and, therefore, their presence is not required for the proper functioning of the access control mechanism. The predefined list of validated write rights in Active Directory cannot be extended by application developers.

The attributes to which the validated write rights apply to, and the specific validations performed, are specified in section 3.1.1.5.3.1.1. The following table summarizes the validated write rights, and the corresponding GUID value identifying each right, that can be specified in an ACE that is supported by each Windows Server operating system version.

Validated write right symbol	**Identifying GUID used in ACE**	**Windows 2000 operating system**	**Windows Server 2003 operating system**	**Windows Server 2008 operating system AD DS**	**Windows Server 2008 AD LDS**	**Windows Server 2008 R2 operating system AD DS**	**Windows Server 2008 R2 AD LDS**	**Windows Server 2012 operating system AD DS**	**Windows Server 2012 AD LDS**	**Windows Server 2012 R2 operating system AD DS**	**Windows Server 2012 R2 AD LDS**
Self-Membership	bf9679c0-0de6-11d0-a285-00aa003049e2 (member attribute)	X	X	X	X	X	X	X	X	X	X
Validated-DNS-Host-Name	72e39547-7b18-11d1-adef-00c04fd8d5cd (dNSHostName attribute)	X	X	X		X		X		X	

Validated write right symbol	Identifying GUID used in ACE	Windows 2000 operating system	Windows Server 2003 operating system	Windows Server 2008 operating system AD DS	Windows Server 2008 AD LDS	Windows Server 2008 R2 operating system AD DS	Windows Server 2008 R2 AD LDS	Windows Server 2012 operating system AD DS	Windows Server 2012 AD LDS	Windows Server 2012 R2 operating system AD DS	Windows Server 2012 R2 AD LDS
Validated-MS-DS-Additional-DNS-Host-Name	80863791-dbe9-4eb8-837e-7f0ab55d9ac7 (msDS-AdditionalDnsHostName attribute)							X		X	
Validated-MS-DS-Behavior-Version	d31a8757-2447-4545-8081-3bb610cacbf2(msDS-Behavior-Version attribute)							X		X	
Validated-SPN	f3a64788-5306-11d1-a9c5-0000f80367c1 (servicePrincipalName attribute)	X	X	X		X		X		X	

5.1.3.3 Checking Access

Before performing a requested access on an object in Active Directory, the DC performs an access check to confirm that the security context of the requester is authorized for the type of access requested. This determination is made by using the following information:

- The requester's security context
- The requester's desired access mask
- An appropriate security descriptor (the security descriptor used for the access check is typically the security descriptor of the object itself, but for some types of access the security descriptor of the object's parent and/or other objects in the directory may be used).

Note that a special principal called "Principal Self," identified by the fixed SID value of S-1-5-10, may appear in the SID field of an ACE in the security descriptor of an object. This fixed SID value represents the object itself in an ACE on a security principal object. For example, when an ACE on a user object grants certain access rights to Principal Self, it essentially grants those access rights to the user represented by that object. During an access check for object O, if O!nTSecurityDescriptor contains any ACEs with the fixed SID for Principal Self the server replaces them with O!objectSid before proceeding with the access check.

For the access checking behavior described in the following sections, it is assumed that any security descriptor used as input to that process has already undergone the SID substitution for Principal Self (as described in this section), if necessary.

5.1.3.3.1 Null vs. Empty DACLs

The presence of a NULL DACL in the nTSecurityDescriptor attribute of an object grants full access to the object to any principal that requests it; normal access checks are not performed with respect to the object.

An empty DACL, on the other hand, is a properly allocated and initialized DACL containing no ACEs. An empty DACL in the nTSecurityDescriptor attribute of an object grants no access to the object. Note that even with an empty DACL, some rights are implied. For example, the current OWNER of an object is implicitly granted RIGHT_READ_CONTROL and RIGHT_WRITE_DAC access. If the user possesses the SE_TAKE_OWNERSHIP_PRIVILEGE, then RIGHT_WRITE_OWNER access is implied.

5.1.3.3.2 Checking Simple Access

When evaluating standard access rights specified in simple ACEs for an Active Directory object, the security descriptor of the object is used. Let G and D denote the access rights that are granted and denied, respectively, on the object. Set both to a value of 0 initially.

The following rules are used to determine the authorization for the requester's security context:

1. If the security descriptor has no DACL or its "DACL Present" (DP) bit is not set, then grant the requester all possible access rights on the object.
2. If the DACL does not have any ACE, then grant the requester no access rights on the object.
3. If the SID in the Owner field of the object's security descriptor matches any SID in the requester's security context, then add the bits "Read Control" (RC), "Write DACL" (WD) and "Write Owner" (WO) to G.
4. Evaluate the DACL by examining each ACE in sequence, starting with the first ACE. Perform the following sequence of actions for each ACE in the order as shown. Let the ACCESS_MASK field of the ACE have a value M.
    1. If the "Inherit Only" (IO) flag is set in the ACE, skip the ACE.
    2. If the SID in the ACE does not match any SID in the requester's security context, skip the ACE.

3. If the ACE type is "Access Denied" and the access rights in M are not in G, then add the rights in M to D.

4. If the ACE type is "Access Allowed" and the access rights in M are not in D, then add the rights in M to G.

5. When the end of the DACL is reached, the access rights in G is the maximum standard access available to the requester on the object. Check the requested access mask against the access rights granted in G.

5.1.3.3.3 Checking Object-Specific Access

This section describes how object-specific access rights on Active Directory objects are evaluated, with the exception of access rights representing control access rights and validated write rights. That is the subject of the subsequent sections.

When evaluating object-specific access rights specified in object-specific ACEs for an Active Directory object, the security descriptor of the object (or its parent) is used along with a three-level "object type tree" associated with that object. For an object O that is the subject of an access check, the object type tree T(V, E) consists of nodes V={v_1, v_2, ...}, edges E={e_1, e_2, ...}, and a GUID-valued label for each node in V indicated by *Guid(v)*, and is constructed as follows:

- Let O be an object of class c, and let A={a_1, a_2, ...} be the set of attributes that instances of class c may contain. For each attribute a_i that is an element of A, if a_i.attributeSecurityGUID ≠ NULL, then let p_i denote the property set of which a_i is a member and let Guid(p_i) = a_i.attributeSecurityGUID (see Property Set in section 3.1.1.2). Let P be the union of all such sets {p_i}.
- Add c to V as the root node of the tree and set Guid(c) to c!schemaIDGUID.
- For every property set p_i that is an element of P, add a node p_i to V and Guid(p_i) is as specified earlier.
- For every attribute a_i that is an element of A, add a node a_i to V and set Guid(a_i) to a_i!schemaIDGUID.
- For every property set p_i that is an element of P, add an edge (c, p_i) to E such that p_i is a child of c.
- For every attribute a_i that is an element of A, if there exists a property set p_i that is an element of P of which a_i is a member then add an edge (p_i, a_i) to E such that a_i is a child of p_i; otherwise add an edge (c, a_i) to E such that a_i is a child of c.

Note The object type tree used during an access check can include only a subset of the property set (see Property Set in section 3.1.1.2.3.3) nodes and a subset of the attribute nodes that the requester is interested in. An object type tree for an object is illustrated by the following figure.

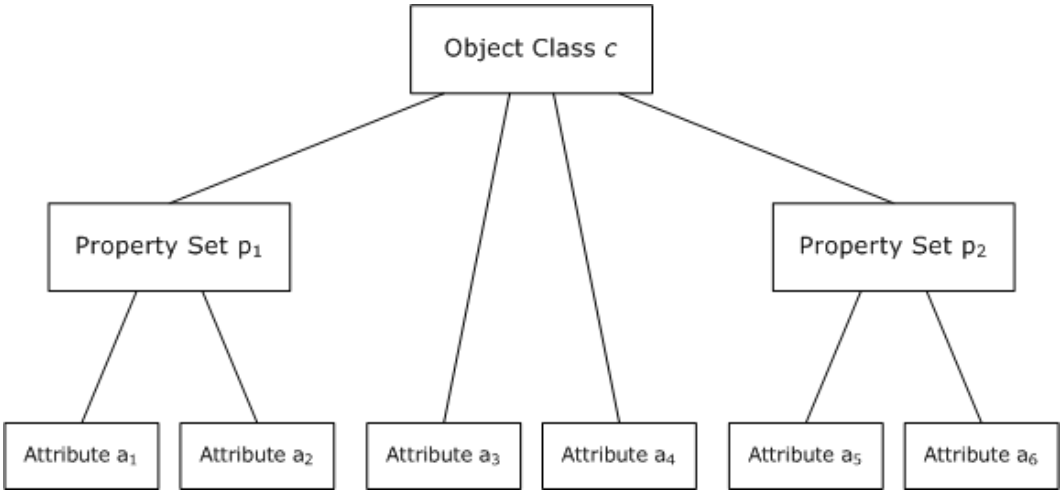


Figure 4: An object type tree

Let r be the root node of the object type tree T. Further, label each node v that is an element of V with two additional labels called Grant(v) and Deny(v) indicating the access rights that are granted and denied, respectively, at that node. Set both labels to a value 0 initially for every node.

The following rules are used to determine the authorization for the requester's security context:

1. If the security descriptor of object O has no DACL or its "DACL Present" (DP) bit is not set, then grant the requester all possible access rights on the object.

2. If the DACL does not have any ACE, then grant the requester no access rights on the object.

3. Evaluate the DACL by examining each ACE in sequence, starting with the first ACE. Perform the following sequence of actions for each ACE in the order as shown. Let the ACCESS_MASK field of the ACE have a value M.

    1. If the "Inherit Only" (IO) flag is set in the ACE, skip the ACE.

    2. If the SID in the ACE does not match any SID in the requester's security context, skip the ACE.

    3. If the ACE type is "Access Allowed" and the access rights in M are not in Deny(r), then add the rights in M to Grant(r) (where r denotes the root node of object type tree T as stated above). For every descendant node u of r, if the rights in M are not in Deny(u), then add the rights in M to Grant(u).

    4. If the ACE type is "Object Access Allowed" and the ObjectType field in the ACE is not present, then treat the ACE type as "Access Allowed" and perform the action in 3.3.

    5. If the ACE type is "Object Access Allowed" and the ObjectType field in the ACE contains a GUID value g:

        If there exists no node v that is an element of V such that Guid(v) = g, then skip the ACE.

        Otherwise, let v that is an element of V be the unique node such that Guid(v) = g. If the rights in M are not in Deny(v), then add the rights in M to Grant(v). For every descendant node u of v, if the rights in M are not in Deny(u), then add the rights in M to Grant(u).

1. If v = r, then proceed to the next ACE.

2. If Grant(v) = Grant(s) for every sibling s of node v, then add the rights in Grant(v) to Grant(p) where p is the parent of node v. Otherwise, proceed to the next ACE.

3. Set v to p, and repeat these three steps.

6. If the ACE type is "Access Denied" and the access rights in M are not in Grant(r), then add the rights in M to Deny(r). For every descendant node u below the root node, if the rights in M are not in Grant(u), then add the rights in M to Deny(u).

7. If the ACE type is "Object Access Denied" and the ObjectType field in the ACE is not present, then treat the ACE type as "Access Denied" and perform the action in 3.6.

8. If the ACE type is "Object Access Denied" and the ObjectType field in the ACE contains a GUID value g:

   If there exists no node v that is an element of V such that Guid(v) = g, then skip the ACE.

   Otherwise, let v be the unique node in P such that Guid(v) = g if any such node exists. If no such node exists, let v be the unique node in A such that Guid(v) = g. If the rights in M are not in Grant(v), then add the rights in M to Deny(v). For every descendant node u of v, if the rights in M are not in Grant(u), then add the rights in M to Deny(u). For every ancestor node w of v, add the rights in M to Deny(w).

4. When the end of the DACL is reached, the access rights in Grant(r) at the root node of tree T is the maximum access available to the requester on the object. For each node u below the root node r, the access rights in Grant(u) is the maximum access available to the requester for that node.

If the requested access is for the entire object, check the requested access mask against the access rights granted in Grant(r). If the requested access is for specific properties on the object, check the requested access mask against the rights granted in Grant(u) where u is the attribute node in tree T that is the target of the request.

5.1.3.3.4 Checking Control Access Right-Based Access

When evaluating the right to perform an operation that is controlled by a control access right identified by the GUID value G, use the following rules to determine the authorization for the requester's security context:

1. If the security descriptor has no DACL or its "DACL Present" (DP) bit is not set, then grant the requester the requested control access right.

2. If the DACL does not have any ACE, then deny the requester the requested control access right.

3. Evaluate the DACL by examining each ACE in sequence, starting with the first ACE. Perform the following sequence of actions for each ACE in the order as shown. Let the ACCESS_MASK field of the ACE have a value M.

   1. If the "Inherit Only" (IO) flag is set in the ACE, skip the ACE.

   2. If the SID in the ACE does not match any SID in the requester's security context, skip the ACE.

3. If the ACE type is "Object Access Allowed", the access right RIGHT_DS_CONTROL_ACCESS (CR) is present in M, and the ObjectType field in the ACE is not present, then grant the requested control access right. Stop any further access checking.

4. If the ACE type is "Object Access Allowed" the access right RIGHT_DS_CONTROL_ACCESS (CR) is present in M, and the ObjectType field in the ACE contains a GUID value equal to G, then grant the requested control access right. Stop any further access checking.

5. If the ACE type is "Object Access Denied", the access right RIGHT_DS_CONTROL_ACCESS (CR) is present in M, and the ObjectType field in the ACE is not present, then deny the requested control access right. Stop any further access checking.

6. If the ACE type is "Object Access Denied" the access right RIGHT_DS_CONTROL_ACCESS (CR) is present in M, and the ObjectType field in the ACE contains a GUID value equal to G, then deny the requested control access right. Stop any further access checking.

5.1.3.3.5 Checking Validated Write-Based Access

When evaluating the right to perform an operation controlled by a validated write access right identified by the GUID value G, use the following rules to determine the authorization for the requester's security context:

1. If the security descriptor has no DACL or its "DACL Present" (DP) bit is not set, then grant the requester the requested validated write right.

2. If the DACL does not have any ACE, then deny the requester the requested validated write right.

3. Evaluate the DACL by examining each ACE in sequence, starting with the first ACE. Perform the following sequence of actions for each ACE in the order as shown. Let the ACCESS_MASK field of the ACE have a value M.

    1. If the "Inherit Only" (IO) flag is set in the ACE, skip the ACE.

    2. If the SID in the ACE does not match any SID in the requester's security context, skip the ACE.

    3. If the ACE type is "Object Access Allowed", the access right RIGHT_DS_WRITE_PROPERTY_EXTENDED (VW) is present in M, and the ObjectType field in the ACE is not present, then grant the requested validated write right. Stop any further access checking.

    4. If the ACE type is "Object Access Allowed" the access right RIGHT_DS_WRITE_PROPERTY_EXTENDED (VW) is present in M, and the ObjectType field in the ACE contains a GUID value equal to G, then grant the requested validated write right. Stop any further access checking.

    5. If the ACE type is "Object Access Denied", the access right RIGHT_DS_WRITE_PROPERTY_EXTENDED (VW) is present in M, and the ObjectType field in the ACE is not present, then deny the requested validated write right. Stop any further access checking.

    6. If the ACE type is "Object Access Denied" the access right RIGHT_DS_WRITE_PROPERTY_EXTENDED (VW) is present in M, and the ObjectType field in the ACE contains a GUID value equal to G, then deny the requested validated write right. Stop any further access checking.

5.1.3.3.6 Checking Object Visibility

An object in Active Directory is considered to be "visible" to a requester if the requester can see the name of the object and thus learn of its existence, even if the requester can see no other attributes of the object. The default behavior of Active Directory with respect to making objects visible to a requesting principal is as follows:

- If a user is granted the RIGHT_DS_LIST_CONTENTS access right on a container, all child objects of that container are visible to the user.
- Otherwise (if a user is not granted the RIGHT_DS_LIST_CONTENTS access right on a container), no child object of that container is visible to the user. This allows the contents of entire containers to be hidden.

However, Active Directory can optionally be put into a special mode, called the "List Object" mode. Active Directory is put into the "List Object" mode by setting the third character of dSHeuristics (section 6.1.1.2.4.1.2) to the value "1". The mode is disabled by setting the same character to the value "0". The default setting is "0".

In "List Object" mode, a requester is allowed to selectively view specific child objects of a container while other child objects remain hidden. In this mode, an object is visible if the user has been granted the RIGHT_DS_LIST_CONTENTS right on the parent object. If, however, the user does not have that right on the parent, then the object is visible if the user is granted the RIGHT_DS_LIST_OBJECT right on both the object and its parent.

In summary, an object is not visible to a requester if:

- The object is not the root object of a NC replica, and
- The requester lacks RIGHT_DS_LIST_CONTENTS right on the object's parent, and
- "List Object" mode is not set (as described above) or the requester lacks the RIGHT_DS_LIST_OBJECT right on both the object and its parent.

5.1.3.4 AD LDS Security Context Construction

The construction of a Windows security context for an authenticated security principal in AD DS is specified in [MS-PAC] section 4.1.2.2.

After a successful authentication to an AD LDS DC, the DC constructs a security context for the authenticated security principal as follows:

1. Create an initial security context.
   - If the bind named an AD LDS user object, the initial security context contains only the objectSid of that object.
   - If the bind named an AD LDS bind proxy, or the SID of some Windows account, the initial security context is the context returned by the Windows login.
2. Extend the security context with well-known SIDs.
   - If the bind named an AD LDS user object or an AD LDS bind proxy object, add the following SIDs to the security context if not already present:
     1. Authenticated Users (section 6.1.1.2.6.2).

2. Everyone (section 6.1.1.2.6.10).

3. Users, for the NC containing the AD LDS object (section 6.1.1.4.13.3).

4. Users, for the config NC of the forest containing the AD LDS object (section 6.1.1.4.13.3).

3. Extend the security context with AD LDS group memberships.

   - If a SID currently in the security context is a member of an AD LDS group on this DC, and that group is not already present in the context, add the SID of that group to the context. (The group membership is represented as a reference to an object whose objectSid equals the SID: either an AD LDS user, an AD LDS bind proxy, an AD LDS group, or a foreignSecurityPrincipal object.) Repeat until there are no more SIDs to add.

6 Additional Information

6.1 Special Objects and Forest Requirements

This section specifies the objects that are necessary for the proper functioning of the DCs in a forest and the requirements that govern the state of these objects.

6.1.1 Special Objects

6.1.1.1 Naming Contexts

References

- Special Attributes: Well-known Objects, Other Well-known Objects, Behavior Version
- Forest Requirements
- FSMO Roles
- State Model: NC Naming
- Security: SD Reference Domain

Glossary Terms: NC, NC Replica, NC root, DC, Forest root, Domain NC, PDC, FSMO

LDAP attributes: instanceType, subRefs, repsTo, repsFrom, replUpToDateVector, wellKnownObjects, otherWellKnownObjects, name, objectClass, nTSecurityDescriptor, fSMORoleOwner, msDS-Behavior-Version, distinguishedName, systemFlags, nTMixedDomain, domainReplica, msDS-AllowedDNSSuffixes, dNSHostName, msDS-AdditionalDnsHostName, msDS-SDReferenceDomain

LDAP classes: configuration, dMD, domainDNS

Constants

- Access mask bits, control access rights: DS-Replication-Get-Changes, DS-Replication-Get-Changes-All, DS-Replication-Get-Changes-In-Filtered-Set
- systemFlags bits: FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE

6.1.1.1.1 Any NC Root

The following attributes have constant semantics across all types of NCs.

instanceType: The instanceType of an NC root is a bit field, which is presented here in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	G	C	A	W	U	H

X: Unused. SHOULD be zero and MUST be ignored.

H (IT_NC_HEAD, 0x00000001): This flag is set (value 1) on all NC roots.

U (IT_UNINSTANT, 0x00000002): If this flag is set, the NC replica that this root represents does not exist locally. This flag implies that this root is a subordinate reference object.

W (IT_WRITE, 0x00000004): This flag is written locally based upon the desired NC replica type. A regular NC replica will have this flag set, and a **partial NC replica** will not have this flag set. The IT_WRITE flag MUST be propagated identically to every object in the NC replica.

A (IT_NC_ABOVE, 0x00000008): This flag indicates that the **local DC** holds an instantiated NC replica that is a parent of the NC replica represented by this NC root. This flag also indicates that this NC root is a subordinate reference object.

C (IT_NC_COMING, 0x00000010): This flag indicates that the NC replica has not completed its initial replication into the local DC, and may not have a full set of objects in the NC represented by this NC root.

G (IT_NC_GOING, 0x00000020): This flag indicates that the NC replica is being removed from the local DC, and may not have a full set of objects in the NC represented by this NC root.

Requirements:

- IT_UNINSTANT can only be set with IT_NC_HEAD and IT_NC_ABOVE. The remaining bits are incompatible with IT_UNINSTANT.
- IT_NC_COMING and IT_NC_GOING cannot be set at the same time.
- If IT_NC_GOING is set, then no replication can occur with that NC, either as server or as client.

subRefs: This value references all child objects in this NC replica of this NC root that are, themselves, NC roots. For example, the schema NC is always referenced by this value on the Config NC root object.

repsTo: This attribute contains the abstract attribute repsTo that is associated with this DC for this NC replica. This attribute is nonreplicated. [MS-DRSR] section 5.170 specifies this abstract attribute.

repsFrom: This attribute contains the abstract attribute repsFrom that is associated with this DC for this NC replica. This attribute is nonreplicated. [MS-DRSR] section 5.169 specifies this abstract attribute.

replUpToDateVector: This attribute contains the abstract attribute replUpToDateVector that is associated with this DC for this NC replica. This attribute is nonreplicated. [MS-DRSR] section 5.165 specifies this abstract attribute.

6.1.1.1.2 Config NC Root

name: Configuration

parent: For AD DS, the forest root NC root object. For AD LDS, no parent.

objectClass: configuration

wellKnownObjects: This attribute holds DN-Binary values. See section 6.1.4 for details.

instanceType: This value can never contain the following flags:

- IT_NC_COMING
- IT_NC_GOING

- IT_UNINSTANT

nTSecurityDescriptor:

- Let D1 be a DC that is instructed to host a writable replica of the config NC (see section 6.1.2.3 for hosting requirements). In order for D1 to replicate the config NC, D1 MUST be granted the following rights on the config NC root:
  - DS-Replication-Get-Changes
  - DS-Replication-Get-Changes-All
  - DS-Replication-Get-Changes-In-Filtered-Set
- Let D2 be a DC that is instructed to host a read-only replica of config NC (see section 6.1.2.3 for hosting requirements) such that the objects in the NC replica will not contain attributes in the filtered attribute set. In order for D2 to replicate the config NC, D2 MUST be granted the following rights on the config NC root:
  - DS-Replication-Get-Changes

msDS-ReplAuthenticationMode: Present and used on AD LDS only. Specifies the authentication that is used for DC-to-DC communication over RPC ([MS-DRSR]). The msDS-ReplAuthenticationMode values 0, 1, and 2 are valid; if absent, the effect is as if the value was 1. See [MS-DRSR] section 2.2.1 for the effects of these values.

objectSid: Present and used on AD LDS only. This attribute contains the SID that is used in generating objectSid values for new AD LDS security principals residing in the config NC, as specified in section 3.1.1.5.2.4. This attribute is not returned by LDAP queries.

6.1.1.1.3 Schema NC Root

name: Schema

parent: Config NC root

objectClass: dMD

fSMORoleOwner: This value refers to the nTDSDSA object of the DC that owns the Schema Master FSMO. See section 6.1.5.

instanceType: This value can never contain the following flags:

- IT_NC_COMING
- IT_NC_GOING
- IT_UNINSTANT

nTSecurityDescriptor: Let D be a DC that is instructed to host the schema replica NC (see section 6.1.2.3 for hosting requirements). In order for D to replicate the schema NC, D must be granted the following rights on the schema NC root:

- DS-Replication-Get-Changes
- DS-Replication-Get-Changes-All
- DS-Replication-Get-Changes-In-Filtered-Set

6.1.1.1.4 Domain NC Root

distinguishedName: See section 3.1.1.1 for more information about domain NC naming rules.

objectClass: domainDNS

fSMORoleOwner: This value refers to the nTDSDSA object of the DC that owns the PDC FSMO role. See section 6.1.5 for more information about the PDC role.

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

wellKnownObjects: This attribute holds DN-Binary values. See section 6.1.4 for details.

otherWellKnownObjects: This attribute holds DN-Binary values. See section 6.1.4 for details.

msDS-Behavior-Version: This value defines the functional level of the domain. See section 6.1.4.

nTMixedDomain: This value defines whether NT BDC replication [MS-NRPC] is available in the domain. See section 6.1.4.1.

domainReplica: See section 3.1.1.5 for more information.

msDS-AllowedDNSSuffixes: List of DNS suffixes that are allowed in the dNSHostName and msDS-AdditionalDnsHostName attributes of computer objects in this domain.

nTSecurityDescriptor:

- Let D1 be a DC that is instructed to host a writable domain replica NC (see section 6.1.2.3 for hosting requirements). In order for D1 to replicate the domain NC, D1 must be granted the following rights on the domain NC root:
  - DS-Replication-Get-Changes
  - DS-Replication-Get-Changes-All
  - DS-Replication-Get-Changes-In-Filtered-Set
- Let D2 be a DC that is instructed to host a partial or read-only domain replica NC (see section 6.1.2.3 for hosting requirements) such that objects in the NC replica can have attributes in the filtered attribute set. In order for D2 to replicate the domain NC, D2 must be granted the following right on the domain NC root:
  - DS-Replication-Get-Changes
  - DS-Replication-Get-Changes-In-Filtered-Set
- Let D3 be a DC that is instructed to host a partial or read-only domain replica NC (see section 6.1.2.3 for hosting requirements) such that objects in the NC replica will not have attributes in the filtered attribute set. In order for D3 to replicate the domain NC, D3 must be granted the following right on the domain NC root:
  - DS-Replication-Get-Changes

msDS-EnabledFeature: This value references the objects that represent optional features that are enabled in the domain. See section 3.1.1.9.

6.1.1.1.5 Application NC Root

distinguishedName: See section 3.1.1.1 for more information about domain NC naming rules.

objectClass: domainDNS (AD DS); any structural or 88 class except dMD and configuration (AD LDS)

wellKnownObjects: This attribute holds DN-Binary values. See section 6.1.4 for details.

otherWellKnownObjects: This attribute holds DN-Binary values. See section 6.1.4 for details.

nTSecurityDescriptor:

- Let D1 be a DC that is instructed to host a writable application replica NC (see section 6.1.2.3 for hosting requirements). In order for D1 to replicate the NC, D1 must be granted the following rights on the NC root:
  - DS-Replication-Get-Changes
  - DS-Replication-Get-Changes-All
  - DS-Replication-Get-Changes-In-Filtered-Set
- Let D2 be a DC that is instructed to host a read-only application replica NC (see section 6.1.2.3 for hosting requirements) such that objects in the NC replica will not contain attributes in the filtered attribute set. In order for D2 to replicate the NC, D2 must be granted the following rights on the NC root:
  - DS-Replication-Get-Changes
- Note that this nTSecurityDescriptor must be resolved with the domain specified on the msDS-SDReferenceDomain attribute on the crossRef object representing this NC; see section 5 for details.

objectSid: Present and used on AD LDS only. This attribute contains the SID that is used in generating objectSid values for new AD LDS security principals residing in this application NC, as specified in section 3.1.1.5.2.4. This attribute is not returned by LDAP queries.

6.1.1.2 Configuration Objects

References

- FSMO Roles
- LDAP
- Special Attributes
- Forest Requirements
- Security
- Knowledge Consistency Checker
- Originating Updates

Glossary Terms: NC, NC Replica, NC root, DC, Domain NC, FSMO, Forest Functional Level, Application NC, KCC, ISTG, Intra-site, Inter-site, Global Catalog, Forest, SMTP, Site, COM (Component Object Model), UUID, MAPI, ANR, NSPI

LDAP attributes: name, objectClass, fSMORoleOwner, msDS-Behavior-Version, msDS-EnabledFeature, distinguishedName, systemFlags, nTMixedDomain, dnsRoot, nCName, msDS-Replication-Notify-First-DSA-Delay, msDS-Replication-Notify-Subsequent-DSA-Delay, nETBIOSName, msDS-SDReferenceDomain, options, schedule, interSiteTopologyGenerator, interSiteTopologyFailover, interSiteTopologyRenew, serverReference, dNSHostName, mailAddress, invocationId, hasMasterNCs, hasPartialReplicaNCs, msDS-HasInstantiatedNCs, instanceType, msDS-OptionalFeatureGUID, msDS-RequiredForestBehaviorVersion, msDS-OptionalFeatureFlags, msDS-HasDomainNCs, msDS-hasMasterNCs, msDS-ReplicationEpoch, enabledConnection, fromServer, transportType, mS-DS-ReplicatesNCReason, siteObject, transportDLLName, transportAddressAttribute, cost, siteList, replInterval, siteLinkList, adminPropertyPages, shellPropertyPages, adminContextMenu, shellContextMenu, adminMultiselectPropertyPages, treatAsLeaf, creationWizard, createWizardExt, dSHeuristics, objectGUID, msDS-KeyVersionNumber, msDS-DeletedObjectLifetime, tombstoneLifetime, sPNMappings, msDS-Other-Settings, rightsGuid, appliesTo, localizationDisplayId, validAccesses, repsTo

LDAP classes: crossRefContainer, crossRef, sitesContainer, site, nTDSSiteSettings, nTDSConnection, serversContainer, server, nTDSDSA, subnetContainer, subnet, interSiteTransportContainer, interSiteTransport, siteLink, container, displaySpecifier, nTDSService, physicalLocation, controlAccessRight

Constants

- systemFlags bits: FLAG_DISALLOW_DELETE, FLAG_CR_NTDS_NC, FLAG_CR_NTDS_DOMAIN, FLAG_CR_NTDS_NOT_GC_REPLICATED, FLAG_DISALLOW_MOVE_ON_DELETE, FLAG_CONFIG_ALLOW_LIMITED_MOVE, FLAG_CONFIG_ALLOW_RENAME
- Replication bits: DRS_SYNC_FORCED

6.1.1.2.1 Cross-Ref-Container Container

name: Partitions

parent: Config NC root

objectClass: crossRefContainer

fSMORoleOwner: This value references the Domain Naming Master FSMO role owner. See section 6.1.5.

systemFlags: {FLAG_DISALLOW_DELETE}

msDS-Behavior-Version: This value defines the forest functional level. See section 6.1.4.

msDS-EnabledFeature: This value references the objects that represent optional features that are enabled in the forest. See section 3.1.1.9.

6.1.1.2.1.1 Cross-Ref Objects

The following is the description of the flags and their meaning for crossRef objects stored in systemFlags. The flags are presented in big-endian byte order.

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

X: Unused. Must be zero and ignored.

NC (FLAG_CR_NTDS_NC, 0x00000001): NC exists within the forest (not external).

D (FLAG_CR_NTDS_DOMAIN, 0x00000002): NC is a domain.

GC (FLAG_CR_NTDS_NOT_GC_REPLICATED, 0x00000004): NC must not be replicated to GC servers as a read-only replica.

The following attributes and attribute values are common to crossRef objects representing all NC types:

parent: crossRefContainer object

objectClass: crossRef

Enabled: If false, this is a "pre-created" crossRef; that is, the crossRef exists, but the corresponding NC root does not yet exist. See section 3.1.1.5.2.8.

dnsRoot: If Enabled equals false, in AD DS dnsRoot holds the DNS name of the DC that will create the root of this NC. If Enabled equals false, in AD LDS, dnsRoot holds the DNS name of the DC that will create the root of this NC followed by a colon (":"), followed by the LDAP port number used by the DC, followed by another colon (":"), followed by the SSL port number used by the DC. If Enabled is not false, in AD DS dnsRoot holds the fully qualified DNS name used for LDAP referrals (section 3.1.1.4.6). If Enabled is not false, in AD LDS dnsRoot is absent.

nCName: If Enabled is not false, a reference to the NC root corresponding to this crossRef.

msDS-Replication-Notify-First-DSA-Delay: Indicates the number of seconds that each DC should delay after receiving updates (originating or replicated) to objects in the NC referred to by nCName before the DC notifies another DC of updates received according to the DCs local repsTos. See IDL_DRSReplicaSync in [MS-DRSR] section 4.1.23.

msDS-Replication-Notify-Subsequent-DSA-Delay: Indicates the number of seconds that each DC should delay after notifying the first DC of updates received to objects in the NC referred to by nCName before notifying each additional DC according to the DCs local repsTos. See IDL_DRSReplicaSync in [MS-DRSR] section 4.1.23.

6.1.1.2.1.1.1 Foreign crossRef Objects

A foreign crossRef object is used to enable referrals for searches that need to return objects from different forests or LDAP services. For more information, see section 3.1.1.3. The following attribute and attribute values are defined for a foreign crossRef:

systemFlags: 0

6.1.1.2.1.1.2 Configuration crossRef Object

name: Enterprise Configuration

systemFlags: { FLAG_CR_NTDS_NC }

nCName: The value must equal the config NC root.

dnsRoot: In AD DS, the value is the forest root's fully qualified DNS name. Not present in AD LDS.

6.1.1.2.1.1.3 Schema crossRef Object

name: Enterprise Schema

systemFlags: { FLAG_CR_NTDS_NC }

nCName: The value must equal the schema NC root.

dnsRoot: In AD DS, the value is the forest root's fully qualified DNS name. Not present in AD LDS.

6.1.1.2.1.1.4 Domain crossRef Object

The following attribute and attribute values are common to domain crossRef objects:

name: The NetBIOS name of the domain.

nCName: The reference must be to a domain NC root.

nETBIOSName: This value is the NetBIOS name of the domain.

trustParent: This attribute is not present on the root domain NC's crossRef object. For child NCs, this value references the parent NC's crossRef object. For a domain NC that is not the root and does not have a parent NC, this value references the root domain's crossRef object.

nTMixedDomain: This value is read-only on this object. It is kept in sync with the same attribute on the NC root of the NC referred to by nCName. See section 6.1.4.1.

systemFlags: { FLAG_CR_NTDS_NC | FLAG_CR_NTDS_DOMAIN }

msDS-Behavior-Version: This value is read-only on this object. It is kept in sync with the same attribute on the NC root of the NC referred to by nCName. See section 6.1.4.

6.1.1.2.1.1.5 Application NC crossRef Object

dnsRoot: In AD DS, the value for dnsRoot for an application NC crossRef is derived by syntactically converting the DN portion of the crossRef's nCName into a fully qualified DNS name as specified in section 3.1.1.1.5. Not present in AD LDS.

systemFlags: { FLAG_CR_NTDS_NC | FLAG_CR_NTDS_NOT_GC_REPLICATED }

msDS-NC-Replica-Locations: This attribute references the nTDSDSA objects representing every DC instructed to hold a writable NC replica of this application NC. See Hosting Requirements in section 6.1.2.3.

msDS-SDReferenceDomain: In AD DS, the attribute references an NC root object for a domain. All security descriptors in this application NC must use the NC represented as the reference domain for resolution. See section 5 for security descriptor reference domain information. Not present in AD LDS.

msDS-NC-RO-Replica-Locations: This attribute references the nTDSDSA object representing every DC instructed to hold a read-only NC replica of this application NC. See Hosting Requirements in section 6.1.2.3.

6.1.1.2.2 Sites Container

Each forest contains a Sites container in the Config NC. For each site in the forest, a site object exists in the Sites container.

name: Sites

parent: Config NC root object

objectClass: sitesContainer

systemFlags: { FLAG_DISALLOW_DELETE | FLAG_DISALLOW_MOVE_ON_DELETE }

6.1.1.2.2.1 Site Object

A site object corresponds to a set of one or more IP subnets that have LAN connectivity. Thus, by virtue of their subnet associations, DCs that are in the same site are well connected in terms of speed. Each site object has a child nTDSSiteSettings object and a Servers container.

parent: Sites container

objectClass: site

systemFlags: { FLAG_CONFIG_ALLOW_RENAME | FLAG_DISALLOW_MOVE_ON_DELETE }

Note The initial AD DS and AD LDS configuration contains one initial site object named Default-First-Site-Name, which has no subnet association.

6.1.1.2.2.1.1 NTDS Site Settings Object

NTDS **site settings objects** identify site-wide settings. There is one nTDSSiteSettings object per site.

name: NTDS Site Settings

parent: site object

objectClass: nTDSSiteSettings

options: One or more bits from the following diagram. The bits are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	RSE	SHE	BHD	W2K	FWB	GCE	ISD	DSD	MHD	TCD	ATD

X: Unused. Must be zero and ignored.

ATD (NTDSSETTINGS_OPT_IS_AUTO_TOPOLOGY_DISABLED, 0x00000001): Automatic topology generation is disabled. See section 6.2 for more information.

TCD (NTDSSETTINGS_OPT_IS_TOPL_CLEANUP_DISABLED, 0x00000002): Automatic topology cleanup is disabled. See section 6.2 for more information.

MHD (NTDSSETTINGS_OPT_IS_TOPL_MIN_HOPS_DISABLED, 0x00000004): Automatic minimum hops topology is disabled. See section 6.2 for more information.

DSD (NTDSSETTINGS_OPT_IS_TOPL_DETECT_STALE_DISABLED, 0x00000008): Automatic stale server detection is disabled. See section 6.2 for more information.

ISD (NTDSSETTINGS_OPT_IS_INTER_SITE_AUTO_TOPOLOGY_DISABLED, 0x00000010): Automatic intersite topology generation is disabled. See section 6.2 for more information.

GCE (NTDSSETTINGS_OPT_IS_GROUP_CACHING_ENABLED, 0x00000020): Caching of users' group memberships is enabled for this site. This caching is an implementation-specific behavior. This flag may be ignored by other implementations but must not be used in a conflicting way that would affect the performance of Windows DCs.

FWB (NTDSSETTINGS_OPT_FORCE_KCC_WHISTLER_BEHAVIOR, 0x00000040): Force the KCC to behave in a forest functional level of DS_BEHAVIOR_WIN2003 or greater. See section 6.2 for more information.

W2K (NTDSSETTINGS_OPT_FORCE_KCC_W2K_ELECTION, 0x00000080): Force the KCC to use the Windows 2000 operating system **intersite topology generator (ISTG)** election algorithm. See section 6.2 for more information.

BHD (NTDSSETTINGS_OPT_IS_RAND_BH_SELECTION_DISABLED, 0x00000100): Prevent the KCC from randomly picking a bridgehead when creating a connection. See section 6.2 for more information.

SHE (NTDSSETTINGS_OPT_IS_SCHEDULE_HASHING_ENABLED, 0x00000200): Allow the KCC to use hashing when creating a replication schedule. See section 6.2 for more information.

RSE (NTDSSETTINGS_OPT_IS_REDUNDANT_SERVER_TOPOLOGY_ENABLED, 0x00000400): Create static failover connections. See section 6.2 for more information.

schedule: The default replication schedule (defined as a SCHEDULE structure) that applies to all nTDSConnection objects for intrasite replication within this site. If this attribute does not contain any value, a schedule of once per hour is applied to replication within this site. See section 6.2 for more information.

interSiteTopologyGenerator: A reference to the nTDSDSA object of the DC that is acting as the ISTG for this site. See section 6.2 for more information on the ISTG.

interSiteTopologyFailover: Indicates how much time must transpire since the last keep-alive for the ISTG to be considered dead. See section 6.2 for more information.

interSiteTopologyRenew: Indicates how often the intersite topology generator updates the keep-alive message that is sent to domain controllers contained in the same site. See section 6.2 for more information.

6.1.1.2.2.1.2 Servers Container

Each site contains a Servers container that contains the server objects for all the DCs that are in that site.

parent: site object

objectClass: serversContainer

systemFlags: FLAG_DISALLOW_MOVE_ON_DELETE

6.1.1.2.2.1.2.1 Server Object

Each DC in a domain has a server object in the config NC. See requirements in section 6.1.2.1. A server object has the following attributes:

parent: The parent of this object is a serversContainer object.

objectClass: server

systemFlags: { FLAG_CONFIG_ALLOW_RENAME | FLAG_CONFIG_ALLOW_LIMITED_MOVE | FLAG_DISALLOW_MOVE_ON_DELETE }

serverReference: In AD DS, a reference to the domain controller object representing this DC. See requirements in section 6.1.2.1. Not present in AD LDS.

dNSHostName: Fully qualified DNS name of the DC.

mailAddress: To enable the DC to perform intersite replication via the SMTP protocol (see [MS-SRPL]), this attribute must contain the SMTP mail address of the server.

6.1.1.2.2.1.2.1.1 nTDSDSA Object

Each DC in a forest has an nTDSDSA object in the config NC. See requirements in section 6.1.2.1. An nTDSDSA object has the following attributes:

name: NTDS Settings

parent: An object with objectClass server.

objectClass: nTDSDSA

dMDLocation: The **DSName** of the schema NC root.

invocationId: The invocationId for this DC (section 3.1.1.1.9).

options: One or more of the following bits presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	DS	DNX	DO	DI	GC

X: Unused. Must be zero and ignored.

GC (NTDSDSA_OPT_IS_GC, 0x00000001): This DC is, or is becoming, a GC server.

DI (NTDSDSA_OPT_DISABLE_INBOUND_REPL, 0x00000002): This DC does not perform inbound replication unless the DRS_SYNC_FORCED flag is passed. See [MS-DRSR] section 4.1.10.4.1, ReplicateNCRequestMsg, for the effects of this option.

DO (NTDSDSA_OPT_DISABLE_OUTBOUND_REPL, 0x00000004): This DC does not perform outbound replication unless the DRS_SYNC_FORCED flag is passed. See [MS-DRSR] section 4.1.10.5.2, GetReplChanges, for the effects of this option.

DNX (NTDSDSA_OPT_DISABLE_NTDSCONN_XLATE, 0x00000008): This DC does not translate connection objects into repsFroms. See section 6.2 for more information.

DS (NTDSDSA_OPT_DISABLE_SPN_REGISTRATION, 0x00000010): This DC does not perform SPN registration. Only interpreted by AD LDS DCs. See [MS-DRSR] sections 2.2.3.3 and 2.2.4.3, SPN for a Target DC in AD LDS, for the effects of this option.

systemFlags: {FLAG_DISALLOW_MOVE_ON_DELETE}

msDS-Behavior-Version: Indicates the DC version. See section 6.1.4.2 for more information.

msDS-PortLDAP: In AD LDS, stores the LDAP port for this instance. Not present in AD DS.

msDS-PortSSL: In AD LDS, stores the SSL port for this instance. Not present in AD DS.

msDS-ServiceAccount: In AD LDS, stores the foreignSecurityPrincipal object that represents the service account running this DC. Not present in AD DS.

hasMasterNCs: Contains the **DSName** of the NC root objects representing the schema NC, config NC, and domain NC for the default domain of the DC. This attribute always contains these three values and only these three values. This attribute is not present on the nTDSDSA object of an RODC.

hasPartialReplicaNCs: Contains the **DSName** of the root objects of all domain NCs within the forest for which the DC hosts a partial NC replica.

msDS-HasInstantiatedNCs: Contains an Object(DN-Binary) value for each NC replica that is hosted by this DC. The DN field is the DN of the root object of the **NC**. The Binary field contains the value of the instanceType attribute on the root object of the NC. This is a binary encoding of attribute instanceType with little-endian byte ordering.

Requirement: The DN fields of all the values of msDS-HasInstantiatedNCs must be equal to the set of DNs contained in the values of msDS-hasMasterNCs and hasPartialReplicaNCs.

msDS-HasDomainNCs: Equals the **DSName** of the NC root object for which the DC is hosting a regular NC replica. This attribute must have only one value. This NC root is called the default domain for the DC.

msDS-hasMasterNCs: Contains the **DSNames** of the root objects of all writable NC replicas hosted by this DC. Not present on the nTDSDSA object of an RODC. On a normal (writable) DC, includes the default NC, config NC, schema NC, and all application NC replicas hosted by the DC.

msDS-hasFullReplicaNCs: Contains the **DSNames** of the root objects of all **read-only full NC replicas** hosted by this DC. Not present on the nTDSDSA object of a normal (writable) DC. On an RODC, includes the default NC, config NC, schema NC, and all application NC replicas hosted by the DC.

msDS-ReplicationEpoch: [MS-DRSR] section 4.1.3.1 (client behavior of IDL_DRSBind) and [MS-DRSR] section 4.1.10.5 (server behavior of IDL_DRSGetNCChanges) specify the effects of this attribute.

msDS-DefaultNamingContext: In AD LDS, specifies the NC that should be returned as the default NC by the defaultNamingContext attribute of the root DSE. If this attribute is not set, AD LDS does not have a default NC and the defaultNamingContext attribute of the root DSE is treated by the server as if it does not exist. Not present in AD DS.

objectCategory: This attribute is a mandatory attribute representing the schema definition of the nTDSDSA object. If the objectCategory points to the classSchema object for the nTDSDSA class, then this nTDSDSA object is for a normal (writable) DC. If the objectCategory points to the classSchema object for the nTDSDSARO class, then this nTDSDSA object is for an RODC.

msDS-EnabledFeature: This value references the objects that represent optional features that are enabled in the DC. See section 3.1.1.9.

6.1.1.2.2.1.2.1.2 Connection Object

An nTDSConnection object represents a path for replication from a source DC to a destination DC. This object is a child of the nTDSDSA object of the destination DC. See section 6.2 for more information about connection objects.

Each nTDSConnection object has the following attributes:

parent: nTDSDSA object

objectClass: nTDSConnection

enabledConnection: Indicates whether the connection can be used for replication.

fromServer: A reference to the nTDSDSA object of the source DC.

schedule: Contains a SCHEDULE structure specifying the time intervals when replication can be performed between the source and the destination DCs. In case of intrasite replication (source and destination DCs are in the same site), the value of this attribute is derived from the schedule attribute on the nTDSSiteSettings object of the site where the two DCs reside. In case of intersite replication (source and destination DCs are in different sites), the value is derived from the schedule attribute on the siteLink object that links the two sites.

systemFlags: {FLAG_CONFIG_ALLOW_RENAME | FLAG_CONFIG_ALLOW_MOVE}

options: One or more bits from the following diagram. The bits are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	UOS	DIC	UN	OND	TS	IG

X: Unused. Must be zero and ignored.

IG (NTDSCONN_OPT_IS_GENERATED, 0x00000001): The nTDSConnection object was generated by the KCC. See section 6.2 for more information.

TS (NTDSCONN_OPT_TWOWAY_SYNC, 0x00000002): Indicates that a replication cycle should be performed in the opposite direction at the end of a replication cycle that is using this connection.

OND (NTDSCONN_OPT_OVERRIDE_NOTIFY_DEFAULT, 0x00000004): Do not use defaults to determine notification.

UN (NTDSCONN_OPT_USE_NOTIFY, 0x00000008): The source DC notifies the destination DC regarding changes on the source DC.

DIC (NTDSCONN_OPT_DISABLE_INTERSITE_COMPRESSION, 0x00000010): For intersite replication, this indicates that the compression of replication data is disabled.

UOS (NTDSCONN_OPT_USER_OWNED_SCHEDULE, 0x00000020): For KCC-generated connections, indicates that the schedule attribute is owned by the user and should not be modified by the KCC. See section 6.2 for more information.

transportType: A reference to the interSiteTransport object for the transport used on this connection. For more information about physical transport types, see [MS-SRPL].

mS-DS-ReplicatesNCReason: For each NC that is replicated using this connection, this attribute contains an Object(DN-Binary) value, where the DN portion is the DN of the NC, and the binary value is a 32-bit–wide bit field. The binary portion contains extended information about a connection object that could be used by administrators. It consists of one or more bits from the following diagram. The bits are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	RS	SIF	SF	IS	ISG	OC	SS	MH	R	GC

X: Unused. Must be zero and ignored.

GC (NTDSCONN_KCC_GC_TOPOLOGY, 0x00000001): Not used.

R (NTDSCONN_KCC_RING_TOPOLOGY, 0x00000002): The connection object is created to form a ring topology.

MH (NTDSCONN_KCC_MINIMIZE_HOPS_TOPOLOGY, 0x00000004): The connection object is created to minimize hops between replicating nodes.

SS (NTDSCONN_KCC_STALE_SERVERS_TOPOLOGY, 0x00000008): If the KCC finds that the destination server is not responding, then it sets this bit.

OC (NTDSCONN_KCC_OSCILLATING_CONNECTION_TOPOLOGY, 0x00000010): The KCC sets this bit if deletion of the connection object was prevented.

When the KCC considers deleting a connection object, it first checks if it previously deleted connection objects with the same source DC, destination DC, and options for an

implementation-specific number of times *T* (default value is 3) over the last implementation-specific time period *t* (the default is 7 days) since the server has started. If it did, it will set the **NTDSCONN_KCC_OSCILLATING_CONNECTION_TOPOLOGY** bit on the connection object and will not delete it. Otherwise, it will delete the connection object.

ISG (NTDSCONN_KCC_INTERSITE_GC_TOPOLOGY, 0x00000020): This connection is to enable replication of partial NC replica between DCs in different sites.

IS (NTDSCONN_KCC_INTERSITE_TOPOLOGY, 0x00000040): This connection is to enable replication of a full NC replica between DCs in different sites.

SF (NTDSCONN_KCC_SERVER_FAILOVER_TOPOLOGY, 0x00000080): This connection is a redundant connection between DCs that is used for failover when other connections between DCs are not functioning.

SIF (NTDSCONN_KCC_SITE_FAILOVER_TOPOLOGY, 0x00000100): This connection is a redundant connection between **bridgehead DCs** in different DCs; it is used for failover when other connections between bridgehead DCs connecting two sites are not functioning.

RS (NTDSCONN_KCC_REDUNDANT_SERVER_TOPOLOGY, 0x00000200): Redundant connection object connecting bridgeheads in different sites.

The connection object is for server-to-server replication implementation only. Peer DCs MAY assign a meaning to it, but it is not required for interoperation with Windows clients.

See section 6.2 for more information about these options.

6.1.1.2.2.1.2.1.3 RODC NTFRS Connection Object

An RODC NTFRS connection object exists for each RODC in the forest. RODC NTFRS connection objects do not exist for writable DCs. An RODC NTFRS connection object represents a path for **File Replication Service (FRS)** replication [MS-FRS1] from a source DC to a destination DC; it is not used for directory replication service (DRS) replication [MS-DRSR]. This object is a child of the nTDSDSA object of the destination RODC. See section 6.2 for more information about connection objects.

Each RODC NTFRS connection object has the following attributes:

name: RODC Connection (SYSVOL)

Note On Windows Server 2008 operating system and Windows Server 2008 R2 operating system, the name attribute was set to "RODC Connection (FRS)".

parent: nTDSDSA object

objectClass: nTDSConnection

enabledConnection: true

fromServer: A reference to the nTDSDSA object of the source DC.

schedule: Contains a SCHEDULE structure that specifies the time intervals when replication can be performed between the source and the destination DCs. See section 6.2.2.7 for more information about how this value is derived.

systemFlags: {FLAG_CONFIG_ALLOW_RENAME}

options: Both of the bits from the following diagram. The bits are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	RT	X	X	X	X	X	IG

X: Unused. Must be zero and ignored.

IG (NTDSCONN_OPT_IS_GENERATED, 0x00000001): The nTDSConnection object was generated by the system, not by a user or administrator directly.

RT (NTDSCONN_OPT_RODC_TOPOLOGY, 0x00000040): The NTDSCONN_OPT_RODC_TOPOLOGY bit in the options attribute indicates whether the connection can be used for DRS replication [MS-DRSR]. When set, the connection should be ignored by DRS replication and used only by FRS replication [MS-FRS1]. See section 6.2 and [MS-FRS1] section 3.1.1.8.

6.1.1.2.2.2 Subnets Container

Each forest contains a Subnets container in the config NC. A network subnet is a segment of a TCP/IP network to which a set of logical IP addresses is assigned. For each subnet in the forest, a subnet object exists in the Subnets container.

name: Subnets

parent: Sites container

objectClass: subnetContainer

systemFlags: FLAG_DISALLOW_DELETE

6.1.1.2.2.2.1 Subnet Object

subnet objects define network subnets in the directory. Subnets group computers in a way that identifies their physical proximity on the network. subnet objects are used to map computers to sites.

- name: The name of the subnet object identifies the set of IP addresses that fall in this subnet. An IP address that falls in this subnet is considered to be in the site specified by the siteObject attribute of this object.

  A valid subnet name must satisfy the following constraints:

  Let s be the subnet name.

  Let l be the length of the subnet name.

  Let BitMask[] = {0x00000000, 0x00000080, 0x000000C0, 0x000000E0, 0x000000F0, 0x000000F8, 0x000000FC, 0x000000FE, 0x000000FF, 0x000080FF, 0x0000C0FF, 0x0000E0FF, 0x0000F0FF, 0x0000F8FF, 0x0000FCFF, 0x0000FEFF, 0x0000FFFF, 0x0080FFFF, 0x00C0FFFF, 0x00E0FFFF, 0x00F0FFFF, 0x00F8FFFF, 0x00FCFFFF, 0x00FEFFFF, 0x00FFFFFF, 0x80FFFFFF, 0xC0FFFFFF, 0xE0FFFFFF, 0xF0FFFFFF, 0xF8FFFFFF, 0xFCFFFFFF, 0xFEFFFFFF, 0xFFFFFFFF };

s is a valid subnet name if:

1. There is only one occurrence of the character "/" in s. Let i be the index of the character "/" in s.

2. The substring s[0, i-1] does not have any leading zeros and is either a valid IPv4 address in dotted decimal notation (as specified in [RFC1166]) or a valid IPv6 address in colon-hexadecimal form or compressed form (as specified in [RFC2373]).

Let b be the binary representation of the address.

3. The substring s[i+1, l-1] does not have any leading zeros and can be converted to an unsigned integer n.

4. When the address is in IPv4 format, 0 < n <= 32. When the address is in IPv6 format, 0 < n <= 128.

5. When the address is in IPv4 format, b & (~BitMask[n]) = 0.

6. When the address is in IPv4 format, b ≠ BitMask[n].

Based on the subnet object name, the range of the IP addresses that the subnet contains can be determined. For example, if the IPv4 subnet object name is 10.121.0.0/22, then according the above definition, b will be 00001010.01111001.00000000.00000000 and n will be 22. This means that the first 22 bits of b will be fixed for the range of the IP addresses the subnet contains. Then the IP address range of the subnet is from 00001010.01111001.00000000.00000000 to 00001010.01111001.00000011.11111111, namely from 10.121.0.0 to 10.121.3.255. Similarly, an IPv6 subnet object name 2001:DA8::/48 represents the IPv6 addresses from 2001:DA8:0:0:0:0:0:0 to 2001:DA8:0:FFFF:FFFF:FFFF:FFFF:FFFF.

- parent: Subnets container
- objectClass: subnet
- systemFlags: FLAG_CONFIG_ALLOW_RENAME
- siteObject: The **DSName** of the site object for the site that covers this subnet.

6.1.1.2.2.3 Inter-Site Transports Container

The Inter-Site Transports container provides the means for specifying the transport or wire protocol to be used for replication between sites. Intersite replication can use either the RPC protocol over IP (see [MS-DRSR]), or the SMTP protocol (see [MS-SRPL]).

name: Inter-Site Transports

parent: Sites container

objectClass: interSiteTransportContainer

systemFlags: FLAG_DISALLOW_DELETE

6.1.1.2.2.3.1 IP Transport Container

The IP Transport container contains all the siteLink and siteLinkBridge objects that connect two or more sites for intersite replication using RPC over IP protocol.

parent: Inter-Site Transports container

objectClass: interSiteTransport

transportDLLName: The value of this attribute MUST be the string "ismip.dll".

transportAddressAttribute: Identifies which attribute on the server object of a DC should be used as the network address of the DC for replication using this transport. For the IP transport, the attribute is dNSHostName.

options: A set of the following bit flags presented in big-endian byte order. For IP transport, the initial value is none present (options value 0x0).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	BR	IS

X: Unused. Must be zero and ignored.

IS (NTDSTRANSPORT_OPT_IGNORE_SCHEDULES, 0x00000001): If present, values of the schedule attribute for siteLink objects associated with this transport are ignored. In this case, the schedule is assumed to be always "on"; that is, that the transport is always available to send and receive messages.

BR (NTDSTRANSPORT_OPT_BRIDGES_REQUIRED, 0x00000002): If present, transitive connectivity between siteLink objects associated with this transport is assumed only if the siteLink objects are in the siteLinkList of the same siteLinkBridge object. If absent, the system behaves as if all siteLink objects associated with this transport were in the siteLinkList of a common siteLinkBridge object associated with this transport.

6.1.1.2.2.3.2 SMTP Transport Container

In AD DS, the SMTP Transport container contains all the siteLink and siteLinkBridge objects that connect two or more sites for intersite replication using the SMTP protocol. Not present in AD LDS.

parent: Inter-Site Transports container

objectClass: interSiteTransport

transportDLLName: The value of this attribute MUST be the string "ismsmtp.dll".

transportAddressAttribute: Identifies which attribute on the server object of a DC should be used as the network address of the DC for replication using this transport. For the SMTP transport, the attribute is mailAddress.

options: A set of bit flags as defined for options in section 6.1.1.2.2.3.1. For SMTP transport, the initial value is **NTDSTRANSPORT_OPT_IGNORE_SCHEDULES** present (options value 0x1).

6.1.1.2.2.3.3 Site Link Object

For a DC in one site to replicate directly with a DC in a different site, a siteLink object (or a series of siteLink objects) must connect the two sites specified. A siteLink object identifies the transport (wire protocol) to be used for replication between the sites. If the transport is IP, the siteLink object is a

child of the IP Transport container. If the transport is SMTP, the siteLink object is a child of the SMTP Transport container. Any single siteLink object may encompass two or more sites. If a siteLink object contains two sites, then those two sites are considered to be directly connected. If a siteLink object contains more than two sites, then all of the sites listed in the siteLink are considered to be connected in a mesh of point-to-point links.

parent: Either IP Transport container or SMTP Transport container.

objectClass: siteLink

systemFlags: FLAG_CONFIG_ALLOW_RENAME

cost: An administrator-defined cost value associated with that replication path.

siteList: Contains the **DSName** of the site objects for the sites that are connected using this site link.

replInterval: An interval that determines how frequently replication occurs over this site link during the times when the schedule allows replication.

schedule: Replication schedule of type SCHEDULE that specifies the time intervals when replication is permitted between the two sites.

options: A set of the following bit flags presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	DC	TS	UN

X: Unused. Must be zero and ignored.

UN (NTDSSITELINK_OPT_USE_NOTIFY, 0x00000001): If present, enables replication notifications (see Updates, section 3.1.1.5) between DCs in different sites in the siteList.

TS (NTDSSITELINK_OPT_TWOWAY_SYNC, 0x00000002): If present, forces a replication cycle in the opposite direction at the end of a replication cycle between DCs in different sites in the siteList.

DC (NTDSSITELINK_OPT_DISABLE_COMPRESSION, 0x00000004): If present, disables compression of IDL_DRSGetNCChanges response messages sent between DCs in different sites in the siteList.

6.1.1.2.2.3.4 Site Link Bridge Object

A siteLinkBridge object connects two or more siteLink objects that are associated with the same transport. The siteLinkBridge object is a child of the interSiteTransport object for the transport used by the siteLink objects that are being connected.

When **NTDSTRANSPORT_OPT_BRIDGES_REQUIRED** is present in the options of a transport, replication only assumes transitive communication between sites as specified in the siteLinkBridge objects for that transport. See the specification of IDL_QuerySitesByCost ([MS-DRSR] section 4.1.16.3).

parent: Either an IP transport container or an SMTP transport container.

objectClass: siteLinkBridge

systemFlags: FLAG_CONFIG_ALLOW_RENAME

siteLinkList: Contains the **DSNames** of the siteLink objects for the site links that are being connected by this site link bridge.

6.1.1.2.3 Display Specifiers Container

The Display Specifier objects are installed in the directory for use by the administrative applications of the directory. Each supported locale (that is, language and location) for the administrative application is assigned a number, called a Locale ID (LCID), and each of the children of the Display Specifier container is named with that number's hexadecimal character representation (for example, 1033 is named "409"). Section 2.2.1 contains a table associating each locale with an LCID. Some locales do not have Display Specifier objects installed by default.

name: DisplaySpecifiers

parent: Config NC root

objectClass: container

systemFlags: FLAG_DISALLOW_DELETE

6.1.1.2.3.1 Display Specifier Object

name: The name of each Display Specifier is a hexadecimal number in Unicode characters that represents a locale.

parent: Display Specifiers container

objectClass: displaySpecifier

The children of the Display Specifier object describe an implementation component of the administrative application. These objects are not interpreted by the DC.

The attributes of the children of the Display Specifier object are:

parent: Display Specifier object

objectClass: displaySpecifier

adminPropertyPages: Each administrative application component for an object of class displaySpecifier associates a **Component Object Model (COM)** object represented by a **universally unique identifier (UUID)** called a property page in this attribute. Each value in this multivalued attribute describes a single COM object. The description of a COM object is a string with the following format:

<order-number>,<UUID>,[optional data]

where:

- The order-number determines the desired ordering in the application for each COM object represented in the value.

- The UUID is a string representation of a UUID representing a COM object enclosed in curly braces.
- The optional data is passed to the COM object by the implementation.

shellPropertyPages: This attribute has the same semantics as adminPropertyPages.

adminContextMenu: This attribute can store values, where each value describes either a single COM object representation or a single application representation. For a COM object, this attribute has the same semantics as adminPropertyPage. For an application representation, the description is stored as a string with the following format:

<order-number>,<context menu name>,<program name>

where:

- The order-number determines the desired ordering for each COM object represented in the value.
- The context menu name is the text of the menu item for the administrative application interface.
- The program name is the application that is executed when the application references this adminContextMenu attribute. Either the full path must be specified, or the application must be in the search path.

shellContextMenu: This attribute has the same semantics as adminContextMenu.

adminMultiselectPropertyPages: This attribute has the same semantics as adminPropertyPages.

treatAsLeaf: This attribute is a Boolean that instructs the administrative application to ignore any child objects of this object, whether they exist or not.

creationWizard: The creationWizard attribute identifies primary object creation COM objects to replace the existing or native object creation wizard in Active Directory administrative applications. The COM objects in this value are represented by UUID.

createWizardExt: The createWizardExt attribute identifies secondary object creation COM objects for the administrative applications, if needed. This attribute is multivalued and requires the following format:

<order number>,<UUID>

where

- The order-number determines the desired ordering for each COM object represented in the value.
- The UUID is a string representation of a UUID representing a COM object.

iconPath: The iconPath attribute can be specified in one of two ways:

1. "<state>,<icon file name>" or
2. "<state>,<module file name>,<resource ID>"

In these examples, the "<state>" is an integer with a value between 0 and 15. The value 0 is defined to be the default or closed state of the icon. The value 1 is defined to be the open state of the icon. The value 2 is the disabled state. All other values are application-defined.

The "<icon file name>" is the path and file name of an icon file that contains the icon image.

The "<module file name>" is the path and file name of a module, such as an EXE or DLL, that contains the icon image in a resource. The "<resource ID>" is an integer that specifies the resource identifier of the icon resource within the module.

6.1.1.2.4 Services

name: Services

parent: Config NC root object

objectClass: container

systemFlags: { FLAG_DISALLOW_DELETE }

6.1.1.2.4.1 Windows NT

name: Windows NT

parent: Services

objectClass: container

6.1.1.2.4.1.1 Directory Service

name: Directory Service

parent: Windows NT (section 6.1.1.2.4.1)

objectClass: nTDSService

tombstoneLifetime: The number of days that a tombstone or recycled-object exists before it is garbage collected. See 3.1.1 for more information.

deletedObjectLifetime: The number of days that a deleted-object exists before it is transformed into a recycled-object. If no value is specified, the value of the tombstoneLifetime attribute is used instead.

sPNMappings: In AD DS, a set of SPN mappings, as specified in [MS-DRSR] section 4.1.4.2.19 (MapSPN). Not present in AD LDS.

msDS-Other-Settings: A multivalued string where each string value encodes a name-value pair. In the encoding, the name and value are separated by an "=". For example, the encoding of the name "DisableVLVSupport" with value "0" is "DisableVLVSupport=0". Each name is the name of an LDAP configurable setting, and the value is the value of that setting. The LDAP configurable settings and their effects are specified in section 3.1.1.3.4.7.

dSHeuristics: See section 6.1.1.2.4.1.2. By default, this attribute is not set.

6.1.1.2.4.1.2 dSHeuristics

dSHeuristics is a Unicode string attribute. Each character in the string represents a heuristic that is used to determine the behavior of Active Directory. These heuristics are described partly in this section and partly elsewhere in this specification.

The following constraints apply to the dSHeuristics string:

- The order of the characters in the string is fixed; characters can be omitted only by truncating the string.
- By default, the dSHeuristics attribute does not exist and, unless otherwise specified, the default value of each character in the dSHeuristics string is "0".
- When modifying an existing dSHeuristics string, the values of all existing characters that are not of interest to the modification should be preserved.

These constraints are illustrated by the following examples.

1. If dSHeuristics is not present or has length of zero, then the fSupFirstLastANR heuristic is false.
2. If dSHeuristics is only two Unicode characters long, then the fDoListObject heuristic, which would be represented by the third character in the string, is false.
3. Consider a scenario where the fSupFirstLastANR, fSupLastFirstANR, and fDoNickRes heuristics are required for certain system behaviors. The dSHeuristics string would consist of at least four characters, **fSupFirstLastANR**, **fSupLastFirstANR**, **fDoListObject**, and **fDoNickRes**, even though the fDoListObject heuristic is not needed. An implementer would set the **fDoListObject** character to the default value of "0" as described earlier.
4. Consider a scenario where anonymous LDAP operations to Active Directory need to be enabled. In this scenario, the seventh character of the dSHeuristics string, **fLDAPBlockAnonOps**, would be set to character "2". If the dSHeuristics string was already in existence before this operation, no characters in the dSHeuristics string other than the seventh character would be modified. If the dSHeuristics string did not yet exist before this operation, the first through sixth characters would be set to their default values, resulting in a dSHeuristics string of "0000002" in this case.

The following table describes the characters of the dSHeuristics string.

Character number	Character name	Description
1	fSupFirstLastANR	If this character is "0", then the fSupFirstLastANR heuristic is false; otherwise, the fSupFirstLastANR heuristic is true. Section 3.1.1.3.1.3.4 specifies the effects of this heuristic.
2	fSupLastFirstANR	If this character is "0", then the fSupLastFirstANR heuristic is false; otherwise, the fSupLastFirstANR heuristic is true. Section 3.1.1.3.1.3.4 specifies the effects of this heuristic.
3	fDoListObject	If this character is "1", then the fDoListObject heuristic is true; otherwise, the fDoListObject heuristic is false. Section 5.1.3.2 specifies the effects of this heuristic.
4	fDoNickRes	If this character is "0", then the fDoNickRes heuristic is false; otherwise, the fDoNickRes heuristic is true. The effects of the fDoNickRes heuristic are

Character number	Character name	Description
		outside the state model. If the fDoNickRes heuristic is true, an ANR request via MAPI attempts an exact match against the MAPI nickname attribute (the attribute with mAPIID equal to 0x3A00) before performing an ANR search (see section 3.1.1.3.1.3.4).
5	fLDAPUsePermMod	If this character is "0", then the fLDAPUsePermMod heuristic is false; otherwise, the fLDAPUsePermMod heuristic is true. If the fLDAPUsePermMod heuristic is true, then all LDAP Modify operations behave as if the LDAP_SERVER_PERMISSIVE_MODIFY_OID control was passed. Section 3.1.1.3.4.1.8 specifies the effects of the LDAP_SERVER_PERMISSIVE_MODIFY_OID control.
6	ulHideDSID	The ulHideDSID heuristic equates to the numeric value of this character; that is, character "0" equates to 0, character "1" equates to 1, and so on. The ulHideDSID heuristic controls when DSIDs are returned in the LDAP extended error string when an operation encounters an error. If the heuristic is 0, then DSIDs will be returned at all times. If the heuristic is 1, then DSIDs will be returned as long as the error is not a name error where different DSIDs may reveal the existence of an object that is not visible to the client. If the heuristic is anything but 0 or 1, then DSIDs will not be returned at all. A DSID consists of the string "DSID-", followed by an implementation-specific 32-bit integer expressed in hexadecimal. The integer identifies the execution point at which an error occurred.
7	fLDAPBlockAnonOps	If this character is "2", then the fLDAPBlockAnonOps heuristic is false; otherwise, the fLDAPBlockAnonOps heuristic is true. If this character is not present in the string, it defaults to "2" when the DC functional level is less than DS_BEHAVIOR_WIN2003, and to "0" otherwise. Section 5.1.3 specifies the effects of this heuristic.
8	fAllowAnonNSPI	If this character is "0", then the fAllowAnonNSPI heuristic is false; otherwise, the fAllowAnonNSPI heuristic is true. If the fAllowAnonNSPI heuristic is true, allow

Character number	Character name	Description
		anonymous calls to the **name service provider interface (NSPI)** RPC bind method. Otherwise, only allow authenticated clients.
9	fUserPwdSupport	If this character is neither "0" nor "2", then the fUserPwdSupport heuristic is true. If this character is "2", then the fUserPwdSupport heuristic is false. If this character is "0", then the fUserPwdSupport heuristic is false for AD DS and true for AD LDS. Sections 3.1.1.3.1.5.2 and 3.1.1.4.4 specify the effects of this heuristic.
10	tenthChar	When setting dSHeuristics to a value that is 10 or more Unicode characters long, if the value of **tenthChar** is not character "1", the server rejects the update. See section 3.1.1.5.3.2.
11	fSpecifyGUIDOnAdd	If this character is "0", then the fSpecifyGUIDOnAdd heuristic is false; otherwise, the fSpecifyGUIDOnAdd heuristic is true. The fSpecifyGUIDOnAdd heuristic applies only to AD DS. AD LDS always treats this heuristic as if the character is "0"; that is, as if the fSpecifyGUIDOnAdd heuristic is false. Section 3.1.1.5.2.2 specifies the effects of this heuristic.
12	fDontStandardizeSDs	If this character is "0", then the fDontStandardizeSDs heuristic is false; otherwise, the fDontStandardizeSDs heuristic is true. Section 6.1.3 specifies the effects of this heuristic.
13	fAllowPasswordOperationsOverNonSecureConnection	If this character is "0", then the fAllowPasswordOperationsOverNonSecureConnection heuristic is false; otherwise, the fAllowPasswordOperationsOverNonSecureConnection heuristic is true. The fAllowPasswordOperationsOverNonSecureConnection heuristic applies only to AD LDS. Sections 3.1.1.3.1.5.1, 3.1.1.5.2.2, and 3.1.1.5.3.2 specify the effects of this heuristic.
14	fDontPropagateOnNoChangeUpdate	If this character is "0", then the fDontPropagateOnNoChangeUpdate heuristic is false; otherwise, the fDontPropagateOnNoChangeUpdate heuristic is true. If the fDontPropagateOnNoChangeUpdate heuristic is true, when the

Character number	Character name	Description
		nTSecurityDescriptor attribute of an object is set to a value that is bitwise identical to the current value, no work item is enqueued for the task that updates the security descriptors on the children of a modified object in order to propagate inherited ACEs (section 6.1.3). If the fDontPropagateOnNoChangeUpdate heuristic is false, a work item is always enqueued when the nTSecurityDescriptor attribute is modified. The fDontPropagateOnNoChangeUpdate heuristic applies to Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system. Windows 2000 Server operating system and Windows Server 2003 operating system versions of Active Directory behave as if the fDontPropagateOnNoChangeUpdate heuristic is false.
15	fComputeANRStats	If this character is "0", then the fComputeANRStats heuristic is false; otherwise, the fComputeANRStats heuristic is true. The effects of the fComputeANRStats heuristic are outside the state model. If the fComputeANRStats heuristic is true, ANR searches (section 3.1.1.3.1.3.4) are optimized using cardinality estimates like all other searches.
16	dwAdminSDExMask	The valid values for this character are from the set "0"–"9" and "a"–"f". The dwAdminSDExMask heuristic equals the character interpreted as a hex digit and converted into a 4-bit value (that is, "1"=0x1, "f"=0xF). Section 3.1.1.6.1 specifies the effects of this heuristic.
17	fKVNOEmuW2K	If this character is "0", then the fKVNOEmuW2K heuristic is false; otherwise, the fKVNOEmuW2K heuristic is true. Section 3.1.1.4.5.16 specifies the effects of this heuristic.
18	fLDAPBypassUpperBoundsOnLimits	If this character is "0", then the fLDAPBypassUpperBoundsOnLimits heuristic is false; otherwise, the fLDAPBypassUpperBoundsOnLimits heuristic is true. If the fLDAPBypassUpperBoundsOnLimits

Character number	Character name	Description
		heuristic is false, DCs impose implementation-dependent limits when interpreting values of the LDAP policies specified in section 3.1.1.3.4.6. If the configured policy value exceeds the limit, the DC ignores the policy value and instead uses the implementation-dependent limit. This heuristic applies to Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2. Windows 2000 Server and Windows Server 2003 versions of Active Directory do not impose any such limits.
19	fDisableAutoIndexingOnSchemaUpdate	If this character is "0", then the fDisableAutoIndexingOnSchemaUpdate heuristic is false; otherwise, the DisableAutoIndexingOnSchemaUpdate heuristic is true. The effects of the fDisableAutoIndexingOnSchemaUpdate heuristic are outside the state model. If the fDisableAutoIndexingOnSchemaUpdate heuristic is false, DCs can initiate index creation upon detection of index-related changes to the searchFlags attribute (see section 2.2.10). If the fDisableAutoIndexingOnSchemaUpdate heuristic is true, it is a hint to DCs that index creation can be delayed upon detection of index-related changes to the searchFlags attribute until either an administrator issues the schemaUpdateNow rootDSE modify operation, the DC is rebooted, or an implementation-dependent time period has elapsed. This heuristic applies to Windows Server 2012 and Windows Server 2012 R2. Earlier versions of Active Directory do not implement support for this heuristic.
20	twentiethChar	When setting dSHeuristics to a value that is 20 or more Unicode characters long, if the value of twentiethChar is not character "2", the server rejects the update. See section 3.1.1.5.3.2.
21	DoNotVerifyUPNUniqueness	If this character is anything other than "0", AD LDS will not check values of userPrincipalName for uniqueness. See section 3.1.1.5.2.2. This heuristic applies to Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, and Windows Server 2012.
22-23	MinimumGetChangesRequestVersion	A hexadecimal value, ranging from "00" to

Character number	Character name	Description
		"FF". This value controls the minimum version of the DRS_MSG_GETCHGREQ* structures the DC will send or accept. If the value is not set, the value "00" is used. When the value is "00", no restriction is enforced. See [MS-DRSR] section 4.1.10.5.1.
24-25	MinimumGetChangesReplyVersion	A hex value, ranging from "00" to "FF". This value controls the minimum version of the DRS_MSG_GETCHGREPLY* structures the DC will send or accept. If the value is not set, the value "00" is used. When the value is "00", no restriction is enforced. See [MS-DRSR] section 4.1.10.5.18.

6.1.1.2.4.1.3 Optional Features Container

A container to store the optional features objects. See section 3.1.1.9, Optional Features.

parent: Directory Service

name: Optional Features

objectClass: container

6.1.1.2.4.1.3.1 Recycle Bin Feature Object

An msDS-OptionalFeature object that represents the Recycle Bin optional feature. See section 3.1.1.9 for information on optional features. See section 3.1.1.9.1 for the effects of the Recycle Bin optional feature.

parent: Optional Features

name: Recycle Bin Feature

objectClass: msDS-OptionalFeature

msDS-OptionalFeatureFlags: FOREST_OPTIONAL_FEATURE (see section 3.1.1.9)

msDS-OptionalFeatureGUID: 766ddcd8-acd0-445e-f3b9-a7f9b6744f2a

msDS-RequiredForestBehaviorVersion: DS_BEHAVIOR_WIN2008R2

6.1.1.2.4.1.4 Query-Policies

A container to store the default queryPolicy object. Can also contain queryPolicy objects created by administrators. See section 3.1.1.3.4.6 for the effects of queryPolicy objects.

name: Query-Policies

parent: Directory Service

objectClass: container

6.1.1.2.4.1.4.1 Default Query Policy

Stores the default LDAP query policies. See section 3.1.1.3.4.6 for the effects of the default queryPolicy object.

name: Default Query Policy

parent: Query-Policies

objectClass: queryPolicy

lDAPAdminLimits: Encoding of the LDAP policies as specified in section 3.1.1.3.4.6.

6.1.1.2.4.1.5 SCP Publication Service Object

This object is present only in AD LDS. It is system-created but can be removed and recreated by the administrator if desired. This object stores forest-wide configuration that is used to control the creation of serviceConnectionPoint objects by an AD LDS DC running on a computer joined to an AD DS domain. Section 6.3.8 specifies the effects of this object.

name: SCP Publication Service

parent: Directory Service

objectClass: msDS-ServiceConnectionPointPublicationService

Enabled: A Boolean value. If false, no DC in this forest will create a serviceConnectionPoint object.

msDS-DisableForInstances: A set of references to nTDSDSA objects in this forest. A DC in this set will not create a serviceConnectionPoint object.

msDS-SCPContainer: If present, is a reference to an AD DS object (a reference to an object outside this AD LDS forest). The parent of any serviceConnectionPoint object created by a DC in this forest is msDS-SCPContainer. If an AD LDS DC in this forest is joined to domain D, then a DC of domain D must be capable of generating a referral to a DC containing a writable replica of the NC containing msDS-SCPContainer.

keywords: A set of strings. The keywords attribute of any serviceConnectionPoint object created by a DC in this forest contains all of these strings. There are no semantic constraints imposed on this attribute apart from any syntactic constraints that may be imposed by the schema.

6.1.1.2.5 Physical Locations

This object is not present on AD LDS.

name: Physical Locations

parent: config NC root object

objectClass: physicalLocation

6.1.1.2.6 WellKnown Security Principals

This object is not present on AD LDS.

name: WellKnown Security Principals

parent: config NC root object

objectClass: container

systemFlags: { FLAG_DISALLOW_DELETE}

6.1.1.2.6.1 Anonymous Logon

name: Anonymous Logon

parent: WellKnown Security Principals

objectSid: S-1-5-7

6.1.1.2.6.2 Authenticated Users

name: Authenticated Users

parent: WellKnown Security Principals

objectSid: S-1-5-11

6.1.1.2.6.3 Batch

name: Batch

parent: WellKnown Security Principals

objectSid: S-1-5-3

6.1.1.2.6.4 Console Logon

name: Console Logon

parent: WellKnown Security Principals

objectSid: S-1-2-1

6.1.1.2.6.5 Creator Group

name: Creator Group

parent: WellKnown Security Principals

objectSid: S-1-3-1

6.1.1.2.6.6 Creator Owner

name: Creator Owner

parent: WellKnown Security Principals

objectSid: S-1-3-0

6.1.1.2.6.7 Dialup

name: Dialup

parent: WellKnown Security Principals

objectSid: S-1-5-1

6.1.1.2.6.8 Digest Authentication

name: Digest Authentication

parent: WellKnown Security Principals

objectSid: S-1-5-64-21

6.1.1.2.6.9 Enterprise Domain Controllers

name: Enterprise Domain Controllers

parent: WellKnown Security Principals

objectSid: S-1-5-9

By default all normal (writable) DCs in the forest belong to this group.

6.1.1.2.6.10 Everyone

name: Everyone

parent: WellKnown Security Principals

objectSid: S-1-1-0

6.1.1.2.6.11 Interactive

name: Interactive

parent: WellKnown Security Principals

objectSid: S-1-5-4

6.1.1.2.6.12 IUSR

name: IUSR

parent: WellKnown Security Principals

objectSid: S-1-5-17

6.1.1.2.6.13 Local Service

name: Local Service

parent: WellKnown Security Principals

objectSid: S-1-5-19

6.1.1.2.6.14 Network

name: Network

parent: WellKnown Security Principals

objectSid: S-1-5-2

6.1.1.2.6.15 Network Service

name: Network Service

parent: WellKnown Security Principals

objectSid: S-1-5-20

6.1.1.2.6.16 NTLM Authentication

name: NTLM Authentication

parent: WellKnown Security Principals

objectSid: S-1-5-64-10

6.1.1.2.6.17 Other Organization

name: Other Organization

parent: WellKnown Security Principals

objectSid: S-1-5-1000

6.1.1.2.6.18 Owner Rights

name: Owner Rights

parent: WellKnown Security Principals

objectSid: S-1-3-4

6.1.1.2.6.19 Proxy

name: Proxy

parent: WellKnown Security Principals

objectSid: S-1-5-8

6.1.1.2.6.20 Remote Interactive Logon

name: Remote Interactive Logon

parent: WellKnown Security Principals

objectSid: S-1-5-14

6.1.1.2.6.21 Restricted

name: Restricted

parent: WellKnown Security Principals

objectSid: S-1-5-12

6.1.1.2.6.22 SChannel Authentication

name: SChannel Authentication

parent: WellKnown Security Principals

objectSid: S-1-5-64-14

6.1.1.2.6.23 Self

name: Self

parent: WellKnown Security Principals

objectSid: S-1-5-10

6.1.1.2.6.24 Service

name: Service

parent: WellKnown Security Principals

objectSid: S-1-5-6

6.1.1.2.6.25 System

name: System

parent: WellKnown Security Principals

objectSid: S-1-5-18

6.1.1.2.6.26 Terminal Server User

name: Terminal Server User

parent: WellKnown Security Principals

objectSid: S-1-5-13

6.1.1.2.6.27 This Organization

name: This Organization

parent: WellKnown Security Principals

objectSid: S-1-5-15

6.1.1.2.7 Extended Rights

name: Extended Rights

parent: Config NC root object

objectClass: container

systemFlags: { FLAG_DISALLOW_DELETE }

6.1.1.2.7.1 controlAccessRight objects

All controlAccessRight objects have:

objectClass: controlAccessRight

rightsGuid: This value is the identifier of the control access right used for security descriptors and SDDL.

appliesTo: Each value in this attribute is a GUID, with each GUID equaling an attribute schemaIDGUID on a schema object defining a class in the schema NC. This class defines the objects in which the control access right can be a security descriptor for. The appliesTo values on the controlAccessRight are not enforced by the directory server; that is, the controlAccessRight can be included in security descriptors of objects of classes not specified in the appliesTo attribute.

localizationDisplayId: This is implementation-specific information for the administrative application.

validAccesses: This is implementation-specific information for the administrative application.

6.1.1.2.7.2 Change-Rid-Master

This object is present in AD DS only.

name: Change-Rid-Master

rightsGuid: d58d5f36-0a98-11d1-adbb-00c04fd8d5cd

appliesTo: 6617188d-8f3c-11d0-afda-00c04fd930c9

6.1.1.2.7.3 Do-Garbage-Collection

This object is present in AD DS and AD LDS.

name: Do-Garbage-Collection

rightsGuid: fec364e0-0a98-11d1-adbb-00c04fd8d5cd

appliesTo: f0f8ffab-1191-11d0-a060-00aa006c33ed

6.1.1.2.7.4 Recalculate-Hierarchy

This object is present in AD DS only.

name: Recalculate-Hierarchy

rightsGuid: 0bc1554e-0a99-11d1-adbb-00c04fd8d5cd

appliesTo: f0f8ffab-1191-11d0-a060-00aa006c33ed

6.1.1.2.7.5 Allocate-Rids

This object is present in AD DS only.

name: Allocate-Rids

rightsGuid: 1abd7cf8-0a99-11d1-adbb-00c04fd8d5cd

appliesTo: f0f8ffab-1191-11d0-a060-00aa006c33ed

6.1.1.2.7.6 Change-PDC

This object is present in AD DS only.

name: Change-PDC

rightsGuid: bae50096-4752-11d1-9052-00c04fc2d4cf

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.7 Add-GUID

This object is present in AD DS and AD LDS.

name: Add-GUID

rightsGuid: 440820ad-65b4-11d1-a3da-0000f875ae0d

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.8 Change-Domain-Master

This object is present in AD DS only.

name: Change-Domain-Master

rightsGuid: 014bf69c-7b3b-11d1-85f6-08002be74fab

appliesTo: ef9e60e0-56f7-11d1-a9c6-0000f80367c1

6.1.1.2.7.9 Public-Information

This object is present in AD DS and AD LDS.

name: Public-Information

rightsGuid: e48d0154-bcf8-11d1-8702-00c04fb96050

appliesTo:

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- bf967a86-0de6-11d0-a285-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2
- ce206244-5827-4a86-ba1c-1c0c386c1b64 (for AD DS only)

6.1.1.2.7.10 msmq-Receive-Dead-Letter

This object is present in AD DS only.

name: msmq-Receive-Dead-Letter

rightsGuid: 4b6e08c0-df3c-11d1-9c86-006008764d0e

appliesTo: 9a0dc344-c100-11d1-bbc5-0080c76670c0

6.1.1.2.7.11 msmq-Peek-Dead-Letter

This object is present in AD DS only.

name: msmq-Peek-Dead-Letter

rightsGuid: 4b6e08c1-df3c-11d1-9c86-006008764d0e

appliesTo: 9a0dc344-c100-11d1-bbc5-0080c76670c0

6.1.1.2.7.12 msmq-Receive-computer-Journal

This object is present in AD DS only.

name: msmq-Receive-computer-Journal

rightsGuid: 4b6e08c2-df3c-11d1-9c86-006008764d0e

appliesTo: 9a0dc344-c100-11d1-bbc5-0080c76670c0

6.1.1.2.7.13 msmq-Peek-computer-Journal

This object is present in AD DS only.

name: msmq-Peek-computer-Journal

rightsGuid: 4b6e08c3-df3c-11d1-9c86-006008764d0e

appliesTo: 9a0dc344-c100-11d1-bbc5-0080c76670c0

6.1.1.2.7.14 msmq-Receive

This object is present in AD DS only.

name: msmq-Receive

rightsGuid: 06bd3200-df3e-11d1-9c86-006008764d0e

appliesTo: 9a0dc343-c100-11d1-bbc5-0080c76670c0

6.1.1.2.7.15 msmq-Peek

This object is present in AD DS only.

name: msmq-Peek

rightsGuid: 06bd3201-df3e-11d1-9c86-006008764d0e

appliesTo: 9a0dc343-c100-11d1-bbc5-0080c76670c0

6.1.1.2.7.16 msmq-Send

This object is present in AD DS only.

name: msmq-Send

rightsGuid: 06bd3202-df3e-11d1-9c86-006008764d0e

appliesTo:

- 9a0dc343-c100-11d1-bbc5-0080c76670c0
- 46b27aac-aafa-4ffb-b773-e5bf621ee87b (only in schema version 30 and greater)

6.1.1.2.7.17 msmq-Receive-journal

This object is present in AD DS only.

name: msmq-Receive-journal

rightsGuid: 06bd3203-df3e-11d1-9c86-006008764d0e

appliesTo: 9a0dc343-c100-11d1-bbc5-0080c76670c0

6.1.1.2.7.18 msmq-Open-Connector

This object is present in AD DS only.

name: msmq-Open-Connector

rightsGuid: b4e60130-df3f-11d1-9c86-006008764d0e

appliesTo: bf967ab3-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.19 Apply-Group-Policy

This object is present in AD DS only.

name: Apply-Group-Policy

rightsGuid: edacfd8f-ffb3-11d1-b41d-00a0c968f939

appliesTo: f30e3bc2-9ff0-11d1-b603-0000f80367c1

6.1.1.2.7.20 RAS-Information

This object is present in AD DS only.

name: RAS-Information

rightsGuid: 037088f8-0ae1-11d2-b422-00a0c968f939

appliesTo:

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- bf967aba-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.21 DS-Install-Replica

This object is present in AD DS and AD LDS.

name: DS-Install-Replica

rightsGuid: 9923a32a-3607-11d2-b9be-0000f87a36b2

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.22 Change-Infrastructure-Master

This object is present in AD DS only.

name: Change-Infrastructure-Master

rightsGuid: cc17b1fb-33d9-11d2-97d4-00c04fd8d5cd

appliesTo: 2df90d89-009f-11d2-aa4c-00c04fd7d83a

6.1.1.2.7.23 Update-Schema-Cache

This object is present in AD DS and AD LDS.

name: Update-Schema-Cache

rightsGuid: be2bb760-7f46-11d2-b9ad-00c04f79f805

appliesTo: bf967a8f-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.24 Recalculate-Security-Inheritance

This object is present in AD DS and AD LDS.

name: Recalculate-Security-Inheritance

rightsGuid: 62dd28a8-7f46-11d2-b9ad-00c04f79f805

appliesTo: f0f8ffab-1191-11d0-a060-00aa006c33ed

6.1.1.2.7.25 DS-Check-Stale-Phantoms

This object is present in AD DS only.

name: DS-Check-Stale-Phantoms

rightsGuid: 69ae6200-7f46-11d2-b9ad-00c04f79f805

appliesTo: f0f8ffab-1191-11d0-a060-00aa006c33ed

6.1.1.2.7.26 Certificate-Enrollment

This object is present in AD DS only.

name: Certificate-Enrollment

rightsGuid: 0e10c968-78fb-11d2-90d4-00c04f79dc55

appliesTo: e5209ca2-3bba-11d2-90cc-00c04fd91ab1

6.1.1.2.7.27 Self-Membership

This object is present in AD DS and AD LDS.

name: Self-Membership

rightsGuid: bf9679c0-0de6-11d0-a285-00aa003049e2

appliesTo: bf967a9c-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.28 Validated-DNS-Host-Name

This object is present in AD DS only.

name: Validated-DNS-Host-Name

rightsGuid: 72e39547-7b18-11d1-adef-00c04fd8d5cd

appliesTo:

- bf967a86-0de6-11d0-a285-00aa003049e2
- ce206244-5827-4a86-ba1c-1c0c386c1b64 (only in schema version 45 and greater)
- 7b8b558a-93a5-4af7-adca-c017e67f1057 (only in schema version 55 and greater)

6.1.1.2.7.29 Validated-SPN

This object is present in AD DS only.

name: Validated-SPN

rightsGuid: f3a64788-5306-11d1-a9c5-0000f80367c1

appliesTo:

- bf967a86-0de6-11d0-a285-00aa003049e2
- ce206244-5827-4a86-ba1c-1c0c386c1b64 (only in schema version 45 and greater)

6.1.1.2.7.30 Generate-RSoP-Planning

This object is present in AD DS only.

name: Generate-RSoP-Planning

rightsGuid: b7b1b3dd-ab09-4242-9e30-9980e5d322f7

appliesTo:

- 19195a5b-6da0-11d0-afd3-00c04fd930c9
- bf967aa5-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.31 Refresh-Group-Cache

This object is present in AD DS only.

name: Refresh-Group-Cache

rightsGuid: 9432c620-033c-4db7-8b58-14ef6d0bf477

appliesTo: f0f8ffab-1191-11d0-a060-00aa006c33ed

6.1.1.2.7.32 Reload-SSL-Certificate

This object is present in AD DS and AD LDS.

name: Reload-SSL-Certificate

rightsGuid: 1a60ea8d-58a6-4b20-bcdc-fb71eb8a9ff8

appliesTo: f0f8ffab-1191-11d0-a060-00aa006c33ed

6.1.1.2.7.33 SAM-Enumerate-Entire-Domain

This object is present in AD DS only.

name: SAM-Enumerate-Entire-Domain

rightsGuid: 91d67418-0135-4acc-8d79-c08e857cfbec

appliesTo: bf967aad-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.34 Generate-RSoP-Logging

This object is present in AD DS only.

name: Generate-RSoP-Logging

rightsGuid: b7b1b3de-ab09-4242-9e30-9980e5d322f7

appliesTo:

- 19195a5b-6da0-11d0-afd3-00c04fd930c9
- bf967aa5-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.35 Domain-Other-Parameters

This object is present in AD DS only.

name: Domain-Other-Parameters

rightsGuid: b8119fd0-04f6-4762-ab7a-4986c76b3f9a

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.36 DNS-Host-Name-Attributes

This object is present in AD DS only.

name: DNS-Host-Name-Attributes

rightsGuid: 72e39547-7b18-11d1-adef-00c04fd8d5cd

appliesTo:

- bf967a86-0de6-11d0-a285-00aa003049e2
- ce206244-5827-4a86-ba1c-1c0c386c1b64
- 7b8b558a-93a5-4af7-adca-c017e67f1057 (only in schema version 55 and greater)

6.1.1.2.7.37 Create-Inbound-Forest-Trust

This object is present in AD DS only.

name: Create-Inbound-Forest-Trust

rightsGuid: e2a36dc9-ae17-47c3-b58b-be34c55ba633

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.38 DS-Replication-Get-Changes-All

This object is present in AD DS and AD LDS.

name: DS-Replication-Get-Changes-All

rightsGuid: 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2

appliesTo:

- bf967a8f-0de6-11d0-a285-00aa003049e2
- bf967a87-0de6-11d0-a285-00aa003049e2
- 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.39 Migrate-SID-History

This object is present in AD DS only.

name: Migrate-SID-History

rightsGuid: BA33815A-4F93-4c76-87F3-57574BFF8109

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.40 Reanimate-Tombstones

This object is present in AD DS and AD LDS.

name: Reanimate-Tombstones

rightsGuid: 45EC5156-DB7E-47bb-B53F-DBEB2D03C40F

appliesTo:

- bf967a8f-0de6-11d0-a285-00aa003049e2
- bf967a87-0de6-11d0-a285-00aa003049e2

- 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.41 Allowed-To-Authenticate

This object is present in AD DS only.

name: Allowed-To-Authenticate

rightsGuid: 68B1D179-0D15-4d4f-AB71-46152E79A7BC

appliesTo:

- 4828cc14-1437-45bc-9b07-ad6f015e5f28
- bf967aba-0de6-11d0-a285-00aa003049e2
- bf967a86-0de6-11d0-a285-00aa003049e2
- ce206244-5827-4a86-ba1c-1c0c386c1b64 (only in schema version 45 and greater)

6.1.1.2.7.42 DS-Execute-Intentions-Script

This object is present in AD DS and AD LDS.

name: DS-Execute-Intentions-Script

rightsGuid: 2f16c4a5-b98e-432c-952a-cb388ba33f2e

appliesTo: ef9e60e0-56f7-11d1-a9c6-0000f80367c1

6.1.1.2.7.43 DS-Replication-Monitor-Topology

This object is present in AD DS and AD LDS.

name: DS-Replication-Monitor-Topology

rightsGuid: f98340fb-7c5b-4cdb-a00b-2ebdfa115a96

appliesTo:

- bf967a8f-0de6-11d0-a285-00aa003049e2
- bf967a87-0de6-11d0-a285-00aa003049e2
- 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.44 Update-Password-Not-Required-Bit

This object is present in AD DS only.

name: Update-Password-Not-Required-Bit

rightsGuid: 280f369c-67c7-438e-ae98-1d46f3c6f541

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.45 Unexpire-Password

This object is present in AD DS and AD LDS.

name: Unexpire-Password

rightsGuid: ccc2dc7d-a6ad-4a7a-8846-c04e3cc53501

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.46 Enable-Per-User-Reversibly-Encrypted-Password

This object is present in AD DS only.

name: Enable-Per-User-Reversibly-Encrypted-Password

rightsGuid: 05c74c5e-4deb-43b4-bd9f-86664c2a7fd5

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.47 DS-Query-Self-Quota

This object is present in AD DS and AD LDS.

name: DS-Query-Self-Quota

rightsGuid: 4ecc03fe-ffc0-4947-b630-eb672a8a9dbc

appliesTo: da83fc4f-076f-4aea-b4dc-8f4dab9b5993

6.1.1.2.7.48 Private-Information

This object is present in AD DS only.

name: Private-Information

rightsGuid: 91e647de-d96f-4b70-9557-d63ff4f3ccd8

appliesTo:

- bf967aba-0de6-11d0-a285-00aa003049e2
- 4828cc14-1437-45bc-9b07-ad6f015e5f28

6.1.1.2.7.49 MS-TS-GatewayAccess

This object is present in AD DS only.

name: MS-TS-GatewayAccess

rightsGuid: ffa6f046-ca4b-4feb-b40d-04dfee722543

appliesTo:

- bf967a86-0de6-11d0-a285-00aa003049e2
- ce206244-5827-4a86-ba1c-1c0c386c1b64

6.1.1.2.7.50 Terminal-Server-License-Server

This object is present in AD DS only.

name: Terminal-Server-License-Server

rightsGuid: 5805bc62-bdc9-4428-a5e2-856a0f4c185e

appliesTo:

- bf967aba-0de6-11d0-a285-00aa003049e2
- 4828cc14-1437-45bc-9b07-ad6f015e5f28

6.1.1.2.7.51 Domain-Administer-Server

This object is present in AD DS only.

name: Domain-Administer-Server

rightsGuid: ab721a52-1e2f-11d0-9819-00aa0040529b

appliesTo: bf967aad-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.52 User-Change-Password

This object is present in AD DS and AD LDS.

name: User-Change-Password

rightsGuid: ab721a53-1e2f-11d0-9819-00aa0040529b

appliesTo:

- bf967a86-0de6-11d0-a285-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2
- 4828CC14-1437-45bc-9B07-AD6F015E5F28 (only in schema version 30 and greater)
- ce206244-5827-4a86-ba1c-1c0c386c1b64 (only in schema version 45 and greater, for AD DS only)

6.1.1.2.7.53 User-Force-Change-Password

This object is present in AD DS and AD LDS.

name: User-Force-Change-Password

rightsGuid: 00299570-246d-11d0-a768-00aa006e0529

appliesTo:

- bf967a86-0de6-11d0-a285-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2
- 4828CC14-1437-45bc-9B07-AD6F015E5F28 (only in schema version 30 and greater)

- ce206244-5827-4a86-ba1c-1c0c386c1b64 (only in schema version 45 and greater, for AD DS only)

6.1.1.2.7.54 Send-As

This object is present in AD DS only.

name: Send-As

rightsGuid: ab721a54-1e2f-11d0-9819-00aa0040529b

appliesTo:

- bf967a86-0de6-11d0-a285-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2
- 4828CC14-1437-45bc-9B07-AD6F015E5F28 (only in schema version 30 and greater)
- ce206244-5827-4a86-ba1c-1c0c386c1b64 (only in schema version 45 and greater)

6.1.1.2.7.55 Receive-As

This object is present in AD DS only.

name: Receive-As

rightsGuid: ab721a56-1e2f-11d0-9819-00aa0040529b

appliesTo:

- bf967a86-0de6-11d0-a285-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2
- 4828CC14-1437-45bc-9B07-AD6F015E5F28 (only in schema version 30 and greater)
- ce206244-5827-4a86-ba1c-1c0c386c1b64 (only in schema version 45 and greater)

6.1.1.2.7.56 Send-To

This object is present in AD DS only.

name: Send-To

rightsGuid: ab721a55-1e2f-11d0-9819-00aa0040529b

appliesTo: bf967a9c-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.57 Domain-Password

This object is present in AD DS only.

name: Domain-Password

rightsGuid: c7407360-20bf-11d0-a768-00aa006e0529

appliesTo:

- 19195a5b-6da0-11d0-afd3-00c04fd930c9
- 19195a5a-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.58 General-Information

This object is present in AD DS and AD LDS.

name: General-Information

rightsGuid: 59ba2f42-79a2-11d0-9020-00c04fc2d3cf

appliesTo:

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- bf967aba-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.59 User-Account-Restrictions

This object is present in AD DS and AD LDS.

name: User-Account-Restrictions

rightsGuid: 4c164200-20c0-11d0-a768-00aa006e0529

appliesTo:

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- bf967a86-0de6-11d0-a285-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2
- ce206244-5827-4a86-ba1c-1c0c386c1b64 (for AD DS only)
- 7b8b558a-93a5-4af7-adca-c017e67f1057 (only in schema version 54 and greater, for AD DS only)

6.1.1.2.7.60 User-Logon

This object is present in AD DS and AD LDS.

name: User-Logon

rightsGuid: 5f202010-79a5-11d0-9020-00c04fc2d4cf

appliesTo:

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- bf967aba-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.61 Membership

This object is present in AD DS and AD LDS.

name: Membership

rightsGuid: bc0ac240-79a9-11d0-9020-00c04fc2d4cf

appliesTo:

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- bf967aba-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.62 Open-Address-Book

This object is present in AD DS only.

name: Open-Address-Book

rightsGuid: a1990816-4298-11d1-ade2-00c04fd8d5cd

appliesTo: 3e74f60f-3e73-11d1-a9c0-0000f80367c1

6.1.1.2.7.63 Personal-Information

This object is present in AD DS and AD LDS.

name: Personal-Information

rightsGuid: 77B5B886-944A-11d1-AEBD-0000F80367C1

appliesTo:

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- bf967a86-0de6-11d0-a285-00aa003049e2
- 5cb41ed0-0e4c-11d0-a286-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2
- ce206244-5827-4a86-ba1c-1c0c386c1b64 (for AD DS only)
- 641e87a4-8326-4771-ba2d-c706df35e35a (only in schema version 52 or greater)

6.1.1.2.7.64 Email-Information

This object is present in AD DS and AD LDS.

name: Email-Information

rightsGuid: E45795B2-9455-11d1-AEBD-0000F80367C1

appliesTo:

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- bf967a9c-0de6-11d0-a285-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.65 Web-Information

This object is present in AD DS and AD LDS.

name: Web-Information

rightsGuid: E45795B3-9455-11d1-AEBD-0000F80367C1

appliesTo:

- 4828CC14-1437-45bc-9B07-AD6F015E5F28
- 5cb41ed0-0e4c-11d0-a286-00aa003049e2
- bf967aba-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.66 DS-Replication-Get-Changes

This object is present in AD DS and AD LDS.

name: DS-Replication-Get-Changes

rightsGuid: 1131f6aa-9c07-11d1-f79f-00c04fc2dcd2

appliesTo:

- bf967a8f-0de6-11d0-a285-00aa003049e2
- bf967a87-0de6-11d0-a285-00aa003049e2
- 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.67 DS-Replication-Synchronize

This object is present in AD DS and AD LDS.

name: DS-Replication-Synchronize

rightsGuid: 1131f6ab-9c07-11d1-f79f-00c04fc2dcd2

appliesTo:

- bf967a8f-0de6-11d0-a285-00aa003049e2
- bf967a87-0de6-11d0-a285-00aa003049e2
- 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.68 DS-Replication-Manage-Topology

This object is present in AD DS and AD LDS.

name: DS-Replication-Manage-Topology

rightsGuid: 1131f6ac-9c07-11d1-f79f-00c04fc2dcd2

appliesTo:

- bf967a8f-0de6-11d0-a285-00aa003049e2
- bf967a87-0de6-11d0-a285-00aa003049e2
- 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.69 Change-Schema-Master

This object is present in AD DS and AD LDS.

name: Change-Schema-Master

rightsGuid: e12b56b6-0a95-11d1-adbb-00c04fd8d5cd

appliesTo: bf967a8f-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.70 DS-Replication-Get-Changes-In-Filtered-Set

This object is present in AD DS only.

name: DS-Replication-Get-Changes-In-Filtered-Set

rightsGuid: 89e95b76-444d-4c62-991a-0facbeda640c

appliesTo:

- 19195a5b-6da0-11d0-afd3-00c04fd930c9
- bf967a87-0de6-11d0-a285-00aa003049e2
- bf967a8f-0de6-11d0-a285-00aa003049e2

6.1.1.2.7.71 Run-Protect-Admin-Groups-Task

This object is present in AD DS only.

name: Run-Protect-Admin-Groups-Task

rightsGuid: 7726b9d5-a4b4-4288-a6b2-dce952e80a7f

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.72 Manage-Optional-Features

This object is present in AD DS and AD LDS.

name: Manage-Optional-Features

rightsGuid: 7c0e2a7c-a419-48e4-a995-10180aad54dd

appliesTo: ef9e60e0-56f7-11d1-a9c6-0000f80367c1

6.1.1.2.7.73 Read-Only-Replication-Secret-Synchronization

This object is present in AD DS only.

name: Read-Only-Replication-Secret-Synchronization

rightsGuid: 1131f6ae-9c07-11d1-f79f-00c04fc2dcd2

appliesTo

- bf967a8f-0de6-11d0-a285-00aa003049e2
- bf967a87-0de6-11d0-a285-00aa003049e2
- 19195a5b-6da0-11d0-afd3-00c04fd930c9

6.1.1.2.7.74 Validated-MS-DS-Additional-DNS-Host-Name

This object is present in AD DS only.

name: Validated-MS-DS-Additional-DNS-Host-Name

rightsGuid: 80863791-dbe9-4eb8-837e-7f0ab55d9ac7

appliesTo: bf967a86-0de6-11d0-a285-00aa003049e2

This object exists in schema version 56 or greater.

6.1.1.2.7.75 Validated-MS-DS-Behavior-Version

This object is present in AD DS only.

name: Validated-MS-DS-Behavior-Version

rightsGuid: d31a8757-2447-4545-8081-3bb610cacbf2

appliesTo: f0f8ffab-1191-11d0-a060-00aa006c33ed

This object exists in schema version 56 or greater.

6.1.1.2.7.76 DS-Clone-Domain-Controller

This object is present in AD DS only.

name: DS-Clone-Domain-Controller

rightsGuid: 3e0f7e18-2c7a-4c10-ba82-4d926db99a3e

appliesTo: 19195a5b-6da0-11d0-afd3-00c04fd930c9

This object exists in schema version 56 or greater.

6.1.1.2.7.77 Certificate-AutoEnrollment

This object is present in AD DS only.

name: Certificate-AutoEnrollment

rightsGuid: a05b8cc2-17bc-4802-a710-e7c15ab866a2

appliesTo: e5209ca2-3bba-11d2-90cc-00c04fd91ab1

This object exists in schema version 56 or greater.

6.1.1.2.7.78 DS-Read-Partition-Secrets

name: DS-Read-Partition-Secrets

rightsGuid: 084c93a2-620d-4879-a836-f0ae47de0e89

appliesTo: 26f11b08-a29d-4869-99bb-ef0b99fd883e

This object exists in schema version 69 or greater.

6.1.1.2.7.79 DS-Write-Partition-Secrets

name: DS-Write-Partition-Secrets

rightsGuid: 084c93a2-620d-4879-a836-f0ae47de0e89

appliesTo: 26f11b08-a29d-4869-99bb-ef0b99fd883e

This object exists in schema version 69 or greater.

6.1.1.2.7.80 DS-Set-Owner

name: DS-Set-Owner

rightsGuid: 4125c71f-7fac-4ff0-bcb7-f09a41325286

appliesTo: 26f11b08-a29d-4869-99bb-ef0b99fd883e

This object exists in schema version 69 or greater.

6.1.1.2.7.81 DS-Bypass-Quota

name: DS-Bypass-Quota

rightsGuid: 88a9933e-e5c8-4f2a-9dd7-2527416b8092

appliesTo: 26f11b08-a29d-4869-99bb-ef0b99fd883e

This object exists in schema version 69 or greater.

6.1.1.2.8 Forest Updates Container

The Forest Updates container includes child containers that specify the version of the forest revision. Some or all of the following containers exist, depending on the forest revision.

Container	Minimum forest revision for which the container exists
Operations	0.9
Windows2003Update	0.9
ActiveDirectoryUpdate	2.9

If the version of the RODC revision is 2 or higher, the Forest Updates container includes the child container ActiveDirectoryRodcUpdate.

The major version of the forest revision is stored on the revision attribute of the ActiveDirectoryUpdate container. If the ActiveDirectoryUpdate container does not exist, the major version is 0. After a forest revision upgrade process, it must be equal to the major version of the current revision.

The minor version of the forest revision is stored on the revision attribute of the Windows2003Update container. If the Windows2003Update container does not exist, the minor version is 0. After a forest revision upgrade process, it must be equal to the minor version of the current revision.

The version of the RODC revision is stored on the revision attribute of the ActiveDirectoryRodcUpdate container. If the ActiveDirectoryRodcUpdate container does not exist, the version is 0. After an RODC revision upgrade process, it must be equal to the version of the current revision.

parent: Config NC root object

name: ForestUpdates

objectClass: container

6.1.1.2.8.1 Operations Container

The contents of the Operations container are outside the state model and are implementation-specific.

parent: Forest Updates container

name: Operations

objectClass: container

6.1.1.2.8.2 Windows2003Update Container

This container stores the minor version of the forest revision.

parent: Forest Updates container

name: Windows2003Update

objectClass: container

revision: The minor version of the forest revision.

6.1.1.2.8.3 ActiveDirectoryUpdate Container

This container stores the major version of the forest revision.

parent: Forest Updates container

name: ActiveDirectoryUpdate

objectClass: container

revision: The major version of the forest revision.

6.1.1.2.8.4 ActiveDirectoryRodcUpdate Container

This container stores the version of the RODC revision.

parent: Forest Updates container

name: ActiveDirectoryRodcUpdate

objectClass: container

revision: The version of the RODC revision.

6.1.1.3 Critical Domain Objects

References

- FSMO Roles
- Forest Requirements
- Security
- Originating Updates
- LDAP

Attribute Syntaxes: DN-Binary

Glossary Terms: NC, NC Replica, NC root, DC, Domain NC, FSMO, Forest, UUID, SPN, PDC, RID

LDAP attributes: name, objectClass, distinguishedName, systemFlags, primaryGroupID, servicePrincipalName, dNSHostName, msDS-AdditionalDnsHostName, wellKnownObjects, isDeleted, revision

LDAP classes: computer, container, msDS-QuotaContainer, infrastructureUpdate, organizationalUnit, domainPolicy, samServer

WKGuids: GUID_USERS_CONTAINER_W, GUID_COMPUTERS_CONTAINER_W, GUID_SYSTEMS_CONTAINER_W, GUID_DOMAIN_CONTROLLERS_CONTAINER_W, GUID_INFRASTRUCTURE_CONTAINER_W, GUID_DELETED_OBJECTS_CONTAINER_W, GUID_LOSTANDFOUND_CONTAINER_W, GUID_FOREIGNSECURITYPRINCIPALS_CONTAINER_W, GUID_PROGRAM_DATA_CONTAINER_W, GUID_NTDS_QUOTAS_CONTAINER_W

Constants

- systemFlags bits: FLAG_DISALLOW_DELETE, FLAG_DOMAIN_DISALLOW_RENAME, FLAG_DOMAIN_DISALLOW_MOVE
- userAccountControl bits: ADS_UF_SERVER_TRUST_ACCOUNT, ADS_UF_TRUSTED_FOR_DELEGATION
- groupType bits: GROUP_TYPE_RESOURCE_GROUP, GROUP_TYPE_SECURITY_ENABLED, GROUP_TYPE_ACCOUNT_GROUP

6.1.1.3.1 Domain Controller Object

In AD DS, each normal (not read-only) DC in a domain has a domain controller object in its default NC. The DC's domain controller object is the DC's computer object (subject to the computer object

constraints specified in [MS-SAMR] sections 3.1.1.6 and 3.1.1.8) with additional requirements as described in this section.

An AD DS RODC has a read-only domain controller object as specified in section 6.1.1.3.2. An AD LDS DC does not have a domain controller object.

objectClass: computer

userAccountControl: {ADS_UF_SERVER_TRUST_ACCOUNT | ADS_UF_TRUSTED_FOR_DELEGATION}

primaryGroupID: Contains the value 516.

This attribute is populated by the system during creation of the DC corresponding to the DC object. The primary group of a DC object is the domain relative well-known Domain Controllers security group. So the primaryGroupID attribute of a DC object equals the RID of the Domain Controllers security group, 516.

servicePrincipalName: This attribute contains all of the SPNs for a normal (not read-only) DC, as specified in [MS-DRSR] section 2.2.2.

dNSHostName: Fully qualified DNS name of the DC.

msDS-AdditionalDnsHostName: Additional DNS names by which the DC can be identified.

objectCategory: Contains the distinguished name of the classSchema object for the computer class. This is the value of the defaultObjectCategory attribute of the computer class.

6.1.1.3.2 Read-Only Domain Controller Object

Each RODC in a domain has a read-only DC object in its default NC. The DC's RODC object is the DC's computer object (subject to the computer object constraints specified in [MS-SAMR] sections 3.1.1.6 and 3.1.1.8) with additional requirements as described in this section. An RODC object cannot be created on Windows 2000 Server operating system or Windows Server 2003 operating system DCs and cannot be created until the Read-Only Domain Controllers Object exists in the domain.

objectClass: computer

userAccountControl: {ADS_UF_PARTIAL_SECRETS_ACCOUNT | ADS_UF_WORKSTATION_TRUST_ACCOUNT}

primaryGroupID: Contains the value 521.

This attribute is populated during creation of the RODC corresponding to the RODC object. The primary group of an RODC object is the domain relative well-known RODCs security group. So the primaryGroupID attribute of an RODC object equals the RID of the RODCs security group, 521.

servicePrincipalName: This attribute contains all of the SPNs for the RODC, as specified in [MS-DRSR] section 2.2.2.

dNSHostName: Fully qualified DNS name of the RODC.

msDS-AdditionalDnsHostName: Additional DNS names by which the RODC can be identified.

msDS-RevealedUsers: Contains information about the user objects whose secret attributes are cached at this RODC. This attribute is maintained by the system; see procedure UpdateRevealedList, [MS-DRSR] section 4.1.10.5.9. A more usable form of this attribute is the constructed attribute msDS-RevealedList, specified in section 3.1.1.4.5.34.

msDS-AuthenticatedToAccountlist: Contains a list of user objects that have attempted to authenticate at this RODC. This attribute is a back link attribute whose corresponding forward link is the msDS-AuthenticatedAtDC attribute. The msDS-AuthenticatedAtDC attribute is maintained by the system; see section 6.1.4.6.

msDS-NeverRevealGroup: This attribute is maintained by an administrator. It contains a set of user and security-enabled group objects. A user in this set, or reachable from this set by traversing any number of member links from a group in this set, will not change state from not being cached to being cached at this RODC. If a user is added to this attribute (directly or indirectly) while one of its secret attributes is already cached, the secret attribute remains cached until the secret attribute changes, at which time the caching stops. For the use of this attribute, see procedure RevealSecretsForUserAllowed, [MS-DRSR] section 4.1.10.5.15.

msDS-RevealOnDemandGroup: This attribute is maintained by an administrator. It contains a set of user and security-enabled group objects. A user in this set, or reachable from this set by traversing any number of member links from a group in this set, and not excluded by membership in msDS-NeverRevealGroup can change state from not being cached to being cached at this RODC. For the use of this attribute see procedure RevealSecretsForUserAllowed, [MS-DRSR] section 4.1.10.5.15.

msDS-KrbTgtLink: This attribute is populated during creation of the RODC object. It contains a reference to the RODC's secondary Kerberos ticket-granting ticket account. See [MS-KILE] section 3.1.5.10.

managedBy: If the value of this attribute points to a valid security principal, that security principal will be an implicit member of the administrators group of this RODC. This applies to this RODC only.

objectCategory: Contains the distinguished name of the classSchema object for the computer class. This is the value of the defaultObjectCategory attribute of the computer class.

6.1.1.4 Well-Known Objects

Within each NC (excluding the schema NC), there are certain well-known system objects that can be referred to using a well-known GUID (see section 3.1.1.3 for more information). Domain and config NC root objects contain an attribute called wellKnownObjects that lists the well-known objects (WKO) within that NC. Each value in this list is an Object(DN-Binary) value where the Binary portion is the well-known GUID in binary form and the DN portion is the DN of the object. The well-known GUID can be used in conjunction with the NC DN to refer to the object (for more information, see section 3.1.1.3). In addition to the **wellKnownObjects** attribute, each NC root object may also contain an attribute called **otherWellKnownObjects** that lists other WKOs. Objects listed in the attribute **otherWellKnownObjects** can be referred to in the same way as those in the attribute **wellKnownObjects**.

The following requirements apply to the wellKnownObjects attribute on the NC root object and the referred-to objects, but do not apply to the **otherWellKnownObjects** attribute:

- For each of the well-known GUIDs listed below for a given NC, the wellKnownObjects attribute on the NC root object MUST contain a value such that the binary portion matches the well-known GUID. There MUST be exactly one such value.
- If rename of the referred-to object is permitted (based on the value of the systemFlags attribute on each object), the DN portion of the value is updated.
- The well-known Users container and the well-known Computers container in the domain NC may be redirected, under the following constraints:
  - The modification is made on a DC that owns the PDC FSMO.

- The modification removes the reference to the existing object and adds a new reference in the same operation.
- The new object being referred to is not in the System container of the domain NC.
- The new object being referred to does exist, and if different from the currently referred-to Users or Computers containers, it does not have the following bits in the systemFlags attribute: FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE
- As part of the redirection, the following flags are added to the new object being referred to and removed from the old object: FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE

In AD DS, the following well-known objects exist within each domain NC.

RDN	Symbolic name for well-known GUID
Computers	GUID_COMPUTERS_CONTAINER_W
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
Domain Controllers	GUID_DOMAIN_CONTROLLERS_CONTAINER_W
ForeignSecurityPrincipals	GUID_FOREIGNSECURITYPRINCIPALS_CONTAINER_W
Infrastructure	GUID_INFRASTRUCTURE_CONTAINER_W
LostAndFound	GUID_LOSTANDFOUND_CONTAINER_W
Microsoft[Note 1]	GUID_MICROSOFT_PROGRAM_DATA_CONTAINER_W
NTDS Quotas	GUID_NTDS_QUOTAS_CONTAINER_W
Program Data	GUID_PROGRAM_DATA_CONTAINER_W
System	GUID_SYSTEMS_CONTAINER_W
Users	GUID_USERS_CONTAINER_W

[Note 1] The Microsoft container is a child of the Program Data container.

In AD DS, the following well-known objects exist within each application NC.

RDN	Symbolic name for well-known GUID
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
Infrastructure	GUID_INFRASTRUCTURE_CONTAINER_W
LostAndFound	GUID_LOSTANDFOUND_CONTAINER_W
NTDS Quotas	GUID_NTDS_QUOTAS_CONTAINER_W

In AD DS, the following well-known objects exist within the config NC.

RDN	Symbolic name for well-known GUID
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
LostAndFoundConfig	GUID_LOSTANDFOUND_CONTAINER_W
NTDS Quotas	GUID_NTDS_QUOTAS_CONTAINER_W

In AD LDS, the following well-known objects exist within each application NC.

RDN	Symbolic name for well-known GUID
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
ForeignSecurityPrincipals[Note 2]	GUID_FOREIGNSECURITYPRINCIPALS_CONTAINER_W
LostAndFound	GUID_LOSTANDFOUND_CONTAINER_W
NTDS Quotas	GUID_NTDS_QUOTAS_CONTAINER_W
Roles	GUID_USERS_CONTAINER_W

[Note 2] The ForeignSecurityPrincipals container is created (and the corresponding value created in the wellKnownObjects attribute) when the first foreignSecurityPrincipal object is created in the NC.

In AD LDS, the following well-known objects exist within the config NC.

RDN	Symbolic name for well-known GUID
Deleted Objects	GUID_DELETED_OBJECTS_CONTAINER_W
ForeignSecurityPrincipals	GUID_FOREIGNSECURITYPRINCIPALS_CONTAINER_W
LostAndFoundConfig	GUID_LOSTANDFOUND_CONTAINER_W
NTDS Quotas	GUID_NTDS_QUOTAS_CONTAINER_W
Roles	GUID_USERS_CONTAINER_W

The following other well-known object exists within each domain NC.

RDN	Symbolic name for well-known GUID
Managed Service Accounts	GUID_MANAGED_SERVICE_ACCOUNTS_CONTAINER_W

The following table gives the GUID values for each of the symbolic names of the well-known GUIDs.

Symbolic name for well-known GUID	GUID
GUID_COMPUTERS_CONTAINER_W	AA312825768811D1ADED00C04FD8D5CD
GUID_DELETED_OBJECTS_CONTAINER_W	18E2EA80684F11D2B9AA00C04F79F805
GUID_DOMAIN_CONTROLLERS_CONTAINER_W	A361B2FFFFD211D1AA4B00C04FD7D83A
GUID_FOREIGNSECURITYPRINCIPALS_CONTAINER_W	22B70C67D56E4EFB91E9300FCA3DC1AA

Symbolic name for well-known GUID	GUID
GUID_INFRASTRUCTURE_CONTAINER_W	2FBAC1870ADE11D297C400C04FD8D5CD
GUID_LOSTANDFOUND_CONTAINER_W	AB8153B7768811D1ADED00C04FD8D5CD
GUID_MICROSOFT_PROGRAM_DATA_CONTAINER_W	F4BE92A4C777485E878E9421D53087DB
GUID_NTDS_QUOTAS_CONTAINER_W	6227F0AF1FC2410D8E3BB10615BB5B0F
GUID_PROGRAM_DATA_CONTAINER_W	09460C08AE1E4A4EA0F64AEE7DAA1E5A
GUID_SYSTEMS_CONTAINER_W	AB1D30F3768811D1ADED00C04FD8D5CD
GUID_USERS_CONTAINER_W	A9D1CA15768811D1ADED00C04FD8D5CD
GUID_MANAGED_SERVICE_ACCOUNTS_CONTAINER_W	1EB93889E40C45DF9F0C64D23BBB6237

6.1.1.4.1 Lost and Found Container

Each domain NC, application NC, and config NC contains a Lost and Found container for objects that are orphaned as a result of Add and Delete operations that originated on different DCs.

objectClass: lostAndFound

systemFlags: On domain and application NCs: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

On Config NC: {FLAG_DISALLOW_DELETE}

isCriticalSystemObject: TRUE

6.1.1.4.2 Deleted Objects Container

Each domain NC and application NC, as well as the config NC, contains a Deleted Objects container. Objects within the domain NC that are deleted are stored in this container (unless indicated otherwise by the object's systemFlags).

Tombstones and recycled-objects are stored until at least an amount of time equal to the tombstone lifetime has passed, after which they are permanently removed from storage.

Deleted-objects are stored until at least an amount of time equal to the deleted-object lifetime has passed, after which they are transformed into recycled-objects.

To ensure that this container does not get garbage collected, the replication metadata for the isDeleted attribute must show that the time at which the isDeleted attribute was set to true is 9999-12-29. Furthermore, the isRecycled attribute must have no values. See section 3.1.1.5.5 for more information about the tombstone lifetime, the deleted-object lifetime, and the Deleted Objects container.

objectClass: container

isDeleted: true

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

isCriticalSystemObject: TRUE

6.1.1.4.3 NTDS Quotas Container

Each domain NC, application NC, and the config NC contain an NTDS Quotas Container that contains quotas restricting the number of objects that can be created by a specified security principal.

objectClass: msDS-QuotaContainer

systemFlags:{FLAG_DISALLOW_DELETE}

isCriticalSystemObject: TRUE

msDS-DefaultQuota: Specifies the default object creation quota for security principles. By default this attribute is not set. See section 3.1.1.5.2.5 for details.

6.1.1.4.4 Infrastructure Object

In AD DS, each domain and application NC has an infrastructure object that maintains a reference to the current Infrastructure role owner. This object is not present in AD LDS.

objectClass: infrastructureUpdate

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

fSMORoleOwner: This value refers to the nTDSDSA object of the DC that owns the Infrastructure FSMO role.

isCriticalSystemObject: TRUE

6.1.1.4.5 Domain Controllers OU

This is a well-known container within the domain NC containing the computer objects for domain controllers within this domain.

objectClass: organizationalUnit

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

isCriticalSystemObject: TRUE

6.1.1.4.6 Users Container

Each domain NC contains a well-known default Users container.

objectClass: container

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

isCriticalSystemObject: TRUE

6.1.1.4.7 Computers Container

Each domain NC contains a well-known default Computers container.

objectClass: container

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

isCriticalSystemObject: TRUE

6.1.1.4.8 Program Data Container

Each domain NC contains a well-known default Program Data container. This container initially contains a single object of class container named "Microsoft". This protocol does not constrain the applications that store data in these containers, nor the application-specific data that is stored, beyond the normal access control and schema validation that is applied to all data access.

name: Program Data

parent: domain NC root

objectClass: container

systemFlags: {}

6.1.1.4.9 Managed Service Accounts Container

In AD DS, each domain NC contains this container. This container is not present in AD LDS.

name: Managed Service Accounts

parent: domain NC root

objectClass: container

systemFlags: {}

6.1.1.4.10 Foreign Security Principals Container

In AD DS, each domain NC contains a well-known Foreign Security Principals container. This container holds objects of class foreignSecurityPrincipal. These objects represent security principals from trusted domains external to the forest, and allow foreign security principals to become members of groups within the domain.

In AD LDS, the config NC contains a well-known Foreign Security Principals container. It stores foreign security principals from outside of the AD LDS forest.

In an AD LDS application NC, a Foreign Security Principals container is created (and the corresponding value created in the wellKnownObjects attribute) when the first foreignSecurityPrincipal object is created in the application NC.

The automatic creation of foreignSecurityPrincipal objects is specified in sections 3.1.1.5.2.4 and 3.1.1.5.3.3).

name: ForeignSecurityPrincipals

parent: domain NC root on AD DS; config NC root on AD LDS.

objectClass: container

systemFlags (on AD DS): {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

systemFlags (on AD LDS): {FLAG_DISALLOW_DELETE}

isCriticalSystemObject: TRUE

6.1.1.4.11 System Container

name: System

parent: Domain NC root object

objectClass: container

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

isCriticalSystemObject: TRUE

6.1.1.4.11.1 Password Settings Container

In AD DS, each domain NC contains a well-known Password Settings container. This container is initially empty, but is designed to contain objects of class msDS-PasswordSettings. These objects represent password settings for a group of users in the domain. For more information, see [MS-SAMR] section 3.1.1.5.

name: Password Settings container

parent: System container

objectClass: msDS-PasswordSettings

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

6.1.1.4.12 Builtin Container

In AD DS, each domain NC contains this container. Its children are described later in this section. This container is not present in AD LDS.

name: Builtin

parent: domain NC root

objectClass: builtinDomain

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

The children of the Builtin container are well-known security principals from the built-in domain.

Each child of the Builtin container is a group with the following attributes:

parent: Builtin container

objectClass: group

objectSid: The domain portion is the built-in domain SID (S-1-5-32). The RID portion is specified per object in the following subsections. For instance, the Account Operators RID is 548, so the Account Operators objectSid is S-1-5-32-548.

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

groupType: {GROUP_TYPE_BUILTIN_LOCAL_GROUP | GROUP_TYPE_RESOURCE_GROUP | GROUP_TYPE_SECURITY_ENABLED}

Unless otherwise noted in the following subsections, the initial membership of each group is empty. After initialization, the administrator controls the membership of each group.

6.1.1.4.12.1 Account Operators Group Object

name: Account Operators

RID: 548

6.1.1.4.12.2 Administrators Group Object

name: Administrators

RID: 544

member: Administrator (section 6.1.1.6.1), Domain Administrators (section 6.1.1.6.5), Enterprise Administrators (section 6.1.1.6.10).

6.1.1.4.12.3 Backup Operators Group Object

name: Backup Operators

RID: 551

6.1.1.4.12.4 Certificate Service DCOM Access Group Object

name: Certificate Service DCOM Access

RID: 574

6.1.1.4.12.5 Cryptographic Operators Group Object

name: Cryptographic Operators

RID: 569

6.1.1.4.12.6 Distributed COM Users Group Object

name: Distributed COM Users

RID: 562

6.1.1.4.12.7 Event Log Readers Group Object

name: Event Log Readers

RID: 573

6.1.1.4.12.8 Guests Group Object

name: Guests

RID: 546

member: Guest (section 6.1.1.6.2), Domain Guests (section 6.1.1.6.8)

6.1.1.4.12.9 IIS_IUSRS Group Object

name: IIS_IUSRS

RID: 568

member: NT AUTHORITY\IUSR well-known security principal (SID S-1-5-17).

6.1.1.4.12.10 Incoming Forest Trust Builders Group Object

name: Incoming Forest Trust Builders

RID: 557

6.1.1.4.12.11 Network Configuration Operators Group Object

name: Network Configuration Operators

RID: 556

6.1.1.4.12.12 Performance Log Users Group Object

name: Performance Log Users

RID: 559

6.1.1.4.12.13 Performance Monitor Users Group Object

name: Performance Monitor Users

RID: 558

6.1.1.4.12.14 Pre-Windows 2000 Compatible Access Group Object

name: Pre-Windows 2000 operating system Compatible Access

RID: 554

member: The initial membership of this group depends on the version of Windows running on the first DC of the domain and on the administrator's choice between "Pre-Windows 2000 Compatible Permissions mode" and "Windows 2000-Only Permissions mode". In Windows 2000 Server operating system, in the Pre-Windows 2000 Compatible Permissions mode, Everyone (S-1-1-0) is a member, and in the Windows 2000-Only Permissions mode, the membership is empty. In Windows Server 2003 operating system, in the Pre-Windows 2000 Compatible Permissions mode, Everyone (S-1-1-0) and Anonymous (S-1-5-7) are members, and in the Windows 2000-Only Permissions mode, Authenticated Users (S-1-5-11) are members.

6.1.1.4.12.15 Print Operators Group Object

name: Print Operators

RID: 550

6.1.1.4.12.16 Remote Desktop Users Group Object

name: Remote Desktop Users

RID: 555

6.1.1.4.12.17 Replicator Group Object

name: Replicator

RID: 552

6.1.1.4.12.18 Server Operators Group Object

name: Server Operators

RID: 549

6.1.1.4.12.19 Terminal Server License Servers Group Object

name: Terminal Server License Servers

RID: 561

6.1.1.4.12.20 Users Group Object

name: Users

RID: 545

member: Domain Users group (section 6.1.1.6.9), NT AUTHORITY\Authenticated Users well-known security principal (SID S-1-5-11), NT AUTHORITY\INTERACTIVE well-known security principal (SID S-1-5-4).

6.1.1.4.12.21 Windows Authorization Access Group Group Object

name: Windows Authorization Access Group

RID: 560

member: NT AUTHORITY\ENTERPRISE DOMAIN CONTROLLERS well-known security principal (SID S-1-5-9).

6.1.1.4.13 Roles Container

In AD LDS, each application NC and the config NC contain this container. It stores the well-known AD LDS groups for this NC. This container is not present in AD DS, nor are any of its child objects, which are specified later in this section.

name: Roles

parent: Application NC root or config NC root

objectClass: container

systemFlags: {FLAG_DISALLOW_DELETE}

Each child of the Roles container is a group with the following attributes:

parent: Roles Container

objectClass: group

objectSid: A SID with two SubAuthority values, consisting of the objectSid of the NC root followed by the RID that is specified for each child in the following subsections.

groupType: {GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED}

member: Unless otherwise noted in the following sections, the initial membership of each group is empty. After initialization the administrator can modify the membership of each group.

6.1.1.4.13.1 Administrators Group Object

name: Administrators

RID: 519 (in the config NC) or 512 (in an application NC).

member: At least one foreignSecurityPrincipal is configured into this group by the administrator when creating a forest.

6.1.1.4.13.2 Readers Group Object

name: Readers

RID: 514

6.1.1.4.13.3 Users Group Object

This group is used in constructing an AD LDS security context as specified in section 5.1.3.4.

name: Users

RID: 513

6.1.1.4.13.4 Instances Group Object

In AD LDS, every DC's service account belongs to this group. The system attempts to maintain this group, although an administrator can still modify the membership. This group is only present in the Roles container of the config NC.

name: Instances

RID: 518

member: An AD LDS DC ensures that its service account is a member of this group. If an AD LDS DC's service account is Network Service or Local System, the DC also ensures that its computer object is a member of this group.

6.1.1.5 Other System Objects

The following sections describe other objects that are required by Active Directory, in addition to those listed in section 6.1.1.4.

6.1.1.5.1 AdminSDHolder Object

parent: System container

name: AdminSDHolder

objectClass: container

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

isCriticalSystemObject: TRUE

nTSecurityDescriptor: The default value of nTSecurityDescriptor for AdminSDHolder depends on the schema version (see section 3.1.1.2). In the following text, the value of the nTSecurityDescriptor is specified using SDDL ([MS-DTYP] section2.5.1).

```
Schema version 13:
O:S-1-5-21-1330137634-1750626333-945493308-512G:S-1-5-21-1330137634-1750626333-945493308-
512D:PAI(A;;LCRPLORC;;;AU)(A;;CCDCLCSWRPWPLOCRSDRCWDWO;;;BA)(A;;CCDCLCSWRPWPLOCRRCWDWO;;;S-1-
5-21-1330137634-1750626333-945493308-519)(A;;CCDCLCSWRPWPLOCRRCWDWO;;;S-1-5-21-1330137634-
1750626333-945493308-512)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)(OA;;RP;037088f8-0ae1-11d2-b422-
00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;59ba2f42-79a2-11d0-9020-
00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;4c164200-20c0-11d0-a768-
00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;LCRPLORC;;bf967aba-0de6-11d0-a285-
00aa003049e2;RU)(OA;;CR;ab721a53-1e2f-11d0-9819-
00aa0040529b;;WD)S:AI(AU;CIIDSAFA;CCDCSWWPDTCRSDWDWO;;;WD)
Schema version 30, Schema version 31:
O:DAG:DAD:PAI(A;;LCRPLORC;;;AU)(A;;CCDCLCSWRPWPLOCRSDRCWDWO;;;BA)(A;;CCDCLCSWRPWPLOCRRCWDWO;;
;EA)(A;;CCDCLCSWRPWPLOCRRCWDWO;;;DA)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)(OA;;RP;037088f8-
0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;59ba2f42-79a2-
11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;bc0ac240-79a9-11d0-
9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;4c164200-20c0-11d0-a768-
00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)(OA;;LCRPLORC;;bf967aba-0de6-11d0-a285-
00aa003049e2;RU)(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)(OA;;CR;ab721a53-1e2f-11d0-
9819-00aa0040529b;;PS)(OA;;RPWP;bf967a7f-0de6-11d0-a285-00aa003049e2;;CA)(OA;;RP;037088f8-
0ae1-11d2-b422-00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;59ba2f42-79a2-
11d0-9020-00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;bc0ac240-79a9-11d0-
9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;4c164200-20c0-11d0-a768-
00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)(OA;;LCRPLORC;;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU)(OA;;RP;46a9b11d-60ae-405a-b7e8-ff8a58d456d2;;S-1-5-32-
560)(OA;;RPWP;6db69a1c-9422-11d1-aebd-0000f80367c1;;S-1-5-32-
561)S:AI(AU;SA;WPWDWO;;;WD)(OU;CIIOIDSA;WP;f30e3bbe-9ff0-11d1-b603-0000f80367c1;bf967aa5-
0de6-11d0-a285-00aa003049e2;WD)(OU;CIIOIDSA;WP;f30e3bbf-9ff0-11d1-b603-0000f80367c1;bf967aa5-
0de6-11d0-a285-00aa003049e2;WD)

Schema version 44, Schema version 47:
O:DAG:DAD:PAI(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU)(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-
00aa003049e2;RU)(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU)(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-
```

```
00aa003049e2;RU)(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU)(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-
00aa003049e2;RU)(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU)(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-
00aa003049e2;RU)(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU)(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-
00aa003049e2;RU)(OA;;RPWP;bf967a7f-0de6-11d0-a285-00aa003049e2;;CA)(OA;;RP;46a9b11d-60ae-
405a-b7e8-ff8a58d456d2;;S-1-5-32-560)(OA;;RPWP;6db69a1c-9422-11d1-aebd-0000f80367c1;;S-1-5-
32-561)(OA;;RPWP;5805bc62-bdc9-4428-a5e2-856a0f4c185e;;S-1-5-32-561)(OA;;LCRPLORC;;4828cc14-
1437-45bc-9b07-ad6f015e5f28;RU)(OA;;LCRPLORC;;bf967aba-0de6-11d0-a285-
00aa003049e2;RU)(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)(OA;;CR;ab721a53-1e2f-11d0-
9819-00aa0040529b;;PS)(OA;CI;RPWPCR;91e647de-d96f-4b70-9557-
d63ff4f3ccd8;;PS)(A;;CCDCLCSWRPWPLOCRRCWDWO;;;DA)(A;;CCDCLCSWRPWPLOCRRCWDWO;;;EA)(A;;CCDCLCSW
RPWPLOCRSDRCWDWO;;;BA)(A;;LCRPLORC;;;AU)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)S:AI(AU;SA;WPWDWO
;;;WD)(OU;CIIOIDSA;WP;f30e3bbe-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-
00aa003049e2;WD)(OU;CIIOIDSA;WP;f30e3bbf-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-
00aa003049e2;WD)
```

6.1.1.5.2 Default Domain Policy Container

This container is not necessary for Active Directory functioning, and this protocol does not define any constraints beyond those listed in this section. This container is used by the **Group Policy** System ([MS-GPOD] section 1.1.4).

parent: System container

name: Default Domain Policy

objectClass: domainPolicy

isCriticalSystemObject: TRUE

6.1.1.5.3 Sam Server Object

parent: System container

name: Server

objectClass: samServer

systemFlags: {FLAG_DISALLOW_DELETE | FLAG_DOMAIN_DISALLOW_RENAME | FLAG_DOMAIN_DISALLOW_MOVE}

Note Domain controllers running Windows Server 2012 operating system and Windows Server 2012 R2 operating system do not create the systemFlags attribute on the Sam Server object.

6.1.1.5.4 Domain Updates Container

The Domain Updates container includes child containers that specify the version of the domain revision. Some or all of the following containers exist, depending on the domain revision.

Container	Minimum domain revision for which the container exists
Operations	0.8
Windows2003Update	0.8
ActiveDirectoryUpdate	3.9

The version of the revision is stored under the Domain Updates child containers.

The major version is stored on the revision attribute of the ActiveDirectoryUpdate container. If the ActiveDirectoryUpdate container does not exist, the major version is 0. After a domain revision upgrade process, the revision attribute of the ActiveDirectoryUpdate container must be equal to the major version of the current revision.

The minor version is stored on the revision attribute of the Windows2003Update container. If the Windows2003Update container does not exist, the minor version is 0. After a domain revision upgrade process, the revision attribute of the Windows2003Update container must be equal to the minor version of the current revision.

parent: System container

name: DomainUpdates

objectClass: container

6.1.1.5.4.1 Operations Container

The contents of the Operations container are outside the state model and are implementation-specific.

parent: Domain Updates container

name: Operations

objectClass: container

6.1.1.5.4.2 Windows2003Update Container

This container stores the minor version of the domain revision.

parent: Domain Updates container

name: Windows2003Update

objectClass: container

revision: The minor version of the domain revision.

6.1.1.5.4.3 ActiveDirectoryUpdate Container

This container stores the major version of the domain revision.

parent: Domain Updates container

name: ActiveDirectoryUpdate

objectClass: container

revision: The major version of the domain revision.

6.1.1.6 Well-Known Domain-Relative Security Principals

In each domain NC, there are certain well-known security principals. These well-known security principals are given default privileges in the domain. For more information, see section 5 and also see [MS-SAMR] section 3.1.4.2.

Each of these objects has the following attributes:

parent: Users container (section 6.1.1.4.6).

objectSid: A SID consisting of the objectSid of the domain NC root, followed by the RID that is specified for each child in the following subsections.

The objects of class user have the following attribute:

primaryGroupID: This value is a RID, which refers to another well-known domain relative security principal.

6.1.1.6.1 Administrator

name: Administrator

objectClass: user

RID: 500

primaryGroupID: 513 (Domain Users)

6.1.1.6.2 Guest

name: Guest

objectClass: user

RID: 501

primaryGroupID: 514 (Domain Guests)

6.1.1.6.3 Key Distribution Center Service Account

name: krbtgt

objectClass: user

RID: 502

primaryGroupID: 513 (Domain Users)

6.1.1.6.4 Cert Publishers

name: Cert Publishers

objectClass: group

RID: 517

groupType: {GROUP_TYPE_RESOURCE_GROUP | GROUP_TYPE_SECURITY_ENABLED}

6.1.1.6.5 Domain Administrators

name: Domain Admins

objectClass: group

RID: 512

groupType: {GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED}

6.1.1.6.6 Domain Computers

name: Domain Computers

objectClass: group

RID: 515

groupType: {GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED}

6.1.1.6.7 Domain Controllers

name: Domain Controllers

objectClass: group

RID: 516

groupType: {GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED}

6.1.1.6.8 Domain Guests

name: Domain Guests

objectClass: group

RID: 514

groupType: {GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED}

6.1.1.6.9 Domain Users

name: Domain Users

objectClass: group

RID: 513

groupType: { GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED }

6.1.1.6.10 Enterprise Administrators

This group exists only in the forest root domain.

name: Enterprise Admins

objectClass: group

RID: 519

groupType:

- If the forest root domain is mixed (section 6.1.4.1):
  - {GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED}
- If the forest root domain is not mixed:
  - {GROUP_TYPE_UNIVERSAL_GROUP | GROUP_TYPE_SECURITY_ENABLED}

6.1.1.6.11 Group Policy Creator Owners

name: Group Policy Creator Owners

objectClass: group

RID: 520

groupType: {GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED}

6.1.1.6.12 RAS and IAS Servers

name: RAS and IAS Servers

objectClass: group

RID: 553

groupType: {GROUP_TYPE_RESOURCE_GROUP | GROUP_TYPE_SECURITY_ENABLED}

6.1.1.6.13 Read-Only Domain Controllers

name: Read-Only Domain Controllers

objectClass: group

RID: 521

groupType: {GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED}

This group is created in a domain by the PDC the first time a Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, or Windows Server 2012 R2 operating system DC holds the PDC FSMO.

6.1.1.6.14 Enterprise Read-Only Domain Controllers

name: Enterprise Read-Only Domain Controllers

objectClass: group

RID: 498

groupType: {GROUP_TYPE_UNIVERSAL_GROUP | GROUP_TYPE_SECURITY_ENABLED}

This group is created in the root domain by the root domain PDC the first time a Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012

operating system, or Windows Server 2012 R2 operating system DC holds the root domain PDC FSMO.

6.1.1.6.15 Schema Admins

This group exists only in the forest root domain.

name: Schema Admins

objectClass: group

RID: 518

groupType:

- If the forest root domain is mixed (section 6.1.4.1):
  - { GROUP_TYPE_ACCOUNT_GROUP | GROUP_TYPE_SECURITY_ENABLED }
- If the forest root domain is not mixed:
  - { GROUP_TYPE_UNIVERSAL_GROUP | GROUP_TYPE_SECURITY_ENABLED }

6.1.1.6.16 Allowed RODC Password Replication Group

name: Allowed RODC Password Replication Group

objectClass: group

RID: 571

groupType: { GROUP_TYPE_RESOURCE_GROUP | GROUP_TYPE_SECURITY_ENABLED }

6.1.1.6.17 Denied RODC Password Replication Group

name: Denied RODC Password Replication Group

objectClass: group

RID: 572

groupType: { GROUP_TYPE_RESOURCE_GROUP | GROUP_TYPE_SECURITY_ENABLED }

6.1.2 Forest Requirements

References: nTDSDSA object, server object, Domain Controller object, SPN construction, crossRef object, NC root object

Glossary Terms: DC, NC, NC Replica

LDAP Attributes: serverReference, dNSHostName, servicePrincipalName, nCName, msDS-NC-Replica-Locations, msDS-hasMasterNCs, msDS-HasInstantiatedNCs, hasPartialReplicaNCs

LDAP Classes: nTDSDSA, server, crossRef

Constants: NTDSDS_OPT_IS_GC

6.1.2.1 DC Existence

For any DC in the forest, the following objects must exist:

- nTDSDSA object: See section 6.1.1.
- server object: See section 6.1.1.
- Domain Controller object (in AD DS, not AD LDS): See section 6.1.1.

For the purposes of this section, an RODC object is a Domain Controller object.

Any one of these objects can be said to "represent" the DC.

Relationships:

- The server object is the parent of the nTDSDSA object. On AD DS, the name of the server object is the computer name of the DC; on AD LDS, the name of the server object is the computer name, followed by "$", followed by the instance name of the DC.
- On AD DS, the attribute serverReference on the server object must reference the domain controller object.
- On AD DS, the dNSHostName attribute of the domain controller object must equal the dNSHostName attribute of the server object.
- The dNSHostName attribute of the server object must equal the DNS hostname of the computer that is physically the DC.
- On AD DS, every value of the servicePrincipalName attribute of the domain controller object, which has a DNS hostname as the instance name (see section 5.1.1.4, "Mutual Authentication", for SPN construction), should have an instance name equal to the dNSHostName of the domain controller object.

6.1.2.2 NC Existence

For any NC in the forest, the following objects must exist:

- crossRef: see section 6.1.1.
- NC root: see section 6.1.1.

Either of these objects can be said to "represent" the NC.

Relationships:

- The nCName attribute of the crossRef object must reference the NC root object.

6.1.2.3 Hosting Requirements

6.1.2.3.1 DC and Application NC Replica

A DC is instructed to host an application NC replica if:

- The attribute msDS-NC-Replica-Locations on the crossRef representing the NC contains the **DSName** of the nTDSDSA object representing the DC.

A DC is hosting an application NC replica when the following are true:

- The attribute msDS-hasMasterNCs on the nTDSDSA object representing the DC contains the **DSName** of the NC root representing the NC.
- The attribute msDS-HasInstantiatedNCs on the nTDSDSA object representing the DC contains an Object(DN-Binary) value such that the DN field is the **DSName** of the NC root representing the NC, and the Data field contains the value of the instanceType attribute on the NC root object on the DC.

6.1.2.3.2 DC and Regular Domain NC Replica

A DC is instructed to host a regular domain NC replica if:

- The domain controller object representing the DC is in the domain NC.

A DC is hosting a regular domain NC replica when the following are true:

- The attribute msDS-hasMasterNCs and attribute hasMasterNCs on the nTDSDSA object representing the DC contain the **DSName** of the NC root representing the domain NC.
- The attribute msDS-HasInstantiatedNCs on the nTDSDSA object representing the DC contains an Object(DN-Binary) value such that the DN field is the **DSName** of the domain NC root representing the domain NC, and the Data field contains the value of the instanceType attribute on the domain NC root object on the DC.
- The attribute msDS-HasDomainNCs on the nTDSDSA object representing the DC references the domain NC root. A DC hosts only one full domain NC replica.

6.1.2.3.3 DC and Schema/Config NC Replicas

Every DC is instructed to host the schema and config NC replicas.

A DC is hosting the schema and config NC replicas when the following are true:

- The attribute msDS-hasMasterNCs and attribute hasMasterNCs on the nTDSDSA object representing the DC contain the **DSName** of both the NC roots representing the schema and config NCs.
- The attribute msDS-HasInstantiatedNCs on the nTDSDSA object representing the DC contains two Object(DN-Binary) values such that the DN fields are the **DSName** of the NC root representing the config and schema NCs, and the binary fields contain the values of the instanceType attribute on the config and schema NC root objects on the DC.

6.1.2.3.4 DC and Partial Replica NCs Replicas

A DC is instructed to host a partial NC replica of every domain NC in the forest if:

- The options attribute of the nTDSDSA object representing that DC has the following flag: NTDSDS_OPT_IS_GC.

A DC hosts a partial NC replica of a domain NC when the following are true:

- The attribute hasPartialReplicaNCs on the nTDSDSA object representing the DC contains the **DSName** of the NC roots representing the domain NC.
- The attribute msDS-HasInstantiatedNCs on the nTDSDSA object representing the DC contains an Object(DN-Binary) value such that the DN field is the **DSName** of the NC root representing the

NC, and the Data field contains the value of the instanceType attribute on the NC root object on the DC.

6.1.3 Security Descriptor Requirements

Constants

- LDAP constants: LDAP_SERVER_SD_FLAGS_OID
- SD flags: OWNER_SECURITY_INFORMATION, GROUP_SECURITY_INFORMATION, DACL_SECURITY_INFORMATION, SACL_SECURITY_INFORMATION, SECURITY_PRIVATE_OBJECT
- Security access mask bits and privileges: SE_RESTORE_PRIVILEGE, RIGHT_WRITE_DAC, RIGHT_WRITE_OWNER, ACCESS_SYSTEM_SECURITY.
- Security descriptor values stored in Active Directory are in SECURITY_DESCRIPTOR format (see [MS-DTYP] section 2.4.6). In addition to the defined fields, the **RM Control** (Resource Manager Control) field is used. It is stored in the Sbz1 byte of the SECURITY_DESCRIPTOR structure. The SECURITY_PRIVATE_OBJECT bit (0x01) might be present in the field.
- Error codes: ERROR_INVALID_OWNER

The following requirements apply to SDs that are maintained by a DC:

1. Each object's SD retains the set of explicit (noninherited) ACEs stamped in its DACL and SACL (if present). It also retains the owner and group SID values as well as various SD flags (see SD reference [MS-DTYP] section 2.4.6). The owner SID may not be NULL, while the group SID may be NULL.
2. The SD also includes the set of inheritable ACEs from its parent object. It includes both applicable and nonapplicable inheritable ACEs. The following exceptions apply to the preceding rule:
    1. The object is the root of an NC. In this case, the SD does not include any inherited ACEs.
    2. If the ACL (either DACL or SACL) has the "protected from inheritance" flag set. In this case, the ACL does not include inheritable ACEs from the parent object's SD.
    3. The object is deleted. In this case, the set of inheritable ACEs that were obtained from the parent object's SD at the time of object deletion is retained.
3. When the forest functional level is DS_BEHAVIOR_WIN2003 or above and the fDontStandardizeSDs heuristic is false (section 6.1.1.2.4.1.2), then the ACEs in the ACLs are sorted according to ACE ordering rules (see the following ACE ordering rules section). Otherwise, if the forest functional level is less than DS_BEHAVIOR_WIN2003, the order of explicit ACEs supplied by the client is preserved.
4. The ACEs with the **inheritedObjectType** field present are marked as effective or ineffective by setting the INHERIT_ONLY_ACE flag. The INHERIT_ONLY_ACE flag identifies an ineffective ACE, which does not control access to the object to which it is attached. If this flag is not set, the ACE is an effective ACE, which controls access to the object to which it is attached. This flag is set according to SD merge rules (see the CreateSecurityDescriptor algorithm in [MS-DTYP] section 2.5.3.4.1), based on the current value of the object's objectClass attribute. Specifically, the following objectClass values are considered when processing inheritable ACEs from the parent's SD: the most specific structural objectClass value, as well as all dynamic auxiliary class values. The static auxiliary classes and non–most specific object classes are not considered. For example, in Active Directory schema, computer objects have the following objectClass values: top, person, organizationalPerson, user, and computer. In this case, only the computer class has to be

considered for inheritance processing. For inheritance processing, each effective objectClass value is converted to the GUID (as per schema mapping object classes to GUIDs; see Schema (section 3.1.1.3.1.1)) and supplied as an input to the SD merge routine.

5. In order to compute the resultant SD value for an object, the CreateSecurityDescriptor algorithm ([MS-DTYP] section 2.5.3.4.1) is invoked with the following input parameters:

    1. *ParentDescriptor*: If the object is NC root, then NULL; otherwise, the SD value of the parent object.

    2. *CreatorDescriptor*: The current SD value stamped on the object. When an LDAP add operation is performed and no SD value is supplied, the SD value is first defaulted according to the rules specified in sections 6.1.3.5 and 6.1.3.6.

    3. *IsContainerObject*: true is always passed.

    4. *AutoInheritFlags*: DACL_AUTO_INHERIT | SACL_AUTO_INHERIT.

    5. *Token*: When processing an originating SD write, the security information of the requester is used. Otherwise, SYSTEM security information is used; note that, in the case of auto-propagation into children, the information from the token is never used, because all required SD parts are always present and there is nothing that needs to be defaulted.

    6. *GenericMapping*: The following mapping table is used for all Active Directory SD operations:

        - GENERIC_READ_MAPPING = RIGHT_READ_CONTROL | RIGHT_DS_LIST_CONTENTS | RIGHT_DS_READ_PROPERTY | RIGHT_DS_LIST_OBJECT
        - GENERIC_WRITE_MAPPING = RIGHT_READ_CONTROL | RIGHT_DS_WRITE_PROPERTY_EXTENDED | RIGHT_DS_WRITE_PROPERTY
        - GENERIC_EXECUTE_MAPPING = RIGHT_READ_CONTROL | RIGHT_DS_LIST_CONTENTS
        - GENERIC_ALL_MAPPING = RIGHT_DELETE | RIGHT_READ_CONTROL | RIGHT_WRITE_DAC | RIGHT_WRITE_OWNER | RIGHT_DS_CREATE_CHILD | RIGHT_DS_DELETE_CHILD | RIGHT_DS_DELETE_TREE | RIGHT_DS_READ_PROPERTY | RIGHT_DS_WRITE_PROPERTY | RIGHT_DS_LIST_CONTENTS | RIGHT_DS_LIST_OBJECT | RIGHT_DS_CONTROL_ACCESS | RIGHT_DS_WRITE_PROPERTY_EXTENDED

6. Any CREATOR/OWNER ineffective ACE has a matching effective ACE granted to the current owner of the object (as obtained from the SD OWNER field).

7. NULL DACLs are disallowed.

6.1.3.1 ACE Ordering Rules

ACE ordering rules apply only to ACLs in canonical form (see [MS-DTYP] section 2.4.5), and only when the forest functional level is DS_BEHAVIOR_WIN2003 or above. The following rules are applied, in the following order:

1. Explicit ACEs come before inherited ACEs.

2. Deny ACEs come before Allow ACEs.

3. Regular ACEs come before object ACEs.

4. Within each group, the ACEs are ordered lexicographically (that is, based on octet string comparison rules).

Rules 3 and 4 above are enforced only when the forest functional level is DS_BEHAVIOR_WIN2003 or above. Otherwise, the order of ACEs within each group defined by rules 1 and 2 is retained as supplied by the user or replication partner.

6.1.3.2 SD Flags Control

When performing an LDAP operation (modify or search), the client may supply an SD Flags Control LDAP_SERVER_SD_FLAGS_OID (section 3.1.1.3.4.1.11) with the operation. The value of the control is an integer, which is used to identify which security descriptor (SD) parts the client intends to read or modify. When the control is not specified, then the default value of 15 (0x0000000F) is used.

The SD parts are identified using the following bit values: OWNER_SECURITY_INFORMATION, GROUP_SECURITY_INFORMATION, DACL_SECURITY_INFORMATION, SACL_SECURITY_INFORMATION, which correspond to OWNER, GROUP, DACL and SACL SD fields, respectively.

If the LDAP_SERVER_SD_FLAGS_OID control is present in an LDAP search request, the server returns an SD with the parts specified in the control when the SD attribute name is explicitly mentioned in the requested attribute list, or when the requested attribute list is empty, or when all attributes are requested ([RFC2251] section 4.5.1). Without the presence of this control, the server returns an SD only when the SD attribute name is explicitly mentioned in the requested attribute list.

For update operations, the bits identify which SD parts are affected by the operation. Note that the client may supply values for other (or all) SD fields. However, the server only updates the fields that are identified by the SD control. The remaining fields are ignored. When performing an LDAP add operation, the client can supply an SD flags control with the operation; however, it will be ignored by the server.

6.1.3.3 Processing Specifics

1. The clients may send in SD values that include both explicit and inherited ACEs (during add or modify operations). Only the set of explicit ACEs is considered authoritative data. Any inherited ACEs that are included in the SD value are ignored. Instead, the set of inherited ACEs is computed per the rules in the preceding sections and set on the object.

2. During an add operation, the DC makes sure that the object's security descriptor value is consistent with the parent's SD value (according to the preceding rules), at the moment when the add operation is committed.

3. During a move operation, the DC makes sure that the moved object's security descriptor value is consistent with the new parent's SD value (according to the preceding rules), at the moment when the move operation is committed. If the moved object has descendant objects (that is, a tree move was performed), then the SD values of the children objects are updated outside of the move transaction (see Modify DN, section 3.1.1.5.4).

4. During an SD modify operation, the DC ensures that the updated object's security descriptor value is consistent with the parent's SD value (according to the preceding rules), at the moment when the modify operation is committed. If the updated object has descendant objects, then the SD values of the children objects are updated outside of the modify transaction.

5. When processing inbound replication containing SD updates, the SD requirements are enforced (in other words, it is not guaranteed that the SD value sent by the replication partner is consistent with the parent's SD value). It is the responsibility of the DC performing the inbound replication to ensure that the set of inherited ACEs present in the SD is consistent in the subtree that is rooted at the affected object (according to the preceding rules). One exception to this rule

is when processing inbound replication of a deleted object. In this case, the DC retains the SD value (including both explicit and inherited ACEs) as it is supplied by the replication partner, in cases when it is supplied by the replication partner. If the SD value is not supplied by the replication partner, then the existing SD value is retained.

6. When an originating add operation is processed, the client may or may not supply an SD value. If the SD value is not supplied, then the DACL and SACL on the newly created object are defaulted according to the SD defaulting rules (section 6.1.3.5). If the SD value is present, then the DACL and SACL are obtained from this value. If the DACL is not present in the supplied value, then the add operation is failed with *unwillingToPerform* / *<unrestricted>* (per the preceding constraint). If the SACL is not present in the supplied value, then a NULL value is written in place of this SACL.

7. If the RM control field is present in the supplied SD value, then its value is reset to contain the SECURITY_PRIVATE_OBJECT bit, and nothing else.

8. AD LDS imposes a restriction on the security principals that can be used in an AD LDS security descriptor (owner, group, and SID values within ACEs). The SID of a security principal within an AD LDS application NC can appear in a security descriptor within that application NC, but cannot appear in a security descriptor within any other NC of the same forest. Other SIDs are not restricted, so for instance a Windows security principal is allowed in any AD LDS security descriptor, as is a security principal from another AD LDS forest, as well as a security principal from the config NC of the same AD LDS forest.

9. Microsoft Windows Server 2008 R2 operating system and above impose a restriction on modifying the OWNER field. If a modify operation attempts to set the OWNER SID to a value to which it is currently set, the operation will fail with a *constraintViolation* / *ERROR_INVALID_OWNER*. This restriction is processed before the security checks described in section 6.1.3.4.

6.1.3.4 Security Considerations

When an add operation is processed, the client is allowed to specify any SD value, subject to some constraints to the OWNER field, specified in this section.

When a modify operation is processed, the following security checks are applied to the requester's security context. If the requester does not pass the check, then *accessDenied* is returned.

1. If the DACL value is written (according to SD flags), then one of the following requirements must be satisfied:

    1. RIGHT_WRITE_DAC is granted to the requester on the object.

    2. The OWNER SID in the SD value is one of the SIDs in the requester's token (either as user SID or group SID).

2. If the OWNER and/or GROUP value is written (according to SD flags), then one of the following requirements must be satisfied:

    1. RIGHT_WRITE_OWNER is granted to the requester on the object.

    2. The requester possesses the SE_TAKE_OWNERSHIP_PRIVILEGE.

    3. The control access right DS-Set-Owner is granted to the requestor on the object that is the root of the naming context to which the object holding the SD belongs.

3. If the SACL value is written (according to SD flags), then the following requirement must be satisfied:

   - The requester possesses the SE_SECURITY_PRIVILEGE.

4. If the object being modified is in the config NC or schema NC, and the RM control of the SD is present and contains SECURITY_PRIVATE_OBJECT bit, then additional requirements on the DC performing the operation must be enforced:

   1. The DC must be a member of the root domain in the forest, or

   2. The DC must be a member of the same domain to which the current object owner belongs.

When the OWNER value is being written (via SD flags control, either in an add or a modify operation), then the following constraint must be satisfied. The value of the OWNER field must be one of the following SIDs:

1. The SID of the user performing the operation.

2. The SID of the "default administrators group" (DAG; section 6.1.3.7), only when the DAG is defined and the user is a member of this group.

3. Any SID, when the user possesses the SE_RESTORE_PRIVILEGE.

If the owner SID does not satisfy the preceding rules, then the server fails the operation, returning an *unwillingToPerform* / *ERROR_INVALID_OWNER* error.

If the owner SID is written on an object in the config NC or schema NC, then additional requirements on the DC performing the operation are enforced:

- The DC must be a member of the root domain in the forest, or
- The DC must be a member of the same domain to which the current object owner belongs.

6.1.3.5 SD Defaulting Rules

When an add operation is performed and the client does not supply an SD value, then the SD value is defaulted as follows:

1. The SD is determined from the defaultSecurityDescriptor value obtained from the classSchema object corresponding to the most specific structural objectClass of the object being created. The value of defaultSecurityDescriptor is an SDDL string. The string is converted to the binary SD value in the context of domain SID (used to resolve domain SID references, such as Domain Administrators alias) and root domain SID (used to resolve forest SID references, such as Enterprise Administrators alias). See [MS-DTYP] section 2.5.1 for more details.

2. When the object is created in an application NC, then the value or sdReferenceDomain from the crossRef corresponding to the NC is used to determine the domain SID used as context in the SDDL conversion process.

6.1.3.6 Owner and Group Defaulting Rules

The OWNER and GROUP fields are defaulted in the following scenarios:

- The SD flags do not include the OWNER bit.
- The SD flags include the OWNER bit, but the OWNER field in the supplied value is NULL.

In the preceding cases, the OWNER field is defaulted as follows:

- If the user performing the operation is a member of the DAG for the object (when it is defined), the SID of this group is written into the OWNER field of the SD.
- Otherwise, if the requester's security context contains the TokenOwner field, then the SID contained in this field is written into the OWNER field of the SD.
- Otherwise, the requester's user SID is written into the OWNER field of the SD.

If the DC functional level is DS_BEHAVIOR_WIN2008 or higher, and the DAG was used as the default OWNER field value, then the same SID is written into the GROUP field. In all other cases, the GROUP field is not modified before the SD value is passed to the CreateSecurityDescriptor algorithm as specified in section 6.1.3.

6.1.3.7 Default Administrators Group

The "default administrators group" (DAG), which is used for OWNER/GROUP defaulting and also in OWNER write access checks, is computed based on two inputs: the contents of the requester's token and the location of the object whose SD is being written. The following rules are applied (in order):

1. When the object belongs to a domain NC:
    1. If the user is a member of Domain Admins for this domain, then Domain Admins is designated as the DAG.
    2. If the user is a member of Enterprise Admins for the forest, then Enterprise Admins is designated as the DAG.
    3. Otherwise, the DAG is undefined.
2. When the object belongs to the config NC:
    1. If the user is a member of Enterprise Admins, then Enterprise Admins is designated as the DAG.
    2. If the user is a member of Domain Admins (for the domain that the current DC belongs to), then this Domain Admins group is designated as the DAG.
    3. Otherwise, the DAG is undefined.
3. When the object belongs to the schema NC:
    1. If the user is a member of Schema Admins, then Schema Admins is designated as the DAG.
    2. If the user is a member of Enterprise Admins, then Enterprise Admins is designated as the DAG.
    3. If the user is a member of Domain Admins (for the domain that the current DC belongs to), then this Domain Admins group is designated as the DAG.
    4. Otherwise, the DAG is undefined.
4. When the object belongs to an application NC:
    1. If the user is a member of Domain Admins for the domain that is designated as sdReferenceDomain for this application NC, then this Domain Admins group is designated as the DAG.

2. If the user is a member of Enterprise Admins, then Enterprise Admins is designated as the DAG.

3. Otherwise, the DAG is undefined.

6.1.4 Special Attributes

Glossary Terms: FSMO Role, PDC FSMO Role Owner

LDAP Attributes: nTMixedDomain, msDS-Behavior-Version

LDAP Classes: nTDSDSA, crossRef

Constants: crossRefContainer

6.1.4.1 ntMixedDomain

The attribute nTMixedDomain is present on each domain NC root object. The value of this attribute MUST be 0 or 1. The value 1 indicates a domain that is in mixed mode and that supports replication to Windows NT operating system backup domain controllers ([MS-NRPC]). The value 0 indicates a domain that does not support such replication.

If the value of nTMixedDomain is 0, it cannot be changed.

The attribute nTMixedDomain on a crossRef object is read-only and equals the attribute nTMixedDomain on the corresponding domain NC root object.

If there are Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, or Windows Server 2012 R2 operating system DCs in the domain, nTMixedDomain MUST be 0. This implies that Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 DCs cannot be used in a domain that is in mixed mode.

6.1.4.2 msDS-Behavior-Version: DC Functional Level

The msDS-Behavior-Version attribute is written on the nTDSDSA object representing a DC. The value is the highest domain or forest functional level that the DC is capable of supporting. A DC supports any domain or forest functional level less than or equal to its msDS-Behavior-Version.

The value of the msDS-Behavior-Version attribute on an nTDSDSA object changes during an operating system upgrade of that DC. The value of the msDS-Behavior-Version attribute never decreases.

The absence of the msDS-Behavior-Version attribute on an nTDSDSA object is equivalent to the msDS-Behavior-Version attribute on that object having the value zero.

The following values are defined.

Identifier	Value
DS_BEHAVIOR_WIN2000	0
DS_BEHAVIOR_WIN2003	2
DS_BEHAVIOR_WIN2008	3
DS_BEHAVIOR_WIN2008R2	4

Identifier	Value
DS_BEHAVIOR_WIN2012	5
DS_BEHAVIOR_WIN2012R2	6

6.1.4.3 msDS-Behavior-Version: Domain NC Functional Level

The msDS-Behavior-Version for domains is written on both the domain NC root object and the crossRef representing the domain. The attribute on the crossRef is read-only and is kept in sync with the attribute on the domain NC root object. Only the PDC FSMO role owner accepts originating updates to the attribute on the domain NC root.

Requirements: The functional level of a domain is never larger than any domain DC's functional level that hosts or is instructed to host (see section 6.1.2.3) the domain NC. When the functional level of a domain is DS_BEHAVIOR_WIN2003 or greater, the attribute nTMixedDomain on the domain NC root is 0 (see section 6.1.4.1).

The absence of the msDS-Behavior-Version attribute on a domain NC root object is equivalent to the msDS-Behavior-Version attribute on that object having the value zero.

The value msDS-Behavior-Version defines the lower limit on the version of the server operating system that can run on domain controllers within the domain. Ensuring this lower limit allows advanced features to be enabled throughout the domain.

The following values are defined.

Identifier	Domain controller operating systems that are allowed in the domain	Value
DS_BEHAVIOR_WIN2000	Windows 2000 Server operating system Windows Server 2003 operating system Windows Server 2008 operating system Windows Server 2008 R2 operating system Windows Server 2012 operating system Windows Server 2012 R2 operating system	0
DS_BEHAVIOR_WIN2003_WITH_MIXED_DOMAINS	Windows Server 2003 Windows Server 2008 Windows Server 2008 R2 Windows Server 2012 Windows Server 2012 R2	1
DS_BEHAVIOR_WIN2003	Windows Server 2003 Windows Server 2008 Windows Server 2008 R2 Windows Server 2012 Windows Server 2012 R2	2
DS_BEHAVIOR_WIN2008	Windows Server 2008 Windows Server 2008 R2 Windows Server 2012	3

Identifier	Domain controller operating systems that are allowed in the domain	Value
	Windows Server 2012 R2	
DS_BEHAVIOR_WIN2008R2	Windows Server 2008 R2 Windows Server 2012 Windows Server 2012 R2	4
DS_BEHAVIOR_WIN2012	Windows Server 2012 Windows Server 2012 R2	5
DS_BEHAVIOR_WIN2012R2	Windows Server 2012 R2	6

6.1.4.4 msDS-Behavior-Version: Forest Functional Level

The msDS-Behavior-Version for the forest is written on the crossRefContainer object (see section 6.1.1.2.1). Only the Domain Naming Master role FSMO owner accepts updates to this attribute.

Requirements: The value of msDS-Behavior-Version for the forest is never larger than any functional level of any domain NC in the forest.

The absence of the msDS-Behavior-Version attribute on a crossRefContainer object is equivalent to the msDS-Behavior-Version attribute on that object having the value zero.

The value msDS-Behavior-Version defines the lower limit on the version of the server operating system that can run on domain controllers within the forest. Ensuring this lower limit allows advanced features to be enabled throughout the forest.

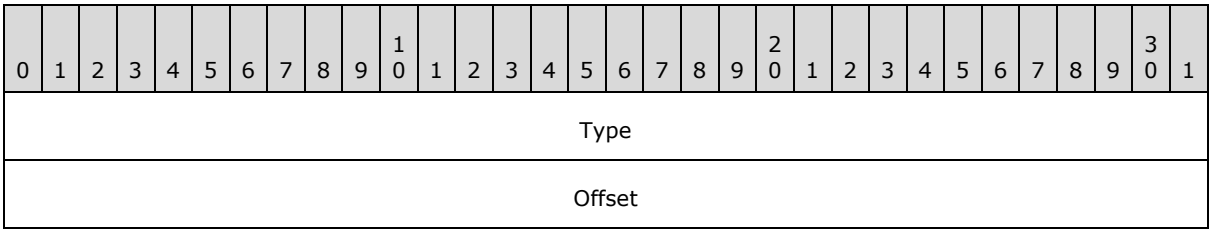
The following values are defined.

Identifier	Domain controller operating systems or products that are allowed in the forest	Value
DS_BEHAVIOR_WIN2000	Windows 2000 Server operating system Windows Server 2003 operating system Windows Server 2008 operating system Windows Server 2008 R2 operating system Windows Server 2012 operating system Windows Server 2012 R2 operating system	0
DS_BEHAVIOR_WIN2003_WITH_MIXED_DOMAINS	Windows Server 2003 Windows Server 2008 Windows Server 2008 R2 Windows Server 2012 Windows Server 2012 R2	1
DS_BEHAVIOR_WIN2003	Windows Server 2003 Active Directory Application Mode (ADAM) Windows Server 2008 Windows Server 2008 R2 Windows Server 2012 Windows Server 2012 R2	2

Identifier	Domain controller operating systems or products that are allowed in the forest	Value
DS_BEHAVIOR_WIN2008	Windows Server 2008 Windows Server 2008 R2 Windows Server 2012 Windows Server 2012 R2	3
DS_BEHAVIOR_WIN2008R2	Windows Server 2008 R2 Windows Server 2012 Windows Server 2012 R2	4
DS_BEHAVIOR_WIN2012	Windows Server 2012 Windows Server 2012 R2	5
DS_BEHAVIOR_WIN2012R2	Windows Server 2012 R2	6

6.1.4.5 Replication Schedule Structures

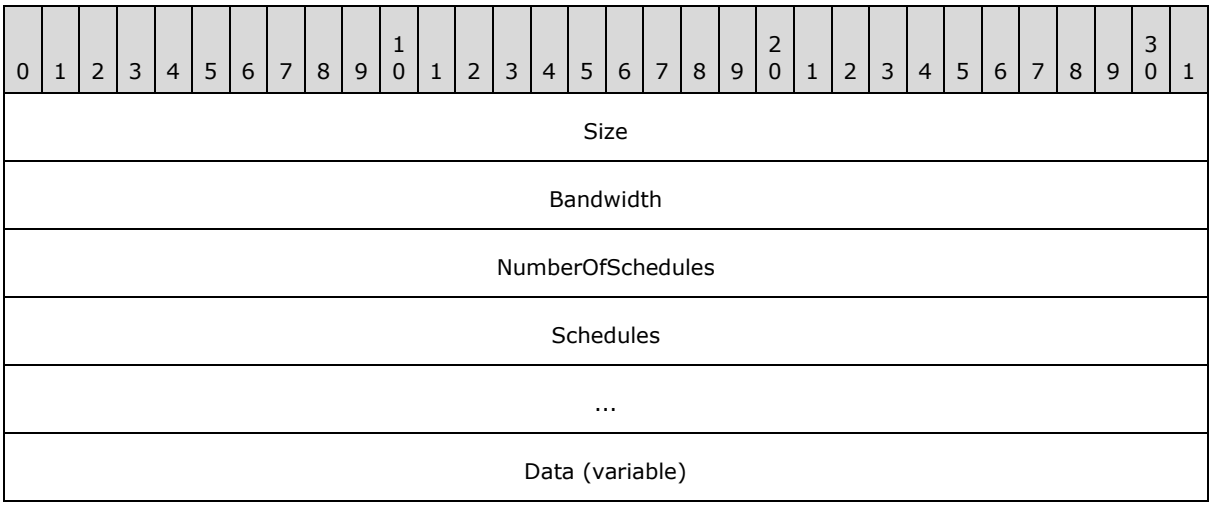
6.1.4.5.1 SCHEDULE_HEADER Structure



Type (4 bytes): This value must be 0.

Offset (4 bytes): An offset, in bytes, into the Data field of the SCHEDULE structure. The offset represents the start of the replication schedule data.

6.1.4.5.2 SCHEDULE Structure



...

Size (4 bytes): Size of the entire replication schedule structure.

Bandwidth (4 bytes): Not used; this field is ignored.

NumberOfSchedules (4 bytes): Number of elements in Schedules. This value is always 1.

Schedules (8 bytes): Array of SCHEDULE_HEADER structures. There is only one SCHEDULE_HEADER element in the array.

Data (variable): This is a sequence of bytes specifying the time slots when replication is permitted between the source and the destination DC. Each schedule header specifies an offset into the data field. The replication schedule data for that schedule is the next 168 bytes. Each byte represents an hour in the week (24 * 7 = 168). The lower 4 bits of each byte represent 15-minute intervals in the hour. The first bit, that is, the fourth least significant bit in the byte, corresponds to the first 15 minutes in the hour, the second bit corresponds to the next 15 minutes, and so on. If one of these bits is set, it indicates that replication is permitted in that 15-minute time interval within that hour.

The offset field of the SCHEDULE_HEADER structure points to the beginning of the Data field, and the Data field is exactly 168 bytes since there is only one schedule.

6.1.4.5.3 REPS_FROM

Specified in [MS-DRSR] section 5.167.

6.1.4.5.4 REPS_TO

Specified in [MS-DRSR] section 5.168.

6.1.4.5.5 MTX_ADDR Structure

Specified in [MS-DRSR] section 5.131.

6.1.4.5.6 REPLTIMES Structure

Specified in [MS-DRSR] section 5.164.

6.1.4.5.7 PAS_DATA Structure

Specified in [MS-DRSR] section 5.148.

6.1.4.6 msDS-AuthenticatedAtDC

This attribute is maintained by the DC on user and computer objects. The attribute contains a list of computer objects, corresponding to the RODCs at which the user or computer has authenticated. This attribute is a forward link attribute whose corresponding back link is the msDS-AuthenticatedToAccountlist attribute (see section 6.1.1.3.2). When a writable DC authenticates a user or computer to an RODC, that writable DC adds the DN of the RODC's computer object to the list in the msDS-AuthenticatedAtDC attribute of the user or computer that was authenticated.

This attribute was first maintained by DCs running Windows Server 2008 operating system.

6.1.5 FSMO Roles

References: SID, RID, RID Allocation, RID Master role in interdomain move, PDC Emulator role, Infrastructure role

Functions: RoleObject, GetRoleScope

Glossary Terms: FSMO Role, NC Replica, DC, SID

Ldap Attributes: fSMORoleOwner

Ldap Classes: nTDSDSA

A FSMO role is defined as a set of objects that may be updated in only one NC replica at any given time. The DC that hosts this NC replica is the owner for that FSMO role.

Each FSMO role is represented by an object in the directory. The function RoleObject (section 3.1.1.5.1.8) specifies the object for a given FSMO role type and NC. This object is an element of the FSMO role and contains the fSMORoleOwner attribute, which references the nTDSDSA object of the DC that owns the role. The function GetRoleScope defined in [MS-DRSR] section 4.1.10.5.16 identifies the set of objects that comprise each FSMO role. These objects must be updated only on the DC that currently owns the FSMO role.

6.1.5.1 Schema Master FSMO Role

The Schema Master FSMO role owner is the DC responsible for performing updates to the directory schema. This DC is the only one that can process updates to the directory schema. Once the schema update is complete, it is replicated from the Schema Master FSMO role owner to all other DCs in the directory. There is only one Schema Master FSMO role per forest.

6.1.5.2 Domain Naming Master FSMO Role

The Domain Naming Master FSMO role owner is the DC responsible for making changes to the forest-wide domain name space of the directory in the Partitions container. This DC is the only one that can add or remove a domain or application NC from the directory. It can also add or remove cross references to domains in external directories. Only the Domain Naming Master FSMO role owner can write to the Partitions container or its children. There is only one Domain Naming Master FSMO role per forest.

6.1.5.3 RID Master FSMO Role

The RID Master FSMO role owner is the single DC responsible for processing RID Pool requests from all DCs within a given domain. It is also responsible for moving an object from one domain to another during an interdomain object move.

When a DC creates a security principal object such as a user or group, it attaches a unique SID to the object. This SID consists of a domain SID (the same for all SIDs created in a domain) and a relative ID (RID) that is unique for each security principal SID created in a domain.

Each DC in a domain is allocated a pool of RIDs that it is allowed to assign to the security principals it creates. When a DC's allocated RID pool falls below a threshold, that DC issues a request for additional RIDs to the domain's RID Master FSMO role owner (see [MS-DRSR] section 4.1.10.4.3, PerformExtendedOpRequestMsg with ulExtendedOp = EXOP_FSMO_RID_REQ_ROLE). The RID Master FSMO role owner responds to the request by retrieving RIDs from the domain's unallocated RID pool and assigns them to the pool of the requesting DC (see [MS-DRSR] section 4.1.10.5.12,

ProcessFsmoRoleRequest with ulExtendedOp = EXOP_FSMO_RID_REQ_ROLE). There is one RID Master FSMO role per domain in a directory.

See section 3.1.1.5 for more information about the RID Master's role in interdomain object move operations.

6.1.5.4 PDC Emulator FSMO Role

The PDC Emulator FSMO role owner performs the following functions:

- Password changes performed by other DCs in the domain are replicated preferentially to the PDC emulator.
- If a logon authentication fails at a given DC in a domain due to a bad password, the DC will forward the authentication request to the PDC emulator to validate the request against the most current password. If the PDC reports an invalid password to the DC, the DC will send back a bad password failure message to the user.
- Account lockout is processed on the PDC emulator.
- The PDC emulator FSMO also fulfills the role of the PDC in the **NetLogon** Remote Protocol methods described in [MS-NRPC] section 3. Therefore, the PDC emulator FSMO MUST support and perform all PDC specific functionality specified in that section. Every DC, other than the PDC emulator FSMO, MUST NOT perform this functionality.

There is one PDC Emulator FSMO role per domain in a directory. See 3.1.1.7 for more information about the PDC Emulator FSMO role.

6.1.5.5 Infrastructure FSMO Role

When an object in one domain is referenced by another object in another domain, it represents the reference as a dsname. There is one Infrastructure FSMO role per domain and application NC in a directory.

If all the domain controllers in a domain also host the GC, then all the domain controllers have the current data, and it is not important which domain controller owns the Infrastructure Master (IM) role. See section 3.1.1.5 for more information about the Infrastructure Master.

When the Recycle Bin optional feature is not enabled, the Infrastructure FSMO role owner is the DC responsible for updating a cross-domain object reference in the event that the referenced object is moved, renamed, or deleted. In this case, the Infrastructure Master role should be held by a domain controller that is not a GC server. If the Infrastructure Master runs on a GC server, it will not update object information, because it does not contain any references to objects that it does not hold. This is because a GC server holds a partial replica of every object in the forest.

When the Recycle Bin optional feature is enabled, every DC is responsible for updating its cross-domain object references in the event that the referenced object is moved, renamed, or deleted. In this case, there are no tasks associated with the Infrastructure FSMO role, and it is not important which domain controller owns the Infrastructure Master role.

6.1.6 Trust Objects

6.1.6.1 Overview (Synopsis)

Active Directory domains rarely exist in isolation. Many Active Directory deployments in customer sites consist of two or more domains that represent boundaries between different geographical,

managerial, organizational, or administrative layouts. For example, when company "A" acquires company "B," it quickly becomes necessary for preexisting domains to start trusting each other. Alternately, in some deployments, servers that have a specific role (such as a mail server) may be members of a "resource domain", easing the management burden by combining like roles under one administrative domain.

Enabling communication between disparate domains, especially secure communication involving authentication and authorization, requires that some stateful knowledge be shared between the peer domains in order for them to trust one another. Some of this knowledge is sensitive, forming the cryptographic basis of trust mechanisms used in protocols such as Kerberos and Netlogon RPC. Other state is public knowledge, such as the NetBIOS name of a peer domain, or which security identifiers are owned by the peer domain. Information like this plays a crucial role when performing name lookups, which are essential for authorization, locating user accounts, or simply displaying information in some type of user interface.

Active Directory stores trust information in trusted domain objects (TDOs) and, depending on the kind of trust established, in associated user accounts (interdomain trust accounts) for the trusted domain. This section of the document details the contents of these objects, focusing on analysis of the properties that are specific to TDOs and interdomain trust accounts, and that are essential for proper interdomain functionality.

6.1.6.2 Relationship to Other Protocols

6.1.6.2.1 TDO Replication over DRS

After they are created, TDOs are replicated along with other objects over replication protocols (as specified in [MS-DRSR] and [MS-SRPL]). In this manner, they are no different than any other directory service object.

6.1.6.2.2 TDO Roles in Authentication Protocols over Domain Boundaries

For most network authentication protocols, if a client wishes to securely authenticate to a service residing in a foreign domain, it becomes necessary for the client and service domains to have some form of trust. Most trust systems in use today rely upon some form of key for trust validation.

TDOs play an important part in the storage and distribution of information used for trust validation between domains. Commonly used Windows network authentication mechanisms such as Kerberos ([RFC4120] section 1.1) retrieve information from TDOs that have been established between the client and service domains. Additionally, services using other protocols such as NTLM, Digest, and SSL Certificate Mapping use the Generic Pass-through Mechanism over the Netlogon Remote Protocol [MS-NRPC] to authenticate users from foreign domains. Establishing the Netlogon Secure Channel requires the use of information contained in TDOs. The format and storage locations for this information will be discussed later (section 6.1.6.9.1), including information on the usage for relevant authentication protocols.

6.1.6.2.3 TDO Roles in Authorization over Domain Boundaries

In some configurations, authorization data from a trusted domain, such as a SID ([MS-DTYP] section 2.4.2) or a client name in a Kerberos cross-realm ticket-granting ticket ([RFC4120] section 5.3), must be scrutinized to protect against attempts in the foreign domain to claim identities from within the local domain. For example, if the foreign DC were to become compromised by an attacker, without these protections it would be possible to inject the SID of the local domain administrator into the transferred TGT. This would have the end result of granting the attacker domain administrator rights in the local domain.

To protect against these attacks, TDOs contain name spaces and SID spaces that legitimately belong to the foreign domain. When enabled, authentication protocols will use this information to verify that authorization data that is passed through the protocol is valid for the trust. If a SID or name within the authorization data does not correspond to those claimed within the TDO, the request is rejected. This can cause network logon attempts to fail or may alternately cause Kerberos ticket requests to fail, as discussed in [MS-PAC] section 4.2.3.

6.1.6.3 Prerequisites/Preconditions

TDOs are only used for storing trust information on Windows 2000 operating system, Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system.

6.1.6.4 Versioning and Capability Negotiation

- Building TDOs that represent **cross-forest trusts** requires that both the domain and the forest are running in a domain and forest functional level of DS_BEHAVIOR_WIN2003 or greater.
- An **uplevel trust**, by definition, is one in which both trusting domains are running all Windows 2000 operating system or newer DCs.
- A **downlevel trust** is one in which either of the trusting domains are running Windows NT 4.0 operating system DCs.

6.1.6.5 Vendor-Extensible Fields

It is possible to store provider-specific values in the trustAuthOutgoing and the trustAuthIncoming attributes [MS-ADA3] on a TDO. See the sections on TDO keys (section 6.1.6.9.1) and trustAuthIncoming (section 6.1.6.7.10) for details on the range of extensible values.

6.1.6.6 Transport

TDOs are replicated along with other DS objects, as described in [MS-DRSR] and [MS-SRPL].

6.1.6.7 Essential Attributes of a Trusted Domain Object

TDOs are stored in the System container, with a CN representing the fully qualified domain name (FQDN) (2) of the **trusted** domain. For example, if a.example.com trusts b.example.com, an object would be created in the System container with a CN of b.example.com. The System container can be found by using the function GetWellknownObject(NC, default NC, GUID_SYSTEM_CONTAINER_W). For more information, see section 3.1.1.1.

The contents of TDOs are described by the trustedDomain schema object [MS-ADSC]. The following table details those attributes that are essential to a well-functioning interdomain trust, with links to specific sections detailing their relevance and format when these attributes are present.

Attribute name	Reference
flatName	MS-ADA1
isCriticalSystemObject	MS-ADA1
msDS-SupportedEncryptionTypes	MS-ADA2, MS-ADTS section 6.1.6.9.1

Attribute name	Reference
msDS-TrustForestTrustInfo	MS-ADA2, MS-ADTS section 6.1.6.9.3
nTSecurityDescriptor	MS-ADA3
objectCategory	MS-ADA3
objectClass	MS-ADA3
securityIdentifier	MS-ADA3
trustAttributes	MS-ADA3
trustAuthIncoming	MS-ADA3, MS-ADTS section 6.1.6.9.1
trustAuthOutgoing	MS-ADA3, MS-ADTS section 6.1.6.9.1
trustDirection	MS-ADA3
trustPartner	MS-ADA3
trustPosixOffset	MS-ADA3, MS-ADTS section 6.1.6.9.4
trustType	MS-ADA3

6.1.6.7.1 flatName

The flatName attribute contains the NetBIOS name (as specified in [RFC1088]) of the trusted domain in String(Unicode) syntax.

This attribute is unique on all TDOs within the domain. The system rejects attempts to create a duplicate value.

6.1.6.7.2 isCriticalSystemObject

A mandatory Boolean attribute. Always set to true for TDOs, which indicates that it must be replicated when a new replica is installed.

6.1.6.7.3 msDs-supportedEncryptionTypes

Implemented on Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system only.

Contains bitmapped values that define the encryption types supported by this trust relationship. One or more of the following flags can be set. Unused flags should be set to 0 when writing the attribute and should be ignored when reading the attribute. The flags are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A256	A128	RC4	MD5	CRC

CRC (KERB_ENCTYPE_DES_CBC_CRC, 0x00000001): Supports CRC32 as described in [RFC3961] page 31.

MD5 (KERB_ENCTYPE_DES_CBC_MD5, 0x00000002): Supports RSA-MD5 as described in [RFC3961] page 31.

RC4 (KERB_ENCTYPE_RC4_HMAC_MD5, 0x00000004): Supports RC4-HMAC-MD5 as described in [RFC4757].

A128 (KERB_ENCTYPE_AES128_CTS_HMAC_SHA1_96, 0x00000008): Supports HMAC-SHA1-96-AES128 as described in [RFC3961] page 31.

A256 (KERB_ENCTYPE_AES256_CTS_HMAC_SHA1_96, 0x00000010): Supports HMAC-SHA1-96-AES256 as described in [RFC3961] page 31.

6.1.6.7.4 msDS-TrustForestTrustInfo

Implemented on Windows Server 2003 operating system, Windows Server 2003 R2 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system.

The contents of this attribute are fully specified in section 6.1.6.9.3.

6.1.6.7.5 nTSecurityDescriptor

A mandatory object attribute that contains the security descriptor that is tied to the Active Directory object. The security descriptor mandates access controls to the object. TDOs are sensitive objects and have tight access controls placed upon them. Stored as the type String(NT-Sec-Desc) in SDDL ([MS-DTYP] section 2.5.1), the default security descriptor for TDOs is as follows.

```
Platforms    Default Security Descriptor in SDDL Format
---------    ------------------------------------------
W2000        D:(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;DA)(A;;RPWPCRCCDCLCLOR
             CWOWDSDDTSW;;;SY)(A;;RPLCLORC;;;AU)

W2003        D:(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;DA)(A;;RPWPCRCCDCLCLOR
W2003R2      CWOWDSDDTSW;;;SY)(A;;RPLCLORC;;;AU)(OA;;WP;736e4812-af31-
W2008        11d2-b7df-00805f48caeb;bf967ab8-0de6-11d0-a285-00aa003049
W2008R2      e2;CO)(A;;SD;;;CO)
```

6.1.6.7.6 objectCategory

A mandatory attribute representing the schema definition for TDOs. The value is a reference to the classSchema object for the trustedDomain class.

6.1.6.7.7 objectClass

A mandatory multivalued attribute representing the classes that the target object is derived from. For a TDO, this value contains [top, leaf, trustedDomain].

6.1.6.7.8 securityIdentifier

The securityIdentifier attribute contains a String(Octet) representation of the SID belonging to the trusted domain. This value contains the domain relative SID ([MS-DTYP] section 2.4.2) of identities issued by the trusted domain. For example, for "example.com", a trusted domain, the value may be S-1-5-2223345-6677. The domain administrator for example.com would have a SID of S-1-5-2223345-6677-512.

This attribute is unique on all TDOs within the domain. The system rejects attempts to create a duplicate value.

6.1.6.7.9 trustAttributes

The trustAttributes attribute contains the value of a trust relationship. This value corresponds to the **TrustAttributes** field detailed in the LSAPR_TRUSTED_DOMAIN_INFORMATION_EX structure ([MS-LSAD] section 2.2.7.9). The flags in the following diagram are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
R	R	R	R	R	R	R	R	O	O	R	R	R	R	R	R	R	R	R	R	R	R	R	R	TARC	TATE	TAWF	TACO	TAFT	TAQD	TAUO	TANT

Name and value	Description and restrictions/special notes
TANT (TRUST_ATTRIBUTE_NON_TRANSITIVE) 0x00000001	If this bit is set, then the trust cannot be used transitively. For example, if domain A trusts domain B, which in turn trusts domain C, and the A<-->B trust has this attribute set, then a client in domain A cannot authenticate to a server in domain C over the A<-->B<-->C trust linkage.
TAUO (TRUST_ATTRIBUTE_UPLEVEL_ONLY) 0x00000002	If this bit is set in the attribute, then only Windows 2000 operating system and newer clients may use the trust link. Netlogon does not consume **trust objects** that have this flag set.
TAQD (TRUST_ATTRIBUTE_QUARANTINED_DOMAIN) 0x00000004	If this bit is set, the trusted domain is quarantined and is subject to the rules of SID Filtering as described in [MS-PAC] section 4.1.2.2.
TAFT (TRUST_ATTRIBUTE_FOREST_TRANSITIVE)	If this bit is set, the trust link is a cross-forest trust [MS-KILE] between the root domains of two forests, both of which are running in a forest functional level of

Name and value	Description and restrictions/special notes
0x00000008	DS_BEHAVIOR_WIN2003 or greater. Only evaluated on Windows Server 2003 operating system, Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system. Can only be set if forest and trusted forest are running in a forest functional level of DS_BEHAVIOR_WIN2003 or greater.
TACO (TRUST_ATTRIBUTE_CROSS_ORGANIZATION) 0x00000010	If this bit is set, then the trust is to a domain or forest that is not part of the **organization**. The behavior controlled by this bit is explained in [MS-KILE] section 3.3.5.7.5 and [MS-APDS] section 3.1.5. Only evaluated on Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2. Can only be set if forest and trusted forest are running in a forest functional level of DS_BEHAVIOR_WIN2003 or greater.
TAWF (TRUST_ATTRIBUTE_WITHIN_FOREST) 0x00000020	If this bit is set, then the trusted domain is within the same forest. Only evaluated on Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2.
TATE (TRUST_ATTRIBUTE_TREAT_AS_EXTERNAL) 0x00000040	If this bit is set, then a cross-forest trust to a domain is to be treated as an external trust for the purposes of SID Filtering. Cross-forest trusts are more stringently filtered than external trusts. This attribute relaxes those cross-forest trusts to be equivalent to external trusts. For more information on how each trust type is filtered, see [MS-PAC] section 4.1.2.2. Only evaluated on Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2. Only evaluated if SID Filtering is used. Only evaluated on cross-forest trusts having TRUST_ATTRIBUTE_FOREST_TRANSITIVE. Can only be set if forest and trusted forest are running in a forest functional level of DS_BEHAVIOR_WIN2003 or greater.
TARC (TRUST_ATTRIBUTE_USES_RC4_ENCRYPTION) 0x00000080	This bit is set on trusts with the trustType set to TRUST_TYPE_MIT, which are capable of using RC4 keys. Historically, MIT Kerberos distributions supported only DES and 3DES keys ([RFC4120], [RFC3961]). MIT 1.4.1 adopted the RC4HMAC encryption type common to Windows 2000 [MS-KILE], so trusted domains deploying later versions of the MIT distribution required this bit. For more information, see "Keys and Trusts", section 6.1.6.9.1. Only evaluated on TRUST_TYPE_MIT

Name and value	Description and restrictions/special notes
R 0x00000100 - 0x00200000 0x01000000 - 0x80000000	Reserved
O 0x00400000 - 0x00800000	Previously used trust bits, and are obsolete.

6.1.6.7.10 trustAuthIncoming

This is a String(Octet) attribute. This value is used to compute keys used in **inbound trust** validation. For more information on the contents of this attribute, see "Keys and Trusts", section 6.1.6.9.1.

This is a Secret Attribute ([MS-DRSR] section 4.1.10.3.12, IsSecretAttribute), and is not readable outside of the context of the LSA on a DC.

6.1.6.7.11 trustAuthOutgoing

This is a String(Octet) attribute. This value is used to compute keys used in **outbound trust** validation. For more information on the contents of this attribute, see "Keys and Trusts", section 6.1.6.9.1.

This is a Secret Attribute ([MS-DRSR] section 4.1.10.3.12, IsSecretAttribute), and is not readable outside of the context of the LSA on a DC.

6.1.6.7.12 trustDirection

The trustDirection attribute dictates in which direction the trust flows. It is stored as an integer value. There are four valid values, corresponding to the **TrustDirection** field in the LSAPR_TRUSTED_DOMAIN_INFORMATION_EX structure ([MS-LSAD] section 2.2.7.9). The flags in the following diagram are presented in big-endian byte order.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	TDO	TDI

TRUST_DIRECTION_DISABLED, 0x00000000: Absence of any flags. The trust relationship exists but has been disabled.

TDI (TRUST_DIRECTION_INBOUND, 0x00000001): The trusted domain trusts the primary domain to perform operations such as name lookups and authentication. If this flag is set, then the trustAuthIncoming attribute is present on this object.

TDO (TRUST_DIRECTION_OUTBOUND, 0x00000002): The primary domain trusts the trusted domain to perform operations such as name lookups and authentication. If this flag is set, then the trustAuthOutgoing attribute is present on this object.

TRUST_DIRECTION_BIDIRECTIONAL, 0x00000003: OR'ing of the preceding flags and behaviors representing that both domains trust one another for operations such as name lookups and authentication.

6.1.6.7.13 trustPartner

This String(Unicode) attribute contains the FQDN (2) of the trusted domain. This is a mandatory attribute.

As with the securityIdentifier attribute, this attribute is unique on all TDOs within the domain. The system rejects attempts to create a duplicate value.

6.1.6.7.14 trustPosixOffset

This integer value contains the Portable Operating System Interface (POSIX) offset for the trusted domain. This value is added to the RID of a SID to give the POSIX user ID or group ID (as specified in [IEEE1003.1] sections 3.188 and 3.425) for that user in the trusted domain. The calculation of this value is documented in section 6.1.6.9.4.

6.1.6.7.15 trustType

The trustType attribute is an integer value that dictates what type of trust has been designated for the trusted domain. Following are the valid values, corresponding to the **TrustType** field in LSAPR_TRUSTED_DOMAIN_INFORMATION_EX, as specified in [MS-LSAD] section 2.2.7.9. The trustType contains one of the following values:

TTD (TRUST_TYPE_DOWNLEVEL, 0x00000001): The trusted domain is a Windows domain not running Active Directory.

TTU (TRUST_TYPE_UPLEVEL, 0x00000002): The trusted domain is a Windows domain running Active Directory.

TTM (TRUST_TYPE_MIT, 0x00000003): The trusted domain is running a non-Windows, RFC4120-compliant Kerberos distribution. This type of trust is distinguished in that (1) a SID is not required for the TDO, and (2) the default key types include the DES-CBC and DES-CRC encryption types (see [RFC4120] section 8.1).

TTDCE (TRUST_TYPE_DCE, 0x00000004): Historical reference; this value is not used in Windows.

6.1.6.8 Essential Attributes of Interdomain Trust Accounts

TDOs contain all the information regarding trusts. Trusts that have the trustDirection attribute equal to TRUST_DIRECTION_INBOUND or TRUST_DIRECTION_BIDIRECTIONAL, however, also have associated user accounts called interdomain trust accounts within the default container for users defined in section 6.1.1.4.6. The TDO O1 and the interdomain trust account object O2 for the same trust are associated through the partner domain's NetBIOS name, used to form the following values: the flatName attribute of O1 and the sAMAccountName attribute of O2. Given the partner domain's NetBIOS <NetBIOS Name>, O1!flatName=<NetBIOS Name> and O2!samAccountName=<NetBIOS Name>$.

The following table lists the attributes that MUST be set in an interdomain trust account.

Attribute name	Reference
cn (RDN)	[MS-ADA1]
objectClass	[MS-ADA3]
sAMAccountName	[MS-ADA3]
sAMAccountType	[MS-ADA3]
userAccountControl	[MS-ADA3]

6.1.6.8.1 cn (RDN)

The RDN of an interdomain trust account, the cn attribute, contains the NetBIOS name of the trusted domain account appended with the character '$', in String(Unicode) syntax.

6.1.6.8.2 objectClass

An attribute that represents the classes that the target object is derived from. For a user account, this value contains the sequence [top, person, organizationalPerson, user].

6.1.6.8.3 sAMAccountName

The sAMAccountName attribute contains the NetBIOS name of the trusted domain account appended with the character '$', in String(Unicode) syntax.

6.1.6.8.4 sAMAccountType

In a domain trust account, the sAMAccountType attribute MUST have the value SAM_TRUST_ACCOUNT (0x30000002), in the Enumeration syntax.

6.1.6.8.5 userAccountControl

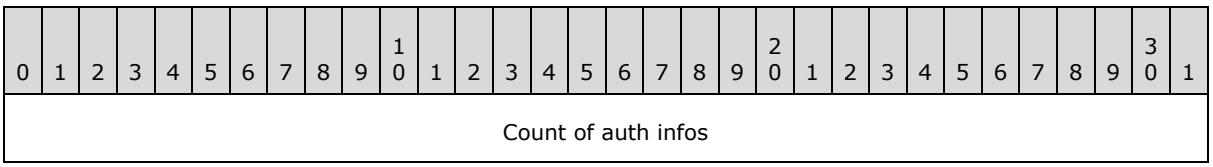
In a domain trust account, the userAccountControl attribute MUST have the flag ADS_UF_INTERDOMAIN_TRUST_ACCOUNT (0x00000800) set.

6.1.6.9 Details

6.1.6.9.1 trustAuthInfo Attributes

Domain peers share a password in order to validate protocol messages flowing between the trusted domains. The password is only good in one direction of the trust. Each direction is stored in its own attribute: the trustAuthIncoming and trustAuthOutgoing attributes. These are both Secret Attributes ([MS-DRSR] section 4.1.10.3.12, IsSecretAttribute), and are not readable outside of the context of the LSA on a DC.

Both trustAuthIncoming and trustAuthOutgoing are stored as a String(Octet). The storage of this information in a TDO is described in the following diagram.



Byte offset to AuthenticationInformation
Byte offset to PreviousAuthenticationInformation
AuthenticationInformation (variable)
...
PreviousAuthenticationInformation (variable)
...

Count of auth infos (4 bytes): A **ULONG** count of the pairs of LSAPR_AUTH_INFORMATION structures. Each current Authentication Information structure is accompanied by a previous Authentication Information structure (even if it is marked as invalid), and the count of the pairs of elements is stored in this field.

Byte offset to AuthenticationInformation (4 bytes): The **BYTE** offset from the base of the structure to the array of LSAPR_AUTH_INFORMATION structures representing the current Authentication information.

Byte offset to PreviousAuthenticationInformation (4 bytes): The **BYTE** offset from the base of the structure to the array of LSAPR_AUTH_INFORMATION structures representing the previous authentication information.

AuthenticationInformation (variable): Array of LSAPR_AUTH_INFORMATION [1...Count].

Following the byte offset to PreviousAuthenticationInformation is an array of LSAPR_AUTH_INFORMATION structures representing the current authentication information.

PreviousAuthenticationInformation (variable): Array of LSAPR_AUTH_INFORMATION [1...Count].

Following the current authentication information is an array of LSAPR_AUTH_INFORMATION structures representing the previous authentication information. If authentication information has not been previously stored, then the previous Authentication Information structure is an exact copy of the current Authentication Information structure.

6.1.6.9.1.1 LSAPR_AUTH_INFORMATION

The format of the LSAPR_AUTH_INFORMATION structure is as follows.

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
LastUpdateTime																															
...																															
AuthType																															

AuthInfoLength
AuthInfo (variable)
...
Padding (variable)
...

LastUpdateTime (8 bytes): This **LARGE_INTEGER** value represents the last time that the authentication information was set, in **FILETIME** format, as specified in [MS-DTYP] section 2.3.

AuthType (4 bytes): This **ULONG** value dictates the type of **AuthInfo** that is being stored. There are four values that are recognized by Windows.

Possible Values	Meaning
TRUST_AUTH_TYPE_NONE 0	**AuthInfo** byte field is invalid/not relevant.
TRUST_AUTH_TYPE_NT4OWF 1	**AuthInfo** byte field contains an RC4 Key [RFC4757].
TRUST_AUTH_TYPE_CLEAR 2	**AuthInfo** byte field contains a cleartext password, encoded as a Unicode string.
TRUST_AUTH_TYPE_VERSION 3	**AuthInfo** byte field contains a version number, used by Netlogon for versioning interdomain **trust secrets**.

AuthInfoLength (4 bytes): A **ULONG** count of bytes in **AuthInfo**.

AuthInfo (variable): A **BYTE** field containing authentication data. Its size is [1...**AuthInfoLength**].

Padding (variable): Some number of bytes used to align the end of the LSAPR_AUTH_INFORMATION structure to a **ULONG** boundary. This padding is not included in the **AuthInfoLength** and consists of zeros.

6.1.6.9.1.2 Kerberos Usages of trustAuthInfo Attributes

Microsoft's implementation of Kerberos ([RFC4120], [MS-KILE]) uses TDOs to retrieve cross-domain passwords when building cross-realm ticket-granting ticket (TGT). The KDC supports the following AuthTypes:

1. TRUST_AUTH_TYPE_CLEAR

   This flag indicates that the information stored in the attribute is a Unicode plaintext password. If this AuthType is present, Kerberos can then use this password to derive additional key types needed to encrypt and decrypt cross-realm TGTs:

   - DES-CBC ([RFC4120] section 8.1)

- DES-CRC [RFC4120]
- RC4HMAC [RFC4757]

Other derivations of the plaintext password are possible using the string-to-key functionality defined in [RFC3961]. It is important to note that if the trustType is set to TRUST_TYPE_MIT, then RC4HMAC keys will not be derived for the trust unless the corresponding TDO's trustAttribute includes the TRUST_ATTRIBUTE_USES_RC4_ENCRYPTION bit flag.

In Windows Server 2008 operating system, Windows Server 2008 R2 operating system, Windows Server 2012 operating system, and Windows Server 2012 R2 operating system, if KERB_ENCTYPE_RC4_HMAC_MD5 (4) is set in the msDs-supportedEncryptionTypes attribute, then the MIT realm supports RC4.

2. TRUST_AUTH_TYPE_NT4OWF

   This flag indicates that the key is stored as a raw RC4HMAC key [RFC4757]. Because the key was precomputed with this AuthType, it is not possible to derive alternate key types for the TDO.

Kerberos' usage of the TDO keys is somewhat counterintuitive. Consider the following scenario involving two trusting Active Directory domains, where a user in a primary domain wishes to authenticate to a service in the trusted domain using Kerberos. The primary domain issues a referral TGT to the trusted domain containing the service.

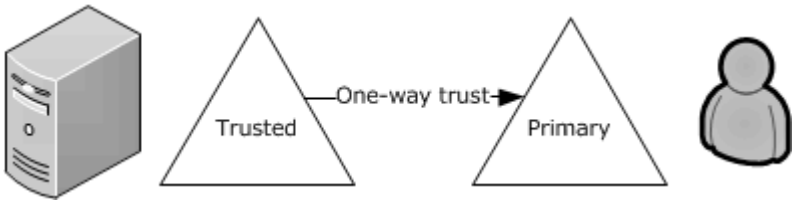


Figure 5: Kerberos protocol usage of keys

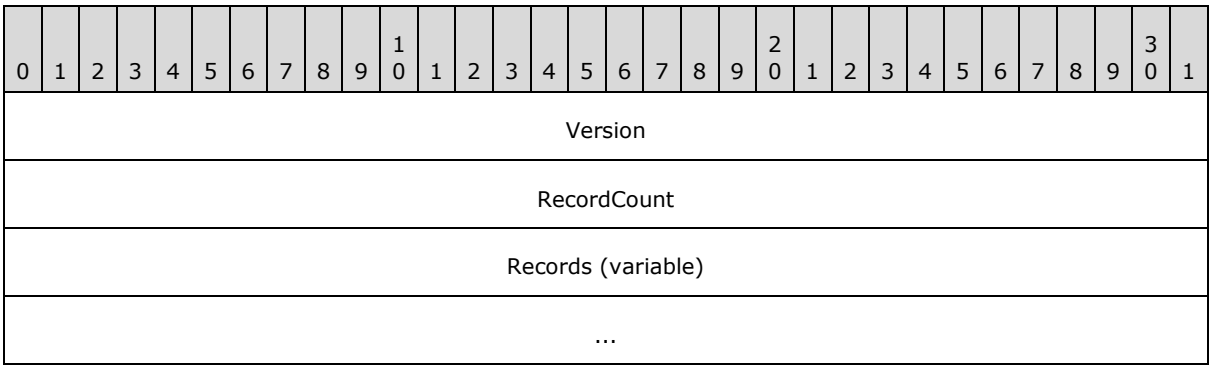
There is a one-way trust in place. The referral TGT issued by the primary domain is encrypted based on the key in trustAuthIncoming, not trustAuthOutgoing. This is non-intuitive but fits the definition of an inbound trust. This direction of trust allows Kerberos to build a TGT for the trusted domain in the primary domain, fulfilling the definition of an inbound trust.

6.1.6.9.2 Netlogon Usages of Trust Objects

Netlogon uses information stored in the TDO and the interdomain trust account to establish the secure channel. The way in which the secure channel is established is described in [MS-NRPC] sections 3.1.1 and 3.1.4.3.

6.1.6.9.3 msDS-TrustForestTrustInfo Attribute

Information about trust relationships with other forests 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 a specific type of data about the forest trust relationship.

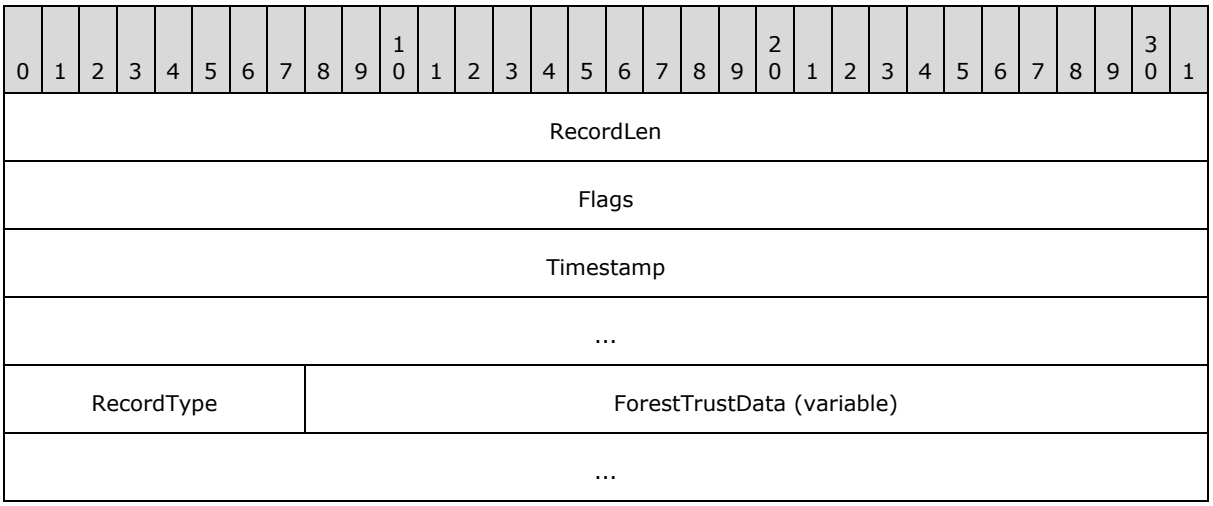
IMPORTANT NOTE: Records are not aligned to 32-bit boundaries. Each record starts at the next byte after the previous record ends.

Each record is represented as described in section 6.1.6.9.3.1.

Note All fields have little-endian byte ordering.

6.1.6.9.3.1 Record

Each Record is represented in the following manner.



RecordLen (4 bytes): Length, in bytes, of the entire record, not including **RecordLen**.

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, represented here in big-endian byte order, 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
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	TDC	TDA	TDN

X: Unused. Must be zero and ignored.

TDN (LSA_TLN_DISABLED_NEW, 0x00000001): Entry is not yet enabled.

TDA (LSA_TLN_DISABLED_ADMIN, 0x00000002): Entry is disabled by administrator.

TDC (LSA_TLN_DISABLED_CONFLICT, 0x00000004): Entry is disabled due to a conflict with another trusted domain.

If RecordType = 2, the **Flags** field, represented here in big-endian byte order, 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
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	NDC	NDA	SDC	SDA

SDA (LSA_SID_DISABLED_ADMIN, 0x00000001): Entry is disabled for SID, NetBIOS, and DNS name–based matches by the administrator.

SDC (LSA_SID_DISABLED_CONFLICT, 0x00000002): Entry is disabled for SID, NetBIOS, and DNS name–based matches due to a SID or DNS name–based conflict with another trusted domain.

NDA (LSA_NB_DISABLED_ADMIN, 0x00000004): Entry is disabled for NetBIOS name–based matches by the administrator.

NDC (LSA_NB_DISABLED_CONFLICT, 0x00000008): Entry is disabled for NetBIOS name–based matches due to a NetBIOS domain name conflict with another trusted domain.

For RecordType = 2, NETBIOS_DISABLED_MASK is defined as a mask on the lower 4 bits of the **Flags** field.

For all record types, LSA_FTRECORD_DISABLED_REASONS is defined as a mask on the lower 16 bits of the **Flags** field. Unused bits covered by the mask are reserved for future use.

Timestamp (8 bytes): 64-bit timestamp value indicating when this entry was created, in system time (see the **FILETIME** structure in [MS-DTYP] section 2.3.3).

RecordType (1 byte): 8-bit value specifying the type of record contained in this specific entry. The structure of the content in the next field depends on this value. The current version of the protocol defines the behavior of the next field **ForestTrustData** if the value of **RecordType** is one of the three values below.

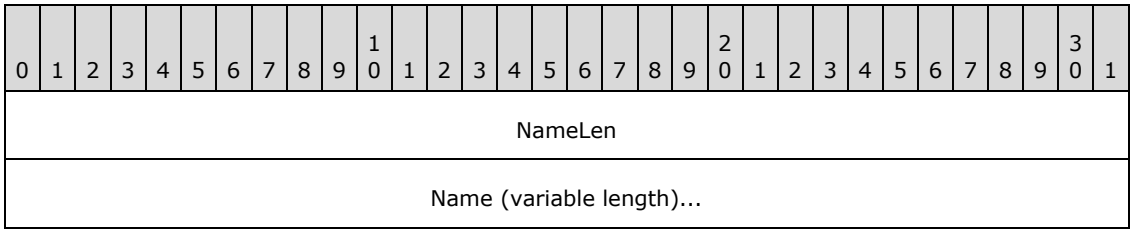
Name	Value
ForestTrustTopLevelName	0
ForestTrustTopLevelNameEx	1
ForestTrustDomainInfo	2

ForestTrustData (variable): Variable-length type-specific record, depending on the RecordType value, containing a 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 follows:

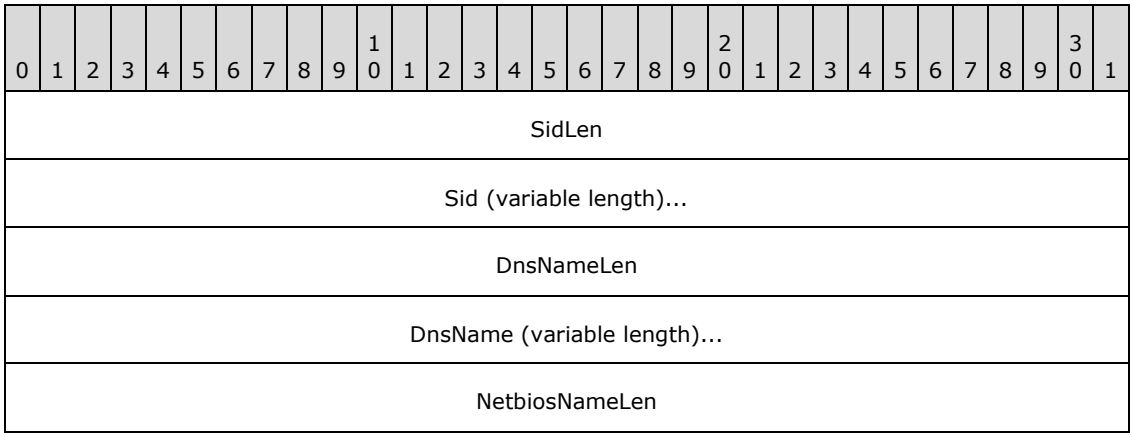
- If RecordType = 0 or RecordType = 1, then the type-specific record is represented in the following manner.



NameLen: Length, in bytes, of the following **Name** field.

Name: The **top level name (TLN)** 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 is necessarily aligned to 32-bit boundaries.



NetbiosName (variable length)...

SidLen: Length, in bytes, of the following **Sid** field.

Sid: The SID of a domain in the trusted forest, specified as a SID structure, which is defined in [MS-DTYP] section 2.4.2.

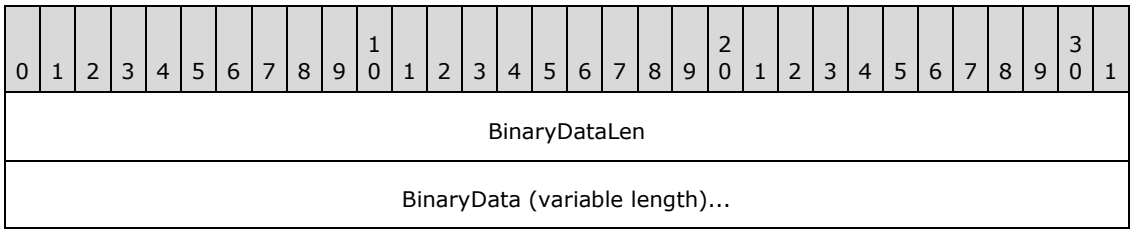
DnsNameLen: Length, in bytes, of the following **DnsName** field.

DnsName: The DNS name of a domain in the trusted forest, in UTF-8 format.

NetbiosNameLen: Length, in bytes, of the following NetbiosName field.

NetbiosName: The NetBIOS name of a domain in the trusted forest, in UTF-8 format.

- If RecordType is not one of the preceding values, the current version of the protocol does not define the behavior for the record data. The type-specific record is represented in the following manner.



BinarydataLen: Length, in bytes, of the following **BinaryData** field.

BinaryData: The record data. If the **BinarydataLen** field has a value other than 0, this field MUST NOT be NULL.

6.1.6.9.3.2 Building Well-Formed msDS-TrustForestTrustInfo Messages

The msDS-TrustForestTrustInfo attribute contains a String(Octet) with the data structures specified in the preceding sections. This attribute contains information about the namespaces that are served by a given trusted forest. For example, if forest a.com contains the domains a.com, b.a.com, and c.a.com, then the msDS-TrustForestTrustInfo for a.com would contain the FQDN and NetBIOS names for each domain, as well as the SID space served by each domain. This section details the rules that well-formed msDS-TrustForestTrustInfo messages must follow.

The msDS-TrustForestTrustInfo attribute is written on the PDC for the trusting and trusted domains. Both the trusted and trusting forest have forest functional level DS_BEHAVIOR_WIN2003 or greater.

Some concepts are necessary to understand the algorithm that is used when validating this attribute.

Namespaces

Namespaces are meant to represent those NetBIOS, FQDN, or SID values that a trusted forest or domain claims.

Top Level Names (TLNs)

TLNs are an important concept when detecting and resolving conflicts in namespaces between different TDOs, and for determining which forest owns a given namespace. A TLN really corresponds to a forest namespace, and in order to be enabled, the TLN must be unique among all TDOs. For example, the TLN for the forest example.com is example.com. Note that it is possible that the forest example.com could have another domain corresponding to an entirely different TLN (for example, mailservers.com), in which case two TLNs would need to be registered for the example.com forest. TLNs for a TDO are stored in records identified by the ForestTrustTopLevelName Record Type.

TLNs that must be excluded from a namespace are identified by the ForestTrustTopLevelNameEx RecordType. Exclusion becomes necessary if the namespaces of two forests collide (for example, the forests corp.mycompany.com and the forest hr.corp.mycompany.com). These exclusions are set administratively to ensure proper functioning of the domain.

Superior/Subordinate Namespaces

When evaluating all forest trusts, TLNs are expressed as FQDNs. Parsing the FQDN allows the concept of superior and subordinate namespaces. For example, for the namespace sample.example.com, the superior namespace (and the TLN) is example.com. Similarly, the sample.example.com namespace is subordinate to the example.com namespace. This allows the routing mechanism to understand that the name sample.example.com is associated with the example.com namespace expressed in the TLN, as it is a subordinate.

Enabled Records vs. Disabled Records

During validation of the Records stored in the msDS-ForestTrustForestInfo, it is possible to have TLN or namespace conflicts. In these circumstances, the conflicting record is disabled. Namespace conflicts are determined using the Record Flags specified in the msDS-ForestTrustInfo data format definitions.

1. ForestTrustTopLevelName RecordType (0)

   If the TDN / TDA / TDC Flags are present, then the name that is present in the TLN and its subordinate namespaces (as well as all domains whose FQDNs are equal to or subordinate to the TLN) is not used for routing names or SIDs.

2. ForestTrustTopLevelNameEx RecordType (1)

   If the TDN / TDA / TDC Flags are present, then the name that is present in the exclusion TLN is not used for exclusion purposes, and conflicts will be unresolved. All domains whose FQDNs are equal to or subordinate to the exclusion TLN are not used for routing names or SIDs.

3. ForestTrustDomainInfo RecordType (2)

   If the NDC or NDA Flags are set, then the NetBIOS name is excluded from routing for the NetBIOS name.

   If the SDA or SDC Flags are set, then the entire domain and all domains whose FQDN names are subordinate to the FQDN name of that domain are excluded from name routing by SID, FQDN, or NetBIOS names. The entire subtree of the forest that is rooted at the affected domain is effectively not computed in the trust **domain name** mappings.

msDS-TrustForestTrustInfo Validation

When the TDO information for a domain is added or changed, or if the DC possessing the PDC FSMO role in the root domain of the forest is freshly started, every TDO with msDS-ForestTrustInfo attributes is validated against all other TDOs. The results of that validation are then rewritten to the

DS and replicated to the other DCs in the domain. DCs that do not own the PDC FSMO role treat the attribute as READONLY and internally consistent.

Validation of the matrix of trusted domains and trusted forest information stored in msDS-ForestTrustInfo includes a mechanism to prevent name collisions. Manipulations of this attribute ensure that each namespace is only assigned to a single TDO. If any of the following rules are violated, the colliding RecordFlag is marked as disabled.

The rules for determining whether namespaces collide for ForestTrustDomainInfo Records are as follows:

1. Each SID corresponding to a domain in a trusted forest is unique among all TDOs and among all of the SIDs listed within the ForestTrustData Records. If not, the Record MUST have the SDC bit in the Record Flags.

2. Each SID for each domain in a trusted forest does not equal any SIDs within the domains of the local forest. If not, the Record MUST have the SDC bit in the Record Flags.

3. Each FQDN corresponding to a domain in a trusted forest is unique among all TDOs and among all of the FQDNs and TLNs listed within the ForestTrustData Records. If not, the Record MUST have the SDC bit in the Record Flags.

4. Each FQDN for each domain in the trusted forest does not correspond to any FQDNs within the domains from the local forest. If not, the Record MUST have the SDC bit in the Record Flags.

5. Each NetBIOS domain name corresponding to a domain in a trusted forest is unique among all TDOs and among all of the NetBIOS domains listed within the Forest Trust Data records. If not, the Record MUST have the NDC bit in the Record Flags. For conflict resolution, the TDO with the alphabetically longest name is disabled.

6. Each NetBIOS name for each domain in the trusted forest does not equal any NetBIOS domain name within the domains of the local forest. If not, the Record MUST have the NDC bit in the Record Flags. Local forest NetBIOS names always take precedence over those of trusted forests.

The rules for determining whether namespaces collide for ForestTrustTopLevelName Records are as follows:

1. Each TLN corresponding to a domain in a trusted forest is unique among all TDOs, and among all of the FQDNs and TLNs listed within the Forest Trust Data records. If not, the conflicting Record has the TDC bit in the Record Flags. For the sake of consistency, since the two TLNs are equal, the first TLN Record that is read is authoritative, and subsequent conflicting Records are disabled.

2. Each TLN for each domain in the trusted forest does not correspond to any FQDNs within the domains from the local forest. If not, the Record has the TDC bit in the Record Flags.

ForestTrustTopLevelNameEx Records, by definition, cannot conflict.

Additionally, additions to msDS-TrustForestTrustInfo pass namespace consistency checks before the attribute is set. Any failures in the consistency checks cause the attempt to modify the msDS-TrustForestTrustInfo to fail. The following rules dictate the requirements that each trusted forest must match:

1. At least one ForestTrustTopLevelName TLN Record is specified for each msDS-TrustForestTrustInfo. It is possible for a forest to have more than one TLN if it contains additional TLNs.

2. All domains listed in the ForestTrustDomainInfo for a TDO are subordinate to the TLNs for that TDO.

3. All domains listed in the ForestTrustDomainInfo are not subordinate or superior to other TLNs unless an exclusion record for that TLN or domain is registered.

If all of the preceding tests pass, then the entry is written in binary format to the msDS-ForestTrustInfo, replicated, and honored by all DCs in the forest.

6.1.6.9.4 Computation of trustPosixOffset

When a new TDO is created, a POSIX offset is computed and assigned to the new TDO's trustPosixOffset attribute. This is done by retrieving the values of the trustPosixOffset attribute of all of the existing outgoing Windows trusts (both TRUST_TYPE_UPLEVEL and TRUST_TYPE_DOWNLEVEL). These values are then sorted. Finally, the range of numbers is searched starting from 1, looking for the next unused valid POSIX offset. The selection process excludes the following values, which are reserved for well-known identities.

Value	Description
0x0800	Reserved for built-in domain
0x4000	Reserved for account domain
0xC000	Reserved for primary domain

The selection process only happens on the DC that possesses the PDC FSMO role. If the trust creation happens on another DC the trustPosixOffset value is set to 0 and is computed using the logic above when the TDO replicates to the PDC FSMO role owner. This keeps TDOs from having matching POSIX offsets, which could result in collisions of UIDS and GIDS.

6.1.6.9.5 Mapping Logon SIDs to POSIX Identifiers

Logon SIDs are assigned by the Windows logon process for each logon session and have the form S-1-5-5-X-Y, where X and Y are treated as a single **LARGE_INTEGER** that is incremented for each logon session. POSIX offsets, as described in section 6.1.6.7.14, are not used during the logon SID to POSIX identifier mapping process. These SIDs are mapped to the constant POSIX ID 0xFFF.

6.1.6.9.6 Timers

6.1.6.9.6.1 Trust Secret Cycling

The keys used to validate trusts periodically expire (typically every 30 days). This is performed by the Netlogon service, which performs this operation when establishing the Secure Channel. Resetting the secure channel secret is discussed in [MS-NRPC] section 3.5.4.4.5.

6.1.6.9.7 Initialization

Despite being replicated normally between peer DCs in a domain, the process of creating or manipulating TDOs is specifically restricted to the LSA Policy APIs, as detailed in [MS-LSAD] section 3.1.1.5. Unlike other objects in the DS, TDOs cannot be created or modified by client machines over the LDAPv3 transport. TDOs can be deleted by client machines over the LDAPv3 transport.

The following trust manipulation remote procedure calls specifically target TDOs and are responsible for creating the special properties detailed in section 6.1.6.7. All are documented in [MS-LSAD] section 3.1.4.

- LsarCreateTrustedDomainEx()
- LsarDeleteTrustedDomain()
- LsarSetTrustedDomainInfoByName()
- LsarSetTrustedDomainInformation()

The preceding APIs enforce the following restrictions.

Each TDO corresponds to exactly one trusted domain. The FQDN, SID, and NetBIOS name set on the TDO all reference the same domain.

The server verifies that the trust is pointing either to a domain within the forest or a domain outside the forest. The check is performed by verifying whether any other domain within the forest has the SID, DNS name, or NetBIOS name matching the information being set. One of two options is legal:

1. SID, DNS name, and NetBIOS name all match the same domain within the forest.
2. No SID, DNS name, or NetBIOS name matches any domain within the forest.

Any other alternative (some information pointing inside the forest and some outside, or information pointing at different domains within the forest) is illegal and causes the server to fail the request.

An attempt by the requester to set the TRUST_ATTRIBUTE_FOREST_TRANSITIVE bit in the trust attributes of the trusted domain object can only succeed if the domain is in a forest functional level of DS_BEHAVIOR_WIN2003 or greater and the server is a domain controller in the root domain of the forest. Failing this, the server rejects the request and does not create the TDO.

An attempt by the requester to set the TRUST_ATTRIBUTE_CROSS_ORGANIZATION bit in the trust attributes of the trusted domain object can only succeed if the domain is in a forest functional level of DS_BEHAVIOR_WIN2003 or greater. Failing this, the server rejects the request and does not create the TDO.

Neither TRUST_ATTRIBUTE_FOREST_TRANSITIVE nor TRUST_ATTRIBUTE_CROSS_ORGANIZATION bits are compatible with the TRUST_ATTRIBUTE_WITHIN_FOREST bit. The server rejects invalid combinations of trust attributes and does not create the TDO.

Uplevel or downlevel trusts that have TRUST_DIRECTION_OUTBOUND as one of the direction bits cannot have a SID of NULL. Attempts to set this combination of parameters cause the server to fail the request.

If the TRUST_ATTRIBUTE_FOREST_TRANSITIVE bit is cleared from a TDO's trustAttributes attribute, all of the forest trust information on that TDO is removed from the TDO's msDS-TrustForestTrustInfo attribute.

6.1.6.10 Security Considerations for Implementers

Mechanisms of trust depend on secure initialization. [MS-LSAD] describes the secure trust creation system that is used by Active Directory. In this system, all creation and manipulation of TDOs takes place over a secure session transport, where the client has been authenticated, and sensitive trust information is not sent in the clear. Keys used for trust secrets are sufficiently strong to disallow brute force attacks on the cryptographic material used in cross-domain protocols.

6.1.7 DynamicObject Requirements

Dynamic objects are objects that are automatically deleted after a period of time. When they are deleted (automatically or manually), they do not transform into any other state, such as a tombstone, deleted-object, or recycled-object. They are distinguished from regular objects by the presence of the dynamicObject auxiliary class among their objectClass values. The intended time of deletion is specified by the msDS-Entry-Time-To-Die attribute.

The following requirements apply to dynamic objects:

- All of the dynamic object's descendants are dynamic objects.
- A dynamic object MUST be automatically deleted when all of the following conditions are true:
  - The current time value is greater than or equal to its msDS-Entry-Time-To-Die attribute value.
  - It has no descendants.
- If a dynamic object has descendent objects and the msDS-Entry-Time-To-Die of the dynamic object is earlier than msDS-Entry-Time-To-Die of its descendant, then the DC MUST update the msDS-Entry-Time-To-Die of the object to be greater than the maximum msDS-Entry-Time-To-Die of its descendants. This update MUST occur before the current time reaches its original msDS-Entry-Time-To-Die value.
- NC replicas do not contain objects with linked attribute values referencing deleted dynamic objects. In other words, when a dynamic object is deleted, any linked attribute values on other objects referencing it are removed.
- If any NC replicas contain other objects with nonlinked attribute values referencing deleted dynamic objects, those attribute values on those objects are retained. In other words, when a dynamic object is deleted, any nonlinked attribute values on other objects referencing it are not removed.
- The value of the entryTTL constructed attribute is specified in section 3.1.1.4.5.12.

6.2 Knowledge Consistency Checker

A server running Active Directory is part of a distributed system that performs replication. The Knowledge Consistency Checker (KCC) is a component that reduces the administrative burden of maintaining a functioning replication topology. Additional background is provided in section 3.1.1.1.13.

6.2.1 References

- DRS options bits: [MS-DRSR] section 5.41.
- instanceType bits: [MS-DRSR] section 5.91.
- repsFrom abstract attribute: [MS-DRSR] section 5.169.
- repsTo abstract attribute: [MS-DRSR] section 5.170.
- replUpToDateVector abstract attribute: [MS-DRSR] section 5.165.
- kCCFailedConnections and kCCFailedLinks variables: [MS-DRSR] sections 5.110 and 5.111.
- IDL_DRSGetNCChanges method: [MS-DRSR] section 4.1.10.

- IDL_DRSReplicaAdd method: [MS-DRSR] section 4.1.19.
- IDL_DRSReplicaDel method: [MS-DRSR] section 4.1.20.
- IDL_DRSReplicaModify method: [MS-DRSR] section 4.1.22.
- IDL_DRSExecuteKCC method: [MS-DRSR] section 4.1.6.
- DWORD, GUID types: [MS-DTYP] sections 2.2 and 2.3.4.
- AmIRODC method: [MS-DRSR] section 5.7.

6.2.2 Overview

The KCC automates management of the NC replica graph for each NC in the forest. In doing so, it maintains the following requirements:

- There exists a path from each writable replica to every other NC replica (writable, read-only full, or read-only partial) of the same NC.
- No path from a writable replica to another writable replica passes through a read-only replica.
- For each domain NC, the path from a writable replica to another writable replica utilizes only the **RPC transport** (never SMTP [MS-SRPL]).
- For each domain NC, the path from a writable replica to a read-only full replica utilizes only the RPC transport (never SMTP [MS-SRPL]).
- Replication latency is short between NC replicas on DCs in the same site, at the expense of additional replication traffic within the site.
- Replication traffic between sites is low, at the expense of additional replication latency between sites.
- A state in which one or more DCs are offline or unreachable (temporarily or indefinitely) does not cause the replication latency across the remaining DCs to grow without bound.
- Edges between DCs in different sites constitute a least cost spanning tree for an administrator-defined cost metric.

The KCC performs this work in a sequence of tasks called a "run". These runs execute periodically and on receipt of an IDL_DRSExecuteKCC request. The first periodic run of the Windows KCC begins 5 minutes after system startup. Subsequent runs execute such that the interval between the end of one run and the beginning of the next run is 15 minutes.

These tasks utilize the following inputs:

- Config NC objects: crossRef, interSiteTransport, nTDSDSA, nTDSConnection, site, nTDSSiteSettings, siteLink, siteLinkBridge
- Abstract attributes of NC replicas: repsFrom, repsTo
- Variables of DCs: kCCFailedConnections, kCCFailedLinks
- Current date/time

And produce or update the following:

- Config NC objects: nTDSConnection
- Abstract attributes of NC replicas: repsFrom, repsTo
- Variables of DCs: kCCFailedConnections, kCCFailedLinks

The KCC individual tasks are detailed in the remainder of this section, and are executed in the sequence in which they appear in this document. In summary, these tasks are:

- Refresh kCCFailedLinks and kCCFailedConnections.
- Create intra-site connections.
- Create inter-site connections.
- Remove unnecessary connections.
- Translate connections.
- Remove unnecessary kCCFailedLinks and kCCFailedConnections.

To simplify the task descriptions, the following concepts are used:

- An NC replica that "is present" on a DC. Given NC replica *r* of NC *n* and a DC with nTDSDSA object *o*, *r* "is present" on the DC if both of the following conditions is true:
  - *o*!hasMasterNCs contains *n* or *o*!msDS-hasFullReplicaNCs contains *n* or *o*!hasPartialReplicaNCs contains *n*.
  - One of the following two conditions is true:
    - *o*!msDS-HasInstantiatedNCs contains no value *v* where the dsname portion of *v* = *n*. (In this case *n* is in the process of being instantiated.)
    - *o*!msDS-HasInstantiatedNCs contains a value *v*, where the dsname part of *v* = *n*, and the binary part of *v* (DWORD in big-endian byte order) is an integer such that the IT_NC_GOING bit is clear. (In this case *n* is instantiated, and is not in the process of being uninstantiated.)
- An NC replica that "should be present" on a DC. Given NC replica *r* of NC *n* and a DC with nTDSDSA object *o*, *r* "should be present" on the DC if *r* is one of the following:
  - A writable replica of the config NC, the schema NC, or the DC's default NC on a writable DC.
  - A full read-only replica of the config NC, the schema NC, or the DC's default NC on an RODC.
  - A writable replica of an application NC for which there exists a crossRef object *cr* such that *cr*!nCName = *n* and *cr*!msDS-NC-Replica-Locations contains a reference to *o*.
  - A full read-only replica of an application NC for which there exists a crossRef object *cr* such that *cr*!nCName = *n* and *cr.ms-DS-NC-RO-Replica-Locations* contains a reference to *o*.
  - If the DC is a GC server (that is, if bit NTDSDSA_OPT_IS_GC is set in *o*!options), a partial replica of a domain NC *n* such that *n* ≠ the DC's default NC, and there exists a crossRef object *cr* such that *cr*!nCName = *n*.
- An nTDSConnection object "implies" a tuple in the repsFrom abstract attribute of an NC replica (and a corresponding edge in an NC replica graph). An nTDSConnection object *cn* implies a tuple

in *r*!repsFrom for NC replica *r* of NC *n* on the DC with nTDSDSA object *t*, if each of the following is true:

- *cn* is a child of *t*.
- *cn*!fromServer references an nTDSDSA object *s*.
- An NC replica of *n* "is present" on *s*.
- *r* "should be present" on *t*.
- The NC replica on *s* is a full replica or *r* is a partial replica.
- *n* is not a domain NC, or *r* is a partial replica, or *cn*!transportType has no value, or *cn*!transportType has an RDN of CN=IP.

6.2.2.1 Refresh kCCFailedLinks and kCCFailedConnections

This task refreshes and reconciles the contents of the *kCCFailedLinks* and *kCCFailedConnections* variables.

The KCC updates *kCCFailedLinks* by inspecting the repsFrom abstract attribute associated with each NC replica on the local DC. It first resets the *FailureCount* of each tuple in *kCCFailedLinks* to 0. Then, for each NC replica *r*, for each tuple *rf* in *r*!repsFrom, if *rf.consecutiveFailures* > 0:

- If a tuple *f* exists in *kCCFailedLinks* such that *f.UUIDDsa* = *rf.uuidDsa* and *f.FailureCount* ≠ 0:
  - Set *f.FailureCount* to MAX(*f.FailureCount*, *rf.consecutiveFailures*)
  - Set *f.TimeFirstFailure* to MIN(*f.TimeFirstFailure*, *rf.timeLastSuccess*)
  - Set *f.LastResult* to *rf.resultLastAttempt*
- If a tuple *f* exists in *kCCFailedLinks* such that *f.UUIDDsa* = *rf.uuidDsa* and *f.FailureCount* = 0:
  - Set *f.FailureCount* to *rf.consecutiveFailures*
  - Set *f.TimeFirstFailure* to *rf.timeLastSuccess*
  - Set *f.LastResult* to *rf.resultLastAttempt*
- If no tuple *f* exists in *kCCFailedLinks* such that *f.UUIDDsa* = *rf.uuidDsa*, add tuple *g* to *kCCFailedLinks* such that:
  - *g.UUIDDsa* = *rf.uuidDsa*
  - *g.FailureCount* = *rf.consecutiveFailures*
  - *g.TimeFirstFailure* = *rf.timeLastSuccess*
  - *g.LastResult* = *rf.resultLastAttempt*

For each tuple *k* in *kCCFailedConnections*, the KCC attempts to connect to that DC by calling the IDL_DRSBind method. If the method call is successful, the KCC removes *k* from *kCCFailedConnections*. Otherwise, it increments *k.FailureCount* by 1.

6.2.2.2 Intrasite Connection Creation

This task computes an NC replica graph for each NC replica that "should be present" on the local DC. Then for each edge of the graph directed to an NC replica on the local DC, the KCC reconciles its portion of the NC replica graph by creating an nTDSConnection object to "imply" that edge if one does not already exist.

If the site of the local DC has a site settings object *o* and the NTDSSETTINGS_OPT_IS_AUTO_TOPOLOGY_DISABLED bit is set in *o*!options, the KCC skips this task.

For each NC *x* for which an NC replica "should be present" on the local DC, the KCC constructs an NC replica graph as follows:

- Let *R* be a sequence containing each writable replica *f* of *x* such that *f* "is present" on a DC *s* satisfying the following criteria:
  - *s* is a writable DC other than the local DC.
  - *s* is in the same site as the local DC.
  - If *x* is a read-only full replica and *x* is a domain NC, then the DC's functional level is at least DS_BEHAVIOR_WIN2008.
  - Bit NTDSSETTINGS_OPT_IS_TOPL_DETECT_STALE_DISABLED is set in the options attribute of the site settings object for the local DC's site, or no tuple *z* exists in the *kCCFailedLinks* or *kCCFailedConnections* variables such that *z.UUIDDsa* is the objectGUID of the nTDSDSA object for *s*, *z.FailureCount* > 0, and the current time - *z.TimeFirstFailure* > 2 hours.
- If a partial (not full) replica of *x* "should be present" on the local DC, append to *R* each partial replica *p* of *x* such that *p* "is present" on a DC *s* satisfying the same criteria defined above for full replica DCs.
- Append to *R* the NC replica that "should be present" on the local DC.
- Sort *R* in order of the value of the objectGUID attribute of the corresponding DC's nTDSDSA object. Let r_i be the *i*'th NC replica in *R*, where $0 \le i < |R|$.
- Add a node for each r_i to the NC replica graph.
- Add an edge from r_i to r_{i+1} for each $0 \le i < |R|-1$ if r_i is a full replica or r_{i+1} is a partial replica.
- Add an edge from r_{i+1} to r_i for each $0 \le i < |R|-1$ if r_{i+1} is a full replica or r_i is a partial replica.
- Add an edge from $r_{|R|-1}$ to r_0 if $r_{|R|-1}$ is a full replica or r_0 is a partial replica.
- Add an edge from r_0 to $r_{|R|-1}$ if r_0 is a full replica or $r_{|R|-1}$ is a partial replica.

The KCC can create additional edges, but does not create more than 50 edges directed to a single DC. To optimize replication latency in sites with many NC replicas, the Windows KCC determines that each r_i should have n+2 total edges directed to it such that n is the smallest non-negative integer satisfying $|R| \le 2n^2 + 6n + 7$. For each existing nTDSConnection object implying an edge from r_j of R to r_i such that $j \ne i$, an edge from r_j to r_i is not already in the graph, and the total edges directed to r_i is less than n+2, the KCC adds that edge to the graph. The KCC then adds new edges directed to r_i to bring the total edges to n+2, where the NC replica r_k of R from which the edge is directed is chosen at random such that $k \ne i$ and an edge from r_k to r_i is not already in the graph.

For each edge directed to the NC replica that "should be present" on the local DC, the KCC determines whether an object *c* exists such that:

- *c* is a child of the local DC's nTDSDSA object.
- *c*!objectCategory = nTDSConnection
- Given the NC replica r_i from which the edge is directed, *c*!fromServer is the dsname of the nTDSDSA object of the DC on which r_i "is present".
- *c*!options does not contain NTDSCONN_OPT_RODC_TOPOLOGY

If no such object *c* exists, the KCC adds an object *c* to the local DC's NC replica of the config NC such that it satisfies the above criteria and has the following additional attributes:

- *c*!objectClass contains nTDSConnection
- *c*!enabledConnection = true
- *c*!options = NTDSCONN_OPT_IS_GENERATED
- *c*!systemFlags = FLAG_CONFIG_ALLOW_RENAME + FLAG_CONFIG_ALLOW_MOVE
- *c*!schedule = *z* : SCHEDULE, such that:
  - *z.Size* = 188
  - *z.Bandwidth* = 0
  - *z.NumberOfSchedules* = 1
  - *z.Schedules[0].Type* = 0
  - *z.Schedules[0].Offset* = 20
  - Byte offset 20 from *z* begins a stream of 168 bytes with value 0x01.

If the DC is a GC server, the KCC constructs an additional NC replica graph (and creates nTDSConnection objects) for the config NC as above, except that only NC replicas that "are present" on GC servers are added to *R*.

The DC repeats the NC replica graph computation and nTDSConnection creation for each of the NC replica graphs above, this time assuming that no DC has failed. It does so by re-executing the steps as if the bit NTDSSETTINGS_OPT_IS_TOPL_DETECT_STALE_DISABLED were set in the options attribute of the site settings object for the local DC's site.

The net result of each DC executing this distributed algorithm is the following set of overlapping rings:

- For each NC, a ring containing each full replica in the site.
- For each NC, a ring containing each NC replica (full or partial) in the site.
- A ring containing each GC server in the site.
- For each NC, a ring containing each full replica in the site that has not failed.
- For each NC, a ring containing each NC replica (full or partial) in the site that has not failed.

- A ring containing each GC server in the site that has not failed.

6.2.2.3 Intersite Connection Creation

This task computes an NC replica graph for each NC replica that "should be present" on the local DC or "is present" on any DC in the same site as the local DC. For each edge directed to an NC replica on such a DC from an NC replica on a DC in another site, the KCC reconciles its portion of the NC replica graph by creating an nTDSConnection object to "imply" that edge if one does not already exist.

If the site of the local DC has a site settings object *o* and the NTDSSETTINGS_OPT_IS_INTER_SITE_AUTO_TOPOLOGY_DISABLED bit is set in *o*!options, the KCC skips this task.

Like intrasite connection, intersite connection creation utilizes distributed algorithms—algorithms that rely upon each DC in the forest implementing the same algorithm and arriving at the same conclusions given the same inputs. However, the algorithms used for intersite connection creation are significantly more complex. Sufficient analysis of a given variation of this algorithm may yield that DCs implementing the variation are compatible with Windows DCs, but no such different-yet-compatible algorithm is known. To illustrate this point, consider the following simple example:

Assume a forest F that contains three DCs of the same domain in three distinct sites—DC1 in Site1, DC2 in Site2, and DC3 in Site3—where siteLink objects exist specifying that each site is connected to the other two sites with the same cost. DC1 and DC2 execute one implementation of the KCC, and DC3 executes a different implementation.

DC1 and DC2 determine that the three sites should be connected by a minimum cost spanning tree rooted at site3: both DC1 and DC2 replicate updates from DC3, assuming that DC3 replicates updates from DC1 and DC2.

DC3, because it is running a different implementation, determines that the three sites should be connected by a minimum cost spanning tree rooted at site1: DC3 replicates updates from DC1, assuming that DC2 replicates updates from DC1 and DC1 replicates updates from DC2 and DC3.

The minimum cost spanning trees chosen by all the DCs are equally valid. However, the fact that they did not arrive at the same conclusions results in a violation of the first requirement described in section 6.2.1:

- DC1 replicates updates from DC3.
- DC3 replicates updates from DC1.
- DC2 replicates updates from DC3 (and therefore transitively receives updates from DC1).
- Neither DC1 nor DC2 replicates updates from DC2.

Slight variations in algorithms may result in similar failures that appear only when given specific, complex combinations of inputs. For this reason, these algorithms are described to a high level of detail, and implementers must carefully analyze any deviations from them.

6.2.2.3.1 ISTG Selection

First, the KCC on a writable DC determines whether it acts as an ISTG for its site.

- Let *s* be the object such that *s*!lDAPDisplayName = nTDSDSA and classSchema in *s*!objectClass.

- Let *D* be the sequence of objects *o* in the site of the local DC such that *o*!objectCategory = *s*. *D* is sorted in ascending order by objectGUID.
- Let *o* be the site settings object for the site of the local DC, or NULL if no such *o* exists.
- Let *f* be the duration *o*!interSiteTopologyFailover seconds, or 2 hours if *o*!interSiteTopologyFailover is 0 or has no value.
- If *o* ≠ NULL and *o*!interSiteTopologyGenerator is not the nTDSDSA object for the local DC and *o*!interSiteTopologyGenerator is an element d_j of sequence *D*:
  - Let *c* be the cursor in the *pUpToDateVector* variable associated with the NC replica of the config NC such that *c.uuidDsa* = d_j!invocationId. If no such *c* exists *(No evidence of replication from current ITSG):*
    - Let *i* = *j*.
    - Let *t* = 0.
  - Else if the current time < *c.timeLastSyncSuccess* - *f* (*Evidence of time sync problem on current ISTG*):
    - Let *i* = 0.
    - Let *t* = 0.
  - Else (*Evidence of replication from current ITSG)*:
    - Let *i* = *j*.
    - Let *t* = *c.timeLastSyncSuccess*.
- Otherwise (*Nominate local DC as ISTG*):
  - Let *i* be the integer such that d_i is the nTDSDSA object for the local DC.
  - Let *t* = the current time.
- (*Compute a function that maintains the current ISTG if it is alive, cycles through other candidates if not*.) Let *k* be the integer (*i* + ((current time - *t*) / *o*!interSiteTopologyFailover)) MOD |*D*|.

  The local writable DC acts as an ISTG for its site if and only if d_k is the nTDSDSA object for the local DC. If the local DC does not act as an ISTG, the KCC skips the remainder of this task.

If the local DC does act as an ISTG and *o* exists but *o*!interSiteTopologyGenerator is not the dsname of the local DC's nTDSDSA object, the KCC performs an originating update to set *o*!interSiteTopologyGenerator to this value.

The KCC on an RODC always acts as an ISTG for itself.

6.2.2.3.2 Merge of kCCFailedLinks and kCCFailedLinks from Bridgeheads

The KCC on a writable DC attempts to merge the link and connection failure information from bridgehead DCs in its own site to help it identify failed bridgehead DCs.

For each nTDSDSA object *bh* with objectCategory nTDSDSA other than the local DC but in the local DC's site, if *bh* has a child nTDSConnection object *cn* such that *cn*!fromServer is a reference to an nTDSDSA object in a site other than the local DC's site, and *cn*! options does not contain

NTDSCONN_OPT_RODC_TOPOLOGY, the KCC adds the tuples from *bh*'s *kCCFailedConnections* and *kCCFailedLinks* to the tuples in those same variables on the local DC. It does so by calling in the sequence IDL_DRSBind, IDL_DRSGetReplInfo for DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES, IDL_DRSGetReplInfo for DS_REPL_INFO_KCC_DSA_LINK_FAILURES, and IDL_DRSUnbind.

If any of these calls fails, the KCC adds a tuple for *bh*!objectGUID to *kCCFailedConnections*.

For each DS_REPL_KCC_DSA_FAILUREW *d* it receives, the KCC updates its corresponding variable *v* (*kCCFailedLinks* for DS_REPL_INFO_KCC_DSA_LINK_FAILURES, *kCCFailedConnections* for DS_REPL_INFO_KCC_DSA_CONNECT_FAILURES) as follows:

- If a tuple *f* exists in *v* such that *f.UUIDDsa* = *d.uuidDsaObjGuid* and *f.FailureCount* ≠ 0:
  - Set *f.FailureCount* to MAX(*f.FailureCount, d.cNumFailures*)
  - Set *f.TimeFirstFailure* to MIN(*f.TimeFirstFailure, d.ftimeFirstFailure*)
  - Set *f.LastResult* to *d.dwLastResult*
- If a tuple *f* exists in *v* such that *f.UUIDDsa* = *d.uuidDsaObjGuid* and *f.FailureCount* = 0:
  - Set *f.FailureCount* to *d.cNumFailures*
  - Set *f.TimeFirstFailure* to *d.ftimeFirstFailure*
  - Set *f.LastResult* to *d.dwLastResult*
- If no tuple *f* exists in *v* such that *f.UUIDDsa* = *d.uuidDsaObjGuid*, add tuple *g* to *v* such that
  - *g.UUIDDsa* = *d.uuidDsaObjGuid*
  - *g.FailureCount* = *d.cNumFailures*
  - *g.TimeFirstFailure* = *d.ftimeFirstFailure*
  - *g.LastResult* = *d.dwLastResult*

6.2.2.3.3 Site Graph Concepts

For each NC with an NC replica that "should be present" on the local DC or "is present" on any DC in the same site as the local DC, the KCC constructs a site graph—a precursor to an NC replica graph. The site connectivity for a site graph is defined by objects of class interSiteTransport, siteLink, and siteLinkBridge in the config NC. The semantics of these objects are described in section 6.1.

The pseudocode in the next section maps these objects and the various constraints on these objects as follows.

KCC concept	Site graph concept
site	VERTEX
siteLink	MULTIEDGE
siteLinkBridge	MULTIEDGESET
interSiteTransport	MULTIEDGE.Type
A siteLink object may connect more than two sites.	All vertices in a MULTIEDGE are

KCC concept	Site graph concept
	treated as a fully connected subgraph.
siteLink object attributes: cost, schedule, options, and replInterval.	MULTIEDGE properties in its ReplInfo field: Cost, Schedule, Options, and Interval. As paths are formed, this information is aggregated.
siteLink objects of different interSiteTransports objects co-exist in the same graph and compete based on cost.	MULTIEDGEs with differing Types co-exist in the graph and in the spanning tree.
Only the siteLink objects referenced by a siteLinkBridge may be combined together to form aggregated paths, with the vertices in common acting as routers.	MULTIEDGEs in a MULTIEDGESET are considered transitive.
NTDSTRANSPORT_OPT_BRIDGES_REQUIRED bit in the options attribute of an interSiteTransport object.	If clear, a MULTIEDGESET is inferred that includes all MULTIEDGEs with the corresponding Type.
NTDSTRANSPORT_OPT_IGNORE_SCHEDULES bit in the options attribute of an interSiteTransport object	If set, all MULTIEDGEs with the corresponding Type have a Schedule that is NULL.
For a given NC, a site may contain one or more writable replicas and zero or more partial read-only replicas, zero writable replicas but one or more partial read-only replicas, or zero writable replicas and zero partial read-only replicas.	VERTEX.Color VERTEX.Color is RED, BLACK, or WHITE, respectively.
A full replica cannot replicate from a partial replica.	No edge exists from a black vertex to a red vertex.
For each NC other than the config NC and the schema NC, the path from a writable replica to another full replica utilizes only the RPC transport.	VERTEX.AcceptRedRed VERTEX.AcceptBlack If both vertices for a given edge are red, the edge's type must be in the AcceptRedRed set of both vertices. If one or both vertices for a given edge are black, the edge's type must be in the AcceptBlack set of both vertices.
A site without a bridgehead DC for a particular transport cannot replicate updates over that transport to or from DCs in other sites.	The vertex for such a site does not contain the corresponding type in its AcceptRedRed or AcceptBlack properties.

6.2.2.3.4 Connection Creation

The methods described in this section calculate a spanning tree for each NC replica graph and create corresponding nTDSConnection objects that "imply" the corresponding spanning tree edges.

This pseudocode utilizes a type SEQUENCE<X>, which is a sequence of values of a given type X. Values of type X may be appended to and removed from the sequence. If *s* is a value of type SEQUENCE<X>, *s*[*i*] is the *i*'th value in *s*, such that $0 \le i < |s|$.

It also references the types DWORD and GUID from [MS-DTYP] sections 2.2 and 2.3.4.

6.2.2.3.4.1 Types

The following new types are used to represent and to evaluate site graphs:

```
/***** REPL_INFO *****/
/* Replication parameters of a graph edge. */
struct REPLINFO {
    DWORD Cost;             /* Cost of network traffic between
                             * vertices; lower is preferred. */
    DWORD Interval;         /* Interval between replication attempts.
                             */
    DWORD Options;          /* siteLink object options bits. */
    SCHEDULE Schedule;      /* Schedule during which communication is
                               possible; NULL means "always". */
}

/***** COLOR *****/
/* Color of a vertex. */
enum COLOR {
    RED,    /* Site contains one or more full replicas. */
    BLACK,  /* Site contains no full replicas but one or more
             * partial replicas. */
    WHITE   /* Site contains no replicas. */
}

/***** VERTEX *****/
/* A vertex in the site graph. */
struct VERTEX {
    GUID ID;                        /* objectGUID of corresponding site
                                     * object. */
    SEQUENCE<GUID> EdgeIDs;         /* Edges currently being evaluated
                                     * for this vertex. */
    COLOR Color;                    /* Color of the vertex. */
    SEQUENCE<GUID> AcceptRedRed;    /* Edge types accepted when both
                                     * vertices are RED. */
    SEQUENCE<GUID> AcceptBlack;     /* Edge types accepted when one or
                                     * both vertices are BLACK. */
    REPLINFO ReplInfo;              /* Replication parameters. */
    int DistToRed;                  /* Distance in the spanning tree
                                     * from this vertex to the nearest
                                     * red vertex. */
    /* Dijkstra data */
    GUID RootID;                    /* The ID of the closest RED or
                                     * BLACK vertex. */
    bool Demoted;                   /* TRUE if vertex should be treated
                                     * as if Color is WHITE. */
    /* Kruskal data */
    GUID ComponentID;               /* The id of the graph component
                                     * this vertex is in. */
    int ComponentIndex;             /* The index of the graph
                                       component. */
}
```

```
/***** MULTIEDGE *****/
/* Fully connected subgraph of vertices. */
struct MULTIEDGE {
    GUID ID;                   /* objectGUID of corresponding siteLink
                                * object. */
    SEQUENCE<GUID> VertexIDs;  /* IDs of connected vertices. */
    GUID Type;                 /* Type (interSiteTransport
                                * objectGUID). */
    REPLINFO ReplInfo;         /* Replication parameters. */
    bool Directed;             /* TRUE if uni-directional, FALSE if
                                * bi-directional */
}

/***** MULTIEDGESET *****/
/* Set of transitively connected MULTIEDGEs. All edges within the set
 * have the same Type. */
struct MULTIEDGESET {
    GUID ID;                  /* objectGUID of corresponding
                               * siteLinkBridge object. */
    SEQUENCE<GUID> EdgeIDs;   /* IDs of connected edges. */
}

/***** GRAPH *****/
/* A site graph. */
struct GRAPH {
    SEQUENCE<VERTEX> Vertices;          /* All vertices, sorted by
                                         * ascending ID (site
                                         * objectGUID). */
    SEQUENCE<MULTIEDGE> Edges;          /* All edges. */
    SEQUENCE<MULTIEDGESET> EdgeSets;    /* All edge sets. */
}

/***** INTERNALEDGE *****/
/* Path found in the graph between two non-WHITE vertices. */
struct INTERNALEDGE {
    GUID V1ID, V2ID;    /* The endpoints of the path. */
    bool RedRed;        /* TRUE if and only both endpoints are red. */
    REPLINFO ReplInfo;  /* Combined replication info for the path. */
    GUID Type;          /* All path edges must have same type. */
}
```

6.2.2.3.4.2 Main Entry Point

The CreateIntersiteConnections method is the beginning of the control flow. This method invokes the remainder of the methods, directly or indirectly.

```
/***** CreateIntersiteConnections *****/
/* Computes an NC replica graph for each NC replica that "should be
 * present" on the local DC or "is present" on any DC in the same site
 * as the local DC. For each edge directed to an NC replica on such a
 * DC from an NC replica on a DC in another site, the KCC creates an
 * nTDSConnection object to imply that edge if one does not already
 * exist.
 *
 * OUT: keepConnections - A sequence of objectGUID values of
 *      nTDSConnection objects for edges that are directed to the
```

```
 *      local DC's site in one or more NC replica graphs.
 * RETURNS: TRUE if spanning trees were created for all NC replica
 *      graphs, otherwise FALSE.
 */
CreateIntersiteConnections(OUT SEQUENCE<GUID> keepConnections) : bool
{
    LET allConnected be TRUE
    SET keepConnections to an empty sequence of GUID

    LET crossRefList be the set containing each object o of class
    crossRef such that o is a child of the CN=Partitions child of the
    config NC

    FOR each crossRef object cr in crossRefList
        IF cr!enabled has a value and is false, or if FLAG_CR_NTDS_NC
        is clear in cr!systemFlags, skip cr.
        LET g be the GRAPH return of SetupGraph()

        /* Create nTDSConnection objects, routing replication traffic
         * around "failed" DCs. */
        LET foundFailedDC be a Boolean variable
        LET c be the Boolean return of CreateConnections(g, cr, TRUE,
        keepConnections, foundFailedDC)

        IF !c
            SET allConnected to FALSE
            IF foundFailedDC
                /* One or more failed DCs preclude use of the ideal NC
                 * replica graph. Add connections for the ideal graph.
                 */
                CALL CreateConnections(graph, cr, FALSE,
                keepConnections, foundFailedDCs)
            ENDIF
        ENDIF
    ENDFOR

    RETURN allConnected
}
```

6.2.2.3.4.3 Site Graph Construction

The following methods construct the initial site graph, comprising the vertices, multi-edges, and multi-edge sets corresponding to the site, siteLink, and siteLinkBridge objects (respectively) in the config NC.

```
/***** SetupGraph *****/
/* Set up a GRAPH, populated with a VERTEX for each site object, a
 * MULTIEDGE for each siteLink object, and a MUTLIEDGESET for each
 * siteLinkBridge object (or implied siteLinkBridge).
 *
 * RETURNS: A new graph. */
SetupGraph() : GRAPH
{
    LET vertexIDs be the sequence containing the objectGUID of each
    site object child of the CN=Sites child of the config NC
    LET g be the GRAPH return of CreateGraph(vertexIDs)
    LET localSite be the site object for the site of the local DC
```

```
    FOR each interSiteTransport object t that is a child of the
    CN=Inter-Site Transports child of the CN=Sites child of the config
    NC
        LET L be the set containing each siteLink object that is a
        child of t
        FOR each l in L
            APPEND CreateEdge(t!objectGUID, l) to g.Edges
        ENDFOR
        IF NTDSTRANSPORT_OPT_BRIDGES_REQUIRED bit is clear in
        t!options and NTDSSETTINGS_OPT_W2K3_BRIDGES_REQUIRED bit is
        clear in localSite!options
            APPEND CreateAutoEdgeSet(g, t!objectGUID, L) to g.EdgeSets
        ELSE
            FOR each siteLinkBridge object b that is a child of t
                APPEND CreateEdgeSet(g, t!objectGUID, b) to g.EdgeSets
            ENDFOR
        ENDIF
    ENDFOR
  RETURN g
}

/***** CreateGraph *****/
/* Create a GRAPH instance.
 * IN: vertexIDs - Set containing the ID of each vertex to add to the
 *      graph.
 * RETURNS: A new graph containing vertices with the specified IDs.
 */
CreateGraph(IN SEQUENCE<GUID> vertexIDs) : GRAPH
{
    LET g be a new GRAPH
    SORT vertexIDs in ascending order of objectGUID
    FOR each id in vertexIDs
        LET v be a new VERTEX
        SET v.ID to id
        APPEND v to g.Vertices
    ENDFOR
    RETURN g
}

/***** CreateEdge *****/
/* Create a MULTIEDGE instance.
 * IN: type - Type of edge to add.
 * IN: link - Corresponding siteLink object.
 * RETURNS: A new MULTIEDGE instance.
 */
CreateEdge(IN GUID type, IN siteLink link) : MULTIEDGE
{
    LET e be a new MULTIEDGE
    SET e.ID to link!objectGUID
    SET e.VertexIDs to be the set containing the objectGUID value of
    each site referenced by link!siteList
    SET e.ReplInfo.Cost to link!cost;
    SET e.ReplInfo.Options to link!options
    SET e.ReplInfo.Interval to link!replInterval
    IF link!schedule has a value
        SET e.ReplInfo.Schedule to link!schedule
    ELSE
        SET e.ReplInfo.Schedule to NULL
```

```
    EndIF
    SET e.Type to type
    SET e.Directed to FALSE
 RETURN e
}

/***** CreateAutoEdgeSet *****/
/* Create a MULTIEDGESET instance containing edges for all siteLink
 * objects.
 * INOUT: g - Site graph.
 * IN: type - Type of edges being connected.
 * IN: L - All siteLink objects.
 * RETURNS: A new MULTIEDGESET instance.
 */
CreateAutoEdgeSet(INOUT GRAPH g, IN GUID type,
    IN SET OF siteLink L) : MULTIEDGESET
{
    LET s be a new MULTIEDGESET
    SET s.ID to NULL GUID
    FOR each l in L
        LET e be the edge in g.Edges such that e.ID = l!objectGUID
        IF e.Type = type
            APPEND l!objectGUID to s.EdgeIDs
        ENDIF
    ENDFOR
    RETURN s
}

/***** CreateEdgeSet *****/
/* Create a MULTIEDGESET instance.
 * INOUT: g - Site graph.
 * IN: type - Type of edges being connected.
 * IN: b - Corresponding siteLinkBridge object.
 * RETURNS: A new MULTIEDGESET instance.
 */
CreateEdgeSet(INOUT GRAPH g, IN GUID type, IN siteLinkBridge b)
: MULTIEDGESET
{
    LET s be a new MULTIEDGESET
    SET e.ID to b!objectGUID
    FOR each DSNAME l in b!siteLinkList
        LET e be the edge in g.Edges such that e.ID = l!objectGUID
        IF e.Type = type
            APPEND l!objectGUID to s.EdgeIDs
        ENDIF
    ENDFOR
    RETURN s
}

/***** ColorVertices *****/
/* Color each vertex to indicate which kinds of NC replicas it
 * contains.
 * INOUT: g - Site graph.
 * IN: cr - crossRef for NC.
 * IN: detectFailedDCs - TRUE to detect failed DCs and route
 *      replication traffic around them, FALSE to assume no DC
 *      has failed.
 * RETURNS: TRUE if one or more failed DCs were detected,
 *      otherwise FALSE.
```

```
 */
ColorVertices(INOUT GRAPH g, IN crossRef cr,
    IN bool detectFailedDCs) : bool
{
    LET foundFailedDCs be FALSE

    FOR each v in g.Vertices
        LET s be the site object with objectGUID v.ID
        IF s contains one or more DCs with full replicas of the NC
        cr!nCName
            SET v.Color to COLOR.RED
        ELSEIF s contains one or more partial replicas of the NC
            SET v.Color to COLOR.BLACK
        ELSE
            SET v.Color to COLOR.WHITE
        ENDIF
    ENDFOR
    LET localSiteVertex be the vertex in graph.Vertices such that
    localSiteVertex.ID = objectGUID of the local DC's site object
   FOR each v in g.Vertices
        FOR each interSiteTransport object t that is a child of the
        CN=Inter-Site Transports child of the CN=Sites child of the
        config NC
            IF localSiteVertex.Color = COLOR.RED and t!name ≠ "IP"
            and FLAG_CR_NTDS_DOMAIN bit is set in cr!systemFlags
                Skip t
            ENDIF
            IF no edge e exists in g.Edges such that e.VertexIDs
            contains v.ID
                Skip t
            ENDIF
            LET partialReplicaOkay be TRUE if and only if
            localSiteVertex.Color = COLOR.BLACK

            LET bh be the result of GetBridgeheadDC(
            localSiteVertex.ID, cr, t, partialReplicaOkay,
            detectFailedDCs)
            IF bh = null
                /* No bridgehead DC is currently available. */
                SET foundFailedDCs to TRUE
                Skip t
            ENDIF
            APPEND t!objectGUID to v.AcceptRedRed
            APPEND t!objectGUID to v.AcceptBlack
        ENDFOR
    ENDFOR

    RETURN foundFailedDCs
}
```

6.2.2.3.4.4 Spanning Tree Computation

The following methods process the site graph and compute the minimum-cost spanning tree.

```
/***** GetSpanningTreeEdges *****/
/* Calculate the spanning tree and return the edges that include the
 * vertex for the local site.
```

```
 * INOUT: g - Site graph.
 * OUT: componentCount - Set to the number of graph components
 *      calculated by Kruskal's algorithm. If 1, all sites are
 *      connected by a spanning tree. Otherwise, one or more sites
 *      could not be connected in a spanning tree.
 * RETURNS: Edges that include the vertex for the local site.
 */
GetSpanningTreeEdges(INOUT GRAPH g, OUT int componentCount)
    : SET<MULTIEDGE>
{
    /* Phase I: Run Dijkstra's algorithm and build up a list of
     * internal edges, which are really just shortest-paths
     * connecting colored vertices.
     */
    LET internalEdges be an empty sequence of INTERNALEDGE

    FOR each s in g.EdgeSets
        LET edgeType be NULL GUID
        FOR each v in g.Vertices
            REMOVE all items from v.EdgeIDs
        ENDFOR

        FOR each edge e in g.Edges such that s.EdgeIDs contains e.ID
            SET edgeType to e.Type
            FOR each vertex v in g.Vertices such that e.VertexIDs
            contains v.ID
                APPEND e to v.Edges
            ENDFOR
        ENDFOR

        /* Run Dijkstra's algorithm with just the red vertices as
         * the roots */
        CALL Dijkstra(g, edgeType, FALSE)

        /* Process the minimum-spanning forest built by Dijkstra,
         * and add any inter-tree edges to our list of internal
         * edges */
        CALL ProcessEdgeSet(g, s, internalEdges)

        /* Run Dijkstra's algorithm with red and black vertices as
         * the root vertices */
        CALL Dijkstra(g, edgeType, TRUE)

        /* Process the minimum-spanning forest built by Dijkstra,
         * and add any inter-tree edges to our list of internal
         * edges */
        CALL ProcessEdgeSet(g, s, internalEdges)
    ENDFOR

    /* Process the implicit empty edge set */
    CALL SetupVertices(g)
    CALL ProcessEdgeSet(g, NULL, internalEdges)

    /* Phase II: Run Kruskal's Algorithm on the internal edges. */
    LET outputEdges be the result of Kruskal(g, internalEdges)

    /* Phase III: Post-process the output:
     *  - Traverse tree structure to find one-way black-black edges
     *  - Determine the component structure */
```

```
    FOR each v in g.Vertices
        IF v.Color = COLOR.RED
            SET v.DistToRed to 0
        ELSEIF there exists a path from v to a COLOR.RED vertex
            SET v.DistToRed to the length of the shortest such path
        ELSE
            SET v.DistToRed to MAX DWORD
        ENDIF
    ENDFOR
    SET componentCount to CountComponents(g)
    LET stEdgeList be CopyOutputEdges(g, outputEdges)

    RETURN stEdgeList
}

/***** GetBridgeheadDC *****/
/* Get a bridghead DC.
 * IN: siteObjectGUID - objectGUID of the site object representing
 *      the site for which a bridgehead DC is desired.
 * IN: cr - crossRef for NC to replicate.
 * IN: t - interSiteTransport object for replication traffic.
 * IN: partialReplicaOkay - TRUE if a DC containing a partial
 *      replica or a full replica will suffice, FALSE if only
 *      a full replica will suffice.
 * IN: detectFailedDCs - TRUE to detect failed DCs and route
 *      replication traffic around them, FALSE to assume no DC
 *      has failed.
 * RETURNS: nTDSDSA object for the selected bridgehead DC, or NULL if
 *      none is available.
 */
GetBridgeheadDC(IN GUID siteObjectGUID, IN crossRef cr,
    IN interSiteTransport t, IN bool partialReplicaOkay,
    IN bool detectFailedDCs) : nTDSDSA
{
    LET bhs be the result of GetAllBridgeheadDCs(siteObjectGUID, cr,
    t, partialReplicaOkay, detectFailedDCs)

    IF bhs is empty
        RETURN NULL
    ELSE
        RETURN bhs[0]
    ENDIF
}

/***** GetAllBridgeheadDCs *****/
/* Get all bridghead DCs satisfying the given criteria.
 * IN: siteObjectGUID - objectGUID of the site object representing
 *      the site for which bridgehead DCs are desired.
 * IN: cr - crossRef for NC to replicate.
 * IN: t - interSiteTransport object for replication traffic.
 * IN: partialReplicaOkay - TRUE if a DC containing a partial
 *      replica or a full replica will suffice, FALSE if only
 *      a full replica will suffice.
 * IN: detectFailedDCs - TRUE to detect failed DCs and route
 *      replication traffic around them, FALSE to assume no DC
 *      has failed.
 * RETURNS: nTDSDSA objects for available bridgehead DCs.
 */
GetAllBridgeheadDCs(IN GUID siteObjectGUID, IN crossRef cr,
```

```
    IN interSiteTransport t, IN bool partialReplicaOkay,
    IN bool detectFailedDCs) : SEQUENCE OF nTDSDSA
{
    LET bhs be an empty sequence of nTDSDSA objects
    LET s be the site object such that s!objectGUID = siteObjectGUID
    LET k be an object such that
    s!lDAPDisplayName = nTDSDSA and classSchema in s!objectClass
    LET allDCsInSite be the sequence of objects o that are
    descendants of s such that o!objectCategory = k

    FOR each dc in allDCsInSite
        IF t!bridgeheadServerListBL has one or more values and
        t!bridgeheadServerListBL does not contain a reference to the
        parent object of dc
            Skip dc
        ENDIF

        IF dc is in the same site as the local DC
            IF a replica of cr!nCName is not in the set of NC replicas
            that "should be present" on dc or a partial replica of the
            NC "should be present" but partialReplicasOkay = FALSE
                Skip dc
            ENDIF
        ELSE
            IF an NC replica of cr!nCName is not in the set of NC
            replicas that "are present" on dc or a partial replica of
            the NC "is present" but partialReplicasOkay = FALSE
                Skip dc
            ENDIF
        ENDIF

        IF AmIRODC() and cr!nCName corresponds to default NC then
          Let dsaobj be the nTDSDSA object of the dc
          IF  dsaobj.msDS-Behavior-Version < DS_BEHAVIOR_WIN2008
            Skip dc
          ENDIF
        ENDIF

        IF t!name ≠ "IP" and the parent object of dc has no value for
        the attribute specified by t!transportAddressAttribute
            Skip dc
        ENDIF

        IF BridgeheadDCFailed(dc!objectGUID, detectFailedDCs) = TRUE
            Skip dc
        ENDIF

        APPEND dc to bhs
    ENDFOR

    IF bit NTDSSETTINGS_OPT_IS_RAND_BH_SELECTION_DISABLED is set in
    s!options
        SORT bhs such that all GC servers precede DCs that are not GC
        servers, and otherwise by ascending objectGUID
    ELSE
        SORT bhs in a random order
    ENDIF

    RETURN bhs
```

```
}

/***** BridgeheadDCFailed *****/
/* Determine whether a given DC is known to be in a failed state.
 * IN: objectGUID - objectGUID of the DC's nTDSDSA object.
 * IN: detectFailedDCs - TRUE if and only failed DC detection is
 *      enabled.
 * RETURNS: TRUE if and only if the DC should be considered to be in a
 *      failed state.
 */
BridgeheadDCFailed(IN GUID objectGUID, IN bool detectFailedDCs) : bool
{
    IF bit NTDSSETTINGS_OPT_IS_TOPL_DETECT_STALE_DISABLED is set in
    the options attribute of the site settings object for the local
    DC's site
        RETURN FALSE
    ELSEIF a tuple z exists in the kCCFailedLinks or
    kCCFailedConnections variables such that z.UUIDDsa =
    objectGUID, z.FailureCount > 1, and the current time -
    z.TimeFirstFailure > 2 hours
        RETURN TRUE
    ELSE
        RETURN detectFailedDCs
    ENDIF
}

/***** SetupVertices *****/
/* Setup the fields of the vertices that are relevant to Phase I
 * (Dijkstra's Algorithm). For each vertex, set up its cost,
 * root vertex, and component. This defines the shortest-path
 * forest structures.
 * INOUT: graph - Site graph.
 */
SetupVertices(INOUT GRAPH g)
{
    FOR each v in g.Vertices
        IF v.Color = COLOR.WHITE
            SET v.ReplInfo.Cost to MAX DWORD
            SET v.RootID to NULL GUID
            SET v.ComponentID to NULL GUID
        ELSE
            SET v.ReplInfo.Cost to 0
            SET v.RootID to v.ID
            SET v.ComponentID to v.ID
        ENDIF

        SET v.ReplInfo.Interval to 0
        SET v.ReplInfo.Options to 0xFFFFFFFF
        SET v.ReplInfo.Schedule to NULL
        SET v.HeapLocation to STHEAP_NOT_IN_HEAP
        SET v.Demoted to FALSE
    ENDFOR
}

/***** Dijkstra *****/
/* Run Dijkstra's algorithm with the red (and possibly black) vertices
 * as the root vertices, and build up a shortest-path forest.
 * INOUT: g - Site graph.
 * IN: edgeType - Type of the edges in the current edge set.
```

```
 * IN: fIncludeBlack - If this is true, black vertices are also used
 *      as roots.
 */
Dijkstra(INOUT GRAPH g, IN GUID edgeType, IN bool fIncludeBlack)
{
    LET vs be the result of SetupDijkstra(g, edgeType, fIncludeBlack)
    WHILE vs is not empty
        LET c be the least ReplInfo.Cost of any vertex in vs
        LET u be the vertex in vs with the least ID of all
        vertices with ReplInfo.Cost = c
        REMOVE u from vs

        FOR each e in g.Edges such that u.EdgeIDs contains e.ID
            FOR each vertexId in e.VertexIDs
                LET v be the vertex in g.Vertices such that v.ID =
                vertexId
                CALL TryNewPath(g, vs, u, e, v)
            ENDFOR
        ENDFOR
    ENDWHILE
}

/***** SetupDijkstra *****/
/* Build the initial sequence for use with Dijkstra's algorithm. It
 * will contain the red and black vertices as root vertices, unless
 * these vertices accept no edges of the current edgeType, or unless
 * black vertices are not being including.
 * INOUT: g - Site graph.
 * IN: edgeType - Type of the edges in the current edge set.
 * IN: fIncludeBlack - If this is true, black vertices are also used
 *      as roots.
 * RETURNS: Sequence of vertices.
 */
SetupDijkstra(INOUT GRAPH g, IN GUID edgeType, IN bool fIncludeBlack)
    : SEQUENCE<VERTEX>
{
    CALL SetupVertices(g)
    LET vs be an empty sequence of VERTEX
    FOR each v in g.Vertices
        IF v.Color = COLOR.WHITE
            Skip v
        ENDIF

        IF (v.Color = COLOR.BLACK and fIncludeBlack = FALSE) or
        v.AcceptBlack does not contain edgeType or v.AcceptRedRed
        does not contain edgeType
            /* If black vertices are not being allowing, or if this
             * vertex accepts neither red-red nor black edges, then
             * 'demote' it to a WHITE vertex for the purposes of Phase
             * I. Note that the 'Color' member of the vertex structure
             * is not changed. */
            SET v.ReplInfo.Cost to MAX DWORD
            SET v.RootID to NULL GUID
            SET v.Demoted to TRUE
        ELSE
            APPEND v to vs
        ENDIF
    ENDFOR
```

```
    RETURN vs
}

/***** TryNewPath *****/
/* Helper function for Dijkstra's algorithm. A new path has been found
 * from a root vertex to vertex v. This path is (u->root, ..., u, v).
 * Edge e is the edge connecting u and v. If this new path is better
 * (in this case cheaper, or has a longer schedule), update v to use
 * the new path.
 * INOUT: g - Site graph.
 * INOUT: vs - Vertices being evaluated.
 * IN: u - Vertex connected by e to v.
 * IN: e - Edge between u and v.
 * INOUT: v - Vertex connected by e to u.
 */
TryNewPath(INOUT GRAPH g, INOUT SEQUENCE<VERTEX> vs, IN VERTEX u,
    IN MULTIEDGE e, INOUT VERTEX v)
{
    LET newRI be an empty REPLINFO
    LET fIntersect be the result of CombineReplInfo(g, u.ReplInfo,
        Edge.ReplInfo, OUT newRI)

    IF newRI.Cost > v->ReplInfo.Cost
        RETURN
    ENDIF

    IF newRI.Cost < v.ReplInfo.Cost and fIntersect = FALSE
        RETURN
    ENDIF

    LET newDuration be the total duration newRI.Schedule shows as
    available
    LET oldDuration be the total duration v.ReplInfo.Schedule shows as
    available

    IF newRI.cost < v.ReplInfo.Cost or newDuration > oldDuration
        /* The new path to v is either cheaper or has a longer
         * schedule. Update v with its new root vertex, cost, and
         * replication info. */
        SET v.RootID to u.RootID
        SET v.ComponentID to u.ComponentID
        SET v.ReplInfo to newRI
        APPEND v to vs
    ENDIF
}

/***** CombineReplInfo *****/
/* Merge schedules, replication intervals, options and costs.
 * INOUT: g - Site graph.
 * IN: a - Replication info to combine with b.
 * IN: b - Replication info to combine with a.
 * OUT: c - Combination of a and b.
 * RETURNS: TRUE if schedules intersect, FALSE if they don't.
 */
CombineReplInfo(INOUT GRAPH g, IN REPLINFO a, IN REPLINFO b,
    OUT REPLINFO c) : bool
{
    LET s be the schedule that is the intersection of a.Schedule and
    b.Schedule, such that a given time is available in c if and only
```

```
    if that time is available in both a.Schedule and b.Schedule

    IF s has no available time
        RETURN FALSE
    ENDIF

    IF a.Cost + b.Cost overflows
        SET c.Cost to MAX DWORD
    ELSE
        SET c.Cost to a.Cost + b.Cost
    ENDIF

    SET C.Interval to maximum of a.Interval and b.Interval
    SET C.Options to a.Options BITWISE-AND b.Options
    SET C.Schedule = s

    RETURN TRUE
}

/***** ProcessEdgeSet *****/
/* After running Dijkstra's algorithm to determine the shortest-path
 * forest, examine all edges in this edge set. Find all inter-tree
 * edges, from which to build the list of 'internal edges', which
 * will later be passed on to Kruskal's algorithm.
 * INOUT: g - Site graph.
 * IN: s - Edge set, or NULL for the implicit edge set with no edges.
 * INOUT: internalEdges - Sequence to which to add new internal edges.
 */
ProcessEdgeSet(INOUT GRAPH g, IN MULTIEDGESET s,
    INOUT SEQUENCE<INTERNALEDGE> internalEdges)
{
    IF s = NULL
        FOR each e in g.Edges
            FOR each v in g.Vertices such that e.VertexIDs contains
            v.ID
                CALL CheckDemoteOneVertex(v, e.Type)
            ENDFOR
            CALL ProcessEdge(g, e, internalEdges)
            FOR each v in g.Vertices such that e.VertexIDs contains
            v.ID
                CALL UndemoteOneVertex(v)
            ENDFOR
        ENDFOR
    ELSE
        FOR each e in g.Edges such s.EdgeIDs contains e.ID
            CALL ProcessEdge(g, e, internalEdges)
        ENDFOR
    ENDIF
}

/***** CheckDemoteOneVertex *****/
/* Demote one vertex if necessary
 * INOUT: v - Vertex to check and possibly demote.
 * IN: edgeType - Type of edge being processed.
 */
CheckDemoteOneVertex(INOUT VERTEX v, IN GUID edgeType)
{
    IF v.Color = COLOR.WHITE
        RETURN
```

```
    ENDIF

    IF v.AcceptBlack does not contain edgeType and v.AcceptRedRed does
    not contain edgeType
        /* If this vertex accepts neither red-red nor black edges,
         * then 'demote' it to a WHITE vertex for the purposes of
         * Phase I. Note that the 'Color' member of the vertex
         * structure is not changed. */
        SET v.ReplInfo.Cost to MAX DWORD
        SET v.RootID to NULL GUID
        SET v.Demoted to TRUE
    ENDIF
}

/**** UndemoteOneVertex ******/
/* Clear the demoted state of a vertex
 * INOUT: v - Vertex to 'undemote'.
 */
UndemoteOneVertex(INOUT VERTEX v)
{
    IF v.Color = COLOR.WHITE
        RETURN
    ENDIF

    SET v.ReplInfo.Cost to 0
    SET v.RootID to v.ID
    SET v.Demoted to FALSE
}

/***** ProcessEdge *****/
/* After running Dijkstra's algorithm, this function examines a
 * multi-edge and adds internal edges between every tree connected by
 * this edge.
 * INOUT: g - Site graph.
 * IN: e - Multi-edge to examine.
 * INOUT: internalEdges - Sequence to which to add any new internal
 * edges.
 */
ProcessEdge(INOUT GRAPH g, IN MULTIEDGE e,
    INOUT SEQUENCE<INTERNALEDGE> internalEdges)
{
    /* Find the best vertex to be the 'root' of this multi-edge. */
    LET vs be a sequence containing each vertex v such that
    e.VertexIDs contains v.ID

    SORT vs such that RED vertices precede BLACK vertices, a vertex
    with lower ReplInfo.Cost precedes a vertex with higher
    ReplInfo.Cost if both vertices have the same Color, and a vertex
    with a lower ID precedes a vertex with higher ID if both vertices
    have the same Color and ReplInfo.Cost

    LET bestV be vs[0]

    /* Add to internalEdges an edge from every colored vertex to
       bestV.*/
    FOR each vertex v in g.Vertices such that e.VertexIDs contains v
        IF v.ComponentID ≠ NULL GUID and v.RootID ≠ NULL GUID
            Skip v
        ENDIF
```

```
        /* Only add this edge if it is a valid inter-tree edge.
         * (The two vertices must be reachable from the root vertices,
         * and in different components.) */
        IF bestV.ComponentID ≠ NULL GUID and bestV.RootID ≠ NULL GUID
        and v.ComponentID ≠ NULL GUID and bestV.RootID ≠ NULL GUID
        and bestV.ComponentID ≠ v.ComponentID
            CALL AddIntEdge(g, internalEdges, e, bestV, v)
        ENDIF
    ENDFOR
}

/***** AddIntEdge *****/
/* Add an edge to the list of edges that will be processed with
 * Kruskal's.
 * The endpoints are in fact the roots of the vertices to pass in, so
 * the endpoints are always colored vertices.
 * INOUT: g - Site graph.
 * INOUT: internalEdges - Sequence to which to add the new internal
 *      edge.
 * IN: e - Existing edge being examined.
 * IN: v1 - Vertex to connect with new internal edge.
 * IN: v2 - Vertex to connect with new internal edge.
 */
AddIntEdge(INOUT GRAPH g, INOUT SEQUENCE<INTERNALEDGE> internalEdges,
    IN MULTIEDGE e, IN VERTEX v1, IN VERTEX v2)
{
    /* The edge that is passed on to Kruskal's algorithm actually goes
     * between the roots of the two shortest-path trees. */
    LET root1 be the vertex in g.Vertices such that root1.ID =
    v1.RootID
    LET root2 be the vertex in g.Vertices such that root2.ID =
    v2.RootID

    /* Check if both endpoints will allow this type of edge */
    IF root1.Color = COLOR.RED and root2.Color = COLOR.RED
        LET redRed be TRUE
    ELSE
        LET redRed be FALSE
    ENDIF

    IF redRed = TRUE
        IF root1.AcceptRedRed does not contain e.Type or
        root2.AcceptRedRed does not contain e.Type
            RETURN
        ENDIF
    ELSE
        IF root1.AcceptBlack does not contain e.Type or
        root2.AcceptBlack does not contain e.Type
            RETURN
        ENDIF
    ENDIF

    /* Combine the schedules of the path from root1 to v1, root2 to
     * v2, and edge e */
    LET ri be an empty REPLINFO
    LET ri2 be an empty REPLINFO
    IF CombineReplInfo(g, v1.ReplInfo, v2.ReplInfo, OUT ri) = FALSE
    or CombineReplInfo(g, ri, e.ReplInfo, OUT ri2) = FALSE
```

```
            RETURN
        ENDIF

        /* Set up the internal simple edge from root1 to root2 */
        LET newIntEdge be an empty INTERNALEDGE
        SET newIntEdge.V1ID to root1.ID
        SET newIntEdge.V2ID to root2.ID
        SET newIntEdge.RedRed to redRed
        SET newIntEdge.ReplInfo to ri2
        SET newIntEdge.Type to e.Type

        /* Sort newIntEdge's vertices by ID */
        IF newIntEdge.V1ID > newIntEdge.V2ID
            Swap newIntEdge.V1ID and newIntEdge.V2ID
        ENDIF

        IF internalEdges does not contain an INTERNALEDGE that is
        identical to newIntEdge
            APPEND newIntEdge to internalEdges
        ENDIF
    }

    /***** Kruskal *****/
    /* Run Kruskal's minimum-cost spanning tree algorithm on the internal
     * edges (that represent shortest paths in the original graph between
     * colored vertices).
     * INOUT: g - Site graph.
     * INOUT: internalEdges - Edges between trees.
     * RETURNS: Spanning tree edges for the vertex representing the local
     *      DC's site.
     */
    Kruskal(INOUT GRAPH g, INOUT SEQUENCE<INTERNALEDGE> internalEdges)
        : SEQUENCE<MULTIEDGE>
    {
        FOR each v in g.Vertices
            REMOVE all items from v.EdgeIDs
        ENDFOR

        SORT internalEdges by (descending RedRed, ascending ReplInfo.Cost,
        descending available time in ReplInfo.Schedule, ascending V1ID,
        ascending V2ID, ascending Type)

        LET numExpectedTreeEdges be the count of vertices v in g.Vertices
        such that v.Color = COLOR.RED or v.Color = COLOR.WHITE
        LET cSTEdges be 0
        LET outputEdges be an empty sequence of MULTIEDGE

        WHILE internalEdges is not empty and cSTEdges <
        numExpectedTreeEdges
            LET e be internalEdges[0]

            /* Cycles in the spanning tree must be prevented. If edge e
             * is to be added, its endpoints must be in different
             * components. */
            LET comp1 be the return of GetComponentID(g, e.V1ID)
            LET comp2 be the return of GetComponentID(g, e.V2ID)
            IF comp1 ≠ comp2
                /* Add spanning tree edge. */
                INCREMENT cSTEdges by 1
```

```
            CALL AddOutEdge(g, outputEdges, e)

            /* Combine the two connected components. */
            LET v be the vertex in g.Vertices such that v.ID = comp1
            SET v.ComponentID to comp2
        ENDIF

        REMOVE e from internalEdges
    ENDWHILE

    RETURN outputEdges
}

/***** GetComponentID *****/
/* Returns the id of the component containing vertex v by traversing
 * the up-tree implied by the component pointers.
 * INOUT: g - Site graph.
 * INOUT: v - Vertex for which the component ID is desired.
 * RETURNS: The component ID of v.
 */
GetComponentID(INOUT GRAPH g, INOUT VERTEX v) : GUID
{
    /* Find root of the up-tree created by component pointers */
    LET u be v
    WHILE u.ComponentID ≠ u.ID
        LET id be u.ComponentID
        SET u to the vertex in g.Vertices such that u.ID = id
    ENDWHILE
    LET root be u.ID

    /* Compress the path to the root */
    SET u to v
    WHILE u.ComponentID ≠ u.ID
        LET id be u.ComponentID
        LET w be the vertex in g.Vertices such that w.ID = id
        SET u.ComponentID to root
        SET u to w
    ENDWHILE

    RETURN root
}

/***** AddOutEdge *****/
/* A new edge, e, has been found for the spanning tree edge. Add this
 * edge to the list of output edges.
 * INOUT: g - Site graph.
 * INOUT: outputEdges - Sequence to which to add the output edge.
 * IN: e - Edge to add.
 */
AddOutEdge(INOUT GRAPH g, INOUT SEQUENCE<MULTIEDGE> outputEdges,
    IN INTERNALEDGE e)
{
    LET v1 be the vertex in g.Vertices such that v1.ID = e.V1ID
    LET v2 be the vertex in g.Vertices such that v2.ID = e.V2ID

    /* Create an output multi edge */
    LET ee be an empty MULTIEDGE
    SET ee.Directed to FALSE
    APPEND v1.ID to ee.VertexIDs
```

```
    APPEND v2.ID to ee.VertexIDs
    SET ee.Type to e.Type
    SET ee.ReplInfo to e.ReplInfo
    APPEND ee to outputEdges

    /* Also add this new spanning-tree edge to the edge lists of
     * its endpoints. */
    APPEND ee to v1.EdgeIDs
    APPEND ee to v2.EdgeIDs
}

/***** CountComponents *****/
/* Count the number of components. A component is considered to be a
 * bunch of colored vertices that are connected by the spanning tree.
 * Vertices whose component id is the same as their vertex id are the
 * root of a connected component.
 *
 * When a root of a component has been found, record its 'component
 * index'. The component indices are a contiguous sequence of numbers
 * that uniquely identify a component.
 *
 * INOUT: g - Site graph.
 * RETURNS: Number of components.
 */
CountComponents(INOUT GRAPH g) : int
{
    LET numComponents be 0
    FOR each v in g.Vertices
        IF v.Color = COLOR.WHITE
            Skip v
        ENDIF

        LET compId be the result of GetComponentID(g, v)
        IF compId = v.ID
            /* It's a component root */
            SET v.ComponentIndex to numComponents
            Increment numComponents by 1
        ENDIF
    ENDFOR

    RETURN numComponents
}

/***** CopyOutputEdges *****/
/* Copy all spanning tree edges from outputEdges that contain the
 * vertex for DCs in the local DC's site.
 * INOUT: g - Site graph.
 * IN: outputEdges - All spanning tree edges.
 * RETURNS: Spanning tree edges for DCs in the local DC's site.
 */
CopyOutputEdges(INOUT GRAPH g, IN SEQUENCE<MULTIEDGE> outputEdges)
    : SEQUENCE<MULTIEDGE>
{
    LET s be an empty sequence of MULTIEDGE
    LET vid be the objectGUID of site object for the local DC's site
    FOR each e in outputEdges
        LET v be the vertex in g.Vertices such that v.ID =
        e.VertexIDs[0]
        LET w be the vertex in g.Vertices such that w.ID =
```

```
        e.VertexIDs[1]

        IF v.ID = vid or w.ID = vid
            /* Check if this edge meets the criteria of a 'directed
             * edge'. */
            IF (v.Color = COLOR.BLACK or w.Color = COLOR.BLACK) and
            v.DistToRed ≠ MAX DWORD
                SET e.Directed to TRUE

                /* Swap the vertices so that e->vertexNames[0] is
                 * closer to a red vertex than e->vertexNames[1]. */
                IF w.DistToRed < v.DistToRed
                    Swap e.VertexIDs[0] and e.VertexIDs[1]
                ENDIF
            ENDIF

            APPEND e to s
        ENDIF
    ENDFOR

    RETURN s
}
```

6.2.2.3.4.5 nTDSConnection Creation

The following methods create nTDSConnection objects to "imply" the minimum-cost spanning tree edges for which no nTDSConnection objects yet exist.

```
/***** CreateConnections *****/
/* Construct an NC replica graph for the NC identified by the given
 * crossRef, then create any additional nTDSConnection objects
 * required.
 *
 * INOUT: g - Site graph.
 * IN: cr - crossRef object for NC.
 * IN: detectFailedDCs - TRUE to detect failed DCs and route
 *      replication traffic around them, FALSE to assume no DC
 *      has failed.
 * INOUT: keepConnections - Sequence to which to add any connections
 *      deemed to be "in use".
 * OUT: foundFailedDCs - Set to TRUE if one or more failed DCs
 *      were detected, otherwise set to FALSE.
 * RETURNS: TRUE if the resulting NC replica graph connects
 *      all sites that need to be connected.
 */
CreateConnections(INOUT GRAPH g, IN crossRef cr,
    IN bool detectFailedDCs, INOUT SEQUENCE<GUID> keepConnectoins,
    OUT bool foundFailedDCs) : bool
{
    LET connected be a bool, initialized to true
    SET foundFailedDCs to the return of ColorVertices(g, cr,
    detectFailedDCs)


    LET localSiteVertex be the vertex in g.Vertices such that
    localSiteVertex.ID is the objectGUID of the local DC's site object
    IF localSiteVertex.Color = COLOR.WHITE
```

```
    /* No NC replicas for this NC in the site of the local DC,
     * so no nTDSConnection objects need be created. */
    return TRUE
ENDIF


LET componentCount be an integer
LET edges be the sequence of MULTIEDGE returned by
LET stEdgeList be the result of GetSpanningTreeEdges(graph,
    OUT componentCount)


IF componentCount > 1
    /* Not all sites could be connected by the spanning tree. */
    SET connected to false
ENDIF


LET partialReplicaOkay be TRUE if and only if
localSiteVertex.Color = COLOR.BLACK


FOR each edge e in stEdgeList
    /* Ignore directed edges not directed to our site. */
    IF e.Directed and e.VertexIDs[1] ≠ localSiteVertex.ID
        Skip e
    ENDIF


    IF e.VertexIDs[0] = localSiteVertex.ID
        LET otherSiteVertex be the vertex in g.Vertices such that
        otherSiteVertex.ID = e.VertexIDs[1]
    ELSE
        LET otherSiteVertex be the vertex in g.Vertices such that
        otherSiteVertex.ID = e.VertexIDs[0]
    ENDIF


    LET t be the interSiteTransport object with objectGUID e.Type
    LET rbh be the result of GetBridgeheadDC(otherSiteVertex.ID,
    cr, t, partialReplicaOkay, detectFailedDCs)
    /* RODC acts as an BH for itself */
    IF AmIRODC() then
      LET lbh be the nTDSDSA object of the local DC
    ELSE
      LET lbh be the result of GetBridgeheadDC(localSiteVertex.ID,
      cr, t, partialReplicaOkay, detectFailedDCs)
    ENDIF


    LET sched be a new SCHEDULE such that the first available time
    is that of e.ReplInfo.Schedule and each subsequent available
    time is e.ReplInfo.Interval minutes after the previous
    available time

    CALL CreateConnection(cr, rbh, t, lbh, e.ReplInfo, sched,
    partialReplicaOkay)
ENDFOR
```

```
    RETURN connected
}


/***** CreateConnection *****/
/* Create an nTDSConnection object with the given parameters if one
 * does not already exist.
 * IN: cr - crossRef object for the NC to replicate.
 * IN: rbh - nTDSDSA object for DC to act as the IDL_DRSGetNCChanges
 *      server (which is in a site other than the local DC's site).
 * IN: t - interSiteTransport object for the transport to use for
 *      replication traffic.
 * IN: lbh - nTDSDSA object for DC to act as the IDL_DRSGetNCChanges
 *      client (which is in the local DC's site).
 * IN: ri - Replication parameters (aggregated siteLink options, etc.)
 * IN: sch - Schedule specifying the times at which to begin
 *      replicating.
 * IN: detectFailedDCs - TRUE to detect failed DCs and route
 *      replication traffic around them, FALSE to assume no DC
 *      has failed.
 * IN: partialReplicaOkay - TRUE if bridgehead DCs containing partial
 *      replicas of the NC are acceptable.
 * INOUT: keepConnections - Sequence to which to add any connections
 *      deemed to be "in use".
 */
CreateConnection(IN crossRef cr, IN nTDSDSA rbh,
    IN interSiteTransport t, IN nTDSDSA lbh, IN REPLINFO ri,
    IN SCHEDULE sch, INOUT SEQUENCE<GUID> keepConnections)
{
    LET rsiteGuid be the objectGUID of the site object ancestor of rbh
    LET lsiteGuid be the objectGUID of the site object ancestor of lbh

    LET rbhsAll be the result of GetAllBridgeheadDCs(rsiteGuid, cr,
    t, partialReplicaOkay, FALSE)
    LET rbhsAvail be the result of GetAllBridgeheadDCs(rsiteGuid, cr,
    t, partialReplicaOkay, detectFailedDCs)
    LET lbhsAll be the result of GetAllBridgeheadDCs(lsiteGuid, cr,
    t, partialReplicaOkay, FALSE)
    LET lbhsAvail be the result of GetAllBridgeheadDCs(lsiteGuid, cr,
    t, partialReplicaOkay, detectFailedDCs)


    FOR each nTDSConnection object cn such that the parent of cn is
    a DC in lbhsAll and cn!fromServer references a DC in rbhsAll
        IF bit NTDSCONN_OPT_IS_GENERATED is set in cn!options and
        NTDSCONN_OPT_RODC_TOPOLOGY is clear in cn!options and
        cn!transportType references t
            IF bit NTDSCONN_OPT_USER_OWNED_SCHEDULE is clear in
            cn!options and cn!schedule ≠ sch
               Perform an originating update to set cn!schedule to sch
            ENDIF
            IF bits NTDSCONN_OPT_OVERRIDE_NOTIFY_DEFAULT and
            NTDSCONN_OPT_USE_NOTIFY are set in cn
                IF bit NTDSSITELINK_OPT_USE_NOTIFY is clear in
                ri.Options
                    Perform an originating update to clear bits
                    NTDSCONN_OPT_OVERRIDE_NOTIFY_DEFAULT and
                    NTDSCONN_OPT_USE_NOTIFY in cn!options
```

```
            ENDIF
        ELSE
            IF bit NTDSSITELINK_OPT_USE_NOTIFY is set in
            ri.Options
                Perform an originating update to set bits
                NTDSCONN_OPT_OVERRIDE_NOTIFY_DEFAULT and
                NTDSCONN_OPT_USE_NOTIFY in cn!options
            ENDIF
        ENDIF
        IF bit NTDSCONN_OPT_TWOWAY_SYNC is set in cn!options
            IF bit NTDSSITELINK_OPT_TWOWAY_SYNC is clear in
            ri.Options
                Perform an originating update to clear bit
                NTDSCONN_OPT_TWOWAY_SYNC in cn!options
            ENDIF
        ELSE
            IF bit NTDSSITELINK_OPT_TWOWAY_SYNC is set in
            ri.Options
                Perform an originating update to set bit
                NTDSCONN_OPT_TWOWAY_SYNC in cn!options
            ENDIF
        ENDIF
        IF bit NTDSCONN_OPT_DISABLE_INTERSITE_COMPRESSION is set
        in cn!options
            IF bit NTDSSITELINK_OPT_DISABLE_COMPRESSION is clear
            in ri.Options
                Perform an originating update to clear bit
                NTDSCONN_OPT_DISABLE_INTERSITE_COMPRESSION in
                cn!options
            ENDIF
        ELSE
            IF bit NTDSSITELINK_OPT_DISABLE_COMPRESSION is set in
            ri.Options
                Perform an originating update to set bit
                NTDSCONN_OPT_DISABLE_INTERSITE_COMPRESSION in
                cn!options
            ENDIF
        ENDIF
    ENDIF
ENDFOR

LET cValidConnections be 0
FOR each nTDSConnection object cn such that cn!parent is
a DC in lbhsAll and cn!fromServer references a DC in rbhsAll
    IF (bit NTDSCONN_OPT_IS_GENERATED is clear in cn!options or
        cn!transportType references t) and
    NTDSCONN_OPT_RODC_TOPOLOGY is clear in cn!options
        LET rguid be the objectGUID of the nTDSDSA object
        referenced by cn!fromServer
        LET lguid be (cn!parent)!objectGUID

        IF BridgeheadDCFailed(rguid, detectFailedDCs) = FALSE and
        BridgeheadDCFailed(lguid, detectFailedDCs) = FALSE
            Increment cValidConnections by 1
        ENDIF

        IF keepConnections does not contain cn!objectGUID
            APPEND cn!objectGUID to keepConnections
        ENDIF
```

```
            ENDIF
        ENDFOR


        IF cValidConnections = 0
            LET opt be NTDSCONN_OPT_IS_GENERATED
            IF bit NTDSSITELINK_OPT_USE_NOTIFY is set in ri.Options
                SET bits NTDSCONN_OPT_OVERRIDE_NOTIFY_DEFAULT and
                NTDSCONN_OPT_USE_NOTIFY in opt
            ENDIF
            IF bit NTDSSITELINK_OPT_TWOWAY_SYNC is set in ri.Options
                SET bit NTDSCONN_OPT_TWOWAY_SYNC opt
            ENDIF
            IF bit NTDSSITELINK_OPT_DISABLE_COMPRESSION is set in
            ri.Options
                SET bit NTDSCONN_OPT_DISABLE_INTERSITE_COMPRESSION in opt
            ENDIF

            Perform an originating update to create a new nTDSConnection
            object cn that is a child of lbh, cn!enabledConnection = TRUE,
            cn!options = opt, cn!transportType is a reference to t,
            cn!fromServer is a reference to rbh, and cn!schedule = sch

            APPEND cn!objectGUID to keepConnections
        ENDIF
    }
```

6.2.2.4 Removing Unnecessary Connections

This task deletes nTDSConnection objects that are not needed to imply edges in any NC replica graph.

Given an nTDSConnection object *cn*, if the DC with the nTDSDSA object *dc* that is the parent object of *cn* and the DC with the nTDSDA object referenced by *cn*!fromServer are in the same site, the KCC on *dc* deletes *cn* if all of the following are true:

- Bit NTDSCONN_OPT_IS_GENERATED is clear in *cn*!options.
- No site settings object *s* exists for the local DC's site, or bit NTDSSETTINGS_OPT_IS_TOPL_CLEANUP_DISABLED is clear in *s*!options.
- Another nTDSConnection object *cn2* exists such that *cn* and *cn2* have the same parent object, *cn*!fromServer = *cn2*!fromServer, and either
  - *cn*!whenCreated < *cn2*!whenCreated
  - *cn*!whenCreated = *cn2*!whenCreated and *cn*!objectGUID < *cn2*!objectGUID
- Bit NTDSCONN_OPT_RODC_TOPOLOGY is clear in *cn*!options

Given an nTDSConnection object *cn*, if the DC with the nTDSDSA object *dc* that is the parent object of *cn* and the DC with the nTDSDSA object referenced by *cn*!fromServer are in different sites, a KCC acting as an ISTG in *dc*'s site deletes *cn* if all of the following are true:

- Bit NTDSCONN_OPT_IS_GENERATED is clear in *cn*!options.
- *cn*!fromServer references an nTDSDSA object for a DC in a site other than the local DC's site.

- The *keepConnections* sequence returned by CreateIntersiteConnections() does not contain *cn*!objectGUID, or *cn* is "superseded by" (see below) another nTDSConnection *cn2* and *keepConnections* contains *cn2*!objectGUID.
- The return value of CreateIntersiteConnections() was true.
- Bit NTDSCONN_OPT_RODC_TOPOLOGY is clear in *cn*!options

An nTDSConnection *cn* is said to be "superseded by" another nTDSConnection *cn2* if both of the following are true:

- If *cn* implies a tuple in *r*!repsFrom, *cn2* also implies a tuple in *r*!repsFrom.
- If *s* is (*cn*!fromServer)!objectGUID and *t* is (*cn*!parent)!objectGUID, BridgeheadDCFailed(*s*, true) = false and BridgeheadDCFailed(*t*, true) = false.

6.2.2.5 Connection Translation

This task adjusts values of repsFrom abstract attributes of NC replicas on the local DC to match those "implied" by nTDSConnection objects.

If the NTDSDSA_OPT_DISABLE_NTDSCONN_XLATE bit is set in the value of the options attribute of the local DC's nTDSDSA object, the KCC skips this task.

First, the KCC inspects *n*!repsFrom for each NC replica *n* that "is present" or "should be present" on the local DC. If *n* is not an NC replica that "should be present" on the local DC, the KCC calls IDL_DRSReplicaDel to remove all tuples from *n*!repsFrom and to remove *n*.

Otherwise, for each tuple *t* in *n*!repsFrom, let *s* be the nTDSDSA object such that *s*!objectGUID = *t.uuidDsa*. Let *cn* be the nTDSConnection object such that *cn* is a child of the local DC's nTDSDSA object and *cn*!fromServer = *s* and *cn*!options does not contain NTDSCONN_OPT_RODC_TOPOLOGY, or NULL if no such *cn* exists. The KCC calls IDL_DRSReplicaDel to remove *t* from *n*!repsFrom if any of the following is true:

- *cn* = NULL.
- No NC replica of the NC "is present" on *s*.
- A writable replica of the NC "should be present" on the local DC, but a partial replica "is present" on *s*.

If the KCC did not remove *t* from *n*!repsFrom, it updates *t* if necessary to satisfy the following requirements. Such updates are typically required when the IDL_DRSGetNCChanges server has moved from one site to another—for example, to enable compression when the server is moved from the client's site to another site.

- *t.schedule* = *cn*!schedule
- Bit DRS_PER_SYNC is set in *t.replicaFlags* if and only if *cn*!schedule has a value *v* that specifies scheduled replication is to be performed at least once per week.
- Bit DRS_INIT_SYNC is set in *t.replicaFlags* if and only if *s* and the local DC's nTDSDSA object are in the same site or *s* is the FSMO role owner of one or more FSMO roles in the NC replica.
- If bit NTDSCONN_OPT_OVERRIDE_NOTIFY_DEFAULT is set in *cn*!options, bit DRS_NEVER_NOTIFY is set in *t.replicaFlags* if and only if bit NTDSCONN_OPT_USE_NOTIFY is

clear in *cn*!options. Otherwise, bit DRS_NEVER_NOTIFY is set in *t.replicaFlags* if and only if *s* and the local DC's nTDSDSA object are in different sites.

- Bit DRS_USE_COMPRESSION is set in *t.replicaFlags* if and only if *s* and the local DC's nTDSDSA object are not in the same site and the NTDSCONN_OPT_DISABLE_INTERSITE_COMPRESSION bit is clear in *cn*!options.
- Bit DRS_TWOWAY_SYNC is set in *t.replicaFlags* if and only if bit NTDSCONN_OPT_TWOWAY_SYNC is set in *cn*!options.
- Bits DRS_DISABLE_AUTO_SYNC and DRS_DISABLE_PERIODIC_SYNC are set in *t.replicaFlags* if and only if *cn*!enabledConnection = false.
- If s and the local DC's nTDSDSA object are in the same site, *cn*!transportType has no value, or the RDN of *cn*!transportType is CN=IP:
  - Bit DRS_MAIL_REP in *t.replicaFlags* is clear.
  - *t.uuidTransport* = NULL GUID.
  - *t.uuidDsa* = The **GUID-based DNS name** of *s*.
- Otherwise:
  - Bit DRS_MAIL_REP in *t.replicaFlags* is set.
  - If *x* is the object with dsname *cn*!transportType, *t.uuidTransport* = *x*!objectGUID.
  - Let *a* be the attribute identified by *x*!transportAddressAttribute. If *a* is the dNSHostName attribute, *t.uuidDsa* = the GUID-based DNS name of *s*. Otherwise, *t.uuidDsa* = (*s*!parent)!*a*.

Finally, the KCC calls IDL_DRSReplicaAdd to add a tuple *u* to *n*!repsFrom for each IDL_DRSGetNCChanges server "implied" by the nTDSConnection object children of the local DC's nTDSDSA object if such a *u* does not already exist. For each such nTDSConnection *cn*, a tuple *u* is implied if all of the following are true:

- *cn*!enabledConnection = true.
- *cn*!options does not contain NTDSCONN_OPT_RODC_TOPOLOGY.
- *cn*!fromServer references an nTDSDSA object.
- An NC replica of the NC "is present" on the DC to which the nTDSDSA object referenced by *cn*!fromServer corresponds.
- An NC replica of the NC "should be present" on the local DC.
- The NC replica on the DC referenced by *cn*!fromServer is a writable replica or the NC replica that "should be present" on the local DC is a partial replica.
- The NC is not a domain NC, the NC replica that "should be present" on the local DC is a partial replica, *cn*!transportType has no value, or *cn*!transportType has an RDN of CN=IP.

If tuple *u* is implied, its fields satisfy each of the criteria defined above for tuple *t* when *t* is updated using IDL_DRSReplicaModify, plus the following additional criteria:

- *u.uuidDsa* = the objectGUID of the nTDSDSA object referenced by *cn*!fromServer.

- *u.uuidInvocId*, *u.usnVec*, *u.consecutiveFailure, u.timeLastSuccess, u.timeLastAttempt*, and *u.resultLastAttempt* are 0.

If an attempt to contact another DC is made and it fails, the KCC adds a tuple for that DC to the local DC's *kCCFailedConnections* variable.

6.2.2.6 Remove Unneeded kCCFailedLinks and kCCFailedConnections Tuples

This task removes tuples from *kCCFailedLinks* and *kCCFailedConnections* that are not as inputs to future runs.

For each tuple *f* in *kCCFailedLinks*, if *f.FailureCount* = 0 the KCC removes *f*.

For each tuple *k* in *kCCFailedConnections*, if no attempt was made in this run to contact the corresponding DC (the DC with nTDSDSA object *o* such that *o*!objectGUID = *k.UUIDDsa*) or an attempt was made and it was successful, the KCC removes *k*.

6.2.2.7 Updating the RODC NTFRS Connection Object

This task runs only when the local DC is an RODC. It updates the RODC NTFRS connection object.

Given an nTDSConnection object *cn1*, such that *cn1*!options contains NTDSCONN_OPT_RODC_TOPOLOGY, and another nTDSConnection object *cn2*, such that *cn2*!options does not contain NTDSCONN_OPT_RODC_TOPOLOGY, modify *cn1* to ensure that the following is true:

- *cn1*!fromServer = *cn2*!fromServer
- *cn1*!schedule = *cn2*!schedule

If no such *cn2* can be found, *cn1* is not modified. If no such *cn1* can be found, nothing is modified by this task.

6.3 Publishing and Locating a Domain Controller

Active Directory is a distributed service, which means that when a client needs Active Directory services, it may be able to receive those services from any of a number of equivalent DCs. Clients cannot be expected to know in advance the names of all possible suitable DCs. This implies a need for a protocol by which clients can dynamically discover which DCs are configured, operational, and reachable such that they could supply the needed services, and to choose among those DCs.

Locating a DC works differently for AD DS than for AD LDS.

- **AD DS**

  The process of locating AD DS DCs is performed in two separate ways, one based on NetBIOS and **mailslots**, the other based on DNS and LDAP. While the network representations of the two ways are radically different, they are functionally very similar. It is worthwhile to explain the conceptual similarities and motivations before starting a detailed discussion of the differing implementation details.

  The NetBIOS version is required for compatibility with older clients (such as Windows NT 4.0 operating system) that are not aware of Active Directory. Being based on NetBIOS, however, it is dependent either on network **broadcasts** or on the deployment of a **NetBIOS Name Service (NBNS)** infrastructure; broadcasts cannot be used in a wide area network where they are

typically blocked. The DNS-based version makes no use of broadcasts and includes extra support for determining network locality.

Both versions of the protocol work in two phases. In the first phase, DCs publish data about themselves (in DNS, or in NBNS, or by local configuration of the responder to NetBIOS broadcasts, depending on which version of publication is being used). In the second phase, clients look up this static data to determine a set of possible DCs and then send small messages to some or all of the set, examining the responses in order to determine liveness, reachability, and suitability. Given their conceptual similarity to an Internet Control Message Protocol (ICMP) ping message, these small messages are referred to as "LDAP ping" and "mailslot ping".

Sections 6.3.1 through 6.3.7 specify the precise details about the data that servers publish about themselves. These sections also specify the precise details about the two "ping" protocols.

- **AD LDS**

  An AD LDS DC does not publish data about itself in name services as in the case of an AD DS DC. An AD LDS DC that is joined to an AD DS domain SHOULD publish itself by creating an object in AD DS; a client MAY then query AD DS and select an AD LDS DC based on the query results. The information that an AD LDS DC publishes about itself is described in section 6.3.8. An AD LDS DC that is not joined to an AD DS domain does not publish itself at all; a client must possess an AD LDS server's IP address or host name and port number. This protocol does not provide a means for a client to obtain this information.

6.3.1 Structures and Constants

6.3.1.1 NETLOGON_NT_VERSION Options Bits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
VGC	VL	VIP	VPDC	X	X	X	VNT4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	VCS	V5EP	V5EX	V5	V1

Note The bits are presented in big-endian byte order.

V1 (NETLOGON_NT_VERSION_1, 0x00000001): Unless overridden by **V5**, **V5EX**, or **V5EP**, this bit instructs the server to respond to LDAP ping (section 6.3.3) and mailslot ping (section 6.3.5) using either the NETLOGON_SAM_LOGON_RESPONSE_NT40 structure or the NETLOGON_PRIMARY_RESPONSE structure for the PDC.

V5 (NETLOGON_NT_VERSION_5, 0x00000002): Unless overridden by **V5EX** or **V5EP**, this bit instructs the server to respond to LDAP ping and mailslot ping using the NETLOGON_SAM_LOGON_RESPONSE structure.

V5EX (NETLOGON_NT_VERSION_5EX, 0x00000004): Unless overridden by **V5EP**, this bit instructs the server to respond to LDAP ping and mailslot ping using the NETLOGON_SAM_LOGON_RESPONSE_EX structure.

V5EP (NETLOGON_NT_VERSION_5EX_WITH_IP, 0x00000008): Instructs the server to respond to mailslot ping using the NETLOGON_SAM_LOGON_RESPONSE_EX structure and also to return the IP address of the server in the response.

VCS (NETLOGON_NT_VERSION_WITH_CLOSEST_SITE, 0x00000010): Indicates that the client is querying for the closest site information. This flag is interpreted by DCs with DC functional level greater than or equal to DS_BEHAVIOR_WIN2008.

VNT4 (NETLOGON_NT_VERSION_AVOID_NT4EMUL, 0x01000000): Forces the server to respond to an LDAP ping and to honor all the NetLOGON_NT_VERSION options that the client specifies in the LDAP ping or mailslot ping. The client specifies NETLOGON_NT_VERSION_AVOID_NT4EMUL to force the server to respond to an LDAP ping even if the server is configured to ignore LDAP ping requests, and to honor all the NETLOGON_NT_VERSION options specified by the client in a mailslot ping, even if the server is configured to assume NETLOGON_NT_VERSION_1 in mailslot ping requests.

VPDC (NETLOGON_NT_VERSION_PDC, 0x10000000): Indicates that the client is querying for a PDC.

VIP (NETLOGON_NT_VERSION_IP, 0x20000000): Obsolete, ignored.

VL (NETLOGON_NT_VERSION_LOCAL, 0x40000000): Indicates that the client is the local machine.

VGC (NETLOGON_NT_VERSION_GC, 0x80000000): Indicates that the client is querying for a GC.

X: Reserved for future expansion. The client MUST set it to 0, and the server MUST ignore it.

6.3.1.2 DS_FLAG Options Bits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
FF	FDM	FDNS	X	X	X	X	X	X	X	X	X	X	X	X	X	FW9	FW8	FWS	FFS	FSS	FN	FGT	FW	FC	FT	FK	FD	FL	FG	X	FP

Note The bits are presented in big-endian byte order.

FP (DS_PDC_FLAG, 0x00000001): The server holds the PDC FSMO role (*PdcEmulationMasterRole*). FSMO roles are defined in section 3.1.1.1.11. Certain updates can be performed only on the holder of the PDC FSMO role (see Updates Performed Only on FSMOs (section 3.1.1.5.1.8)).

FG (DS_GC_FLAG, 0x00000004): The server is a GC server and will accept and process messages directed to it on the global catalog ports (see section 3.1.1.3.1.10).

FL (DS_LDAP_FLAG, 0x00000008): The server is an LDAP server.

FD (DS_DS_FLAG, 0x00000010): The server is a DC.

FK (DS_KDC_FLAG, 0x00000020): The server is running the Kerberos Key Distribution Center service.

FT (DS_TIMESERV_FLAG, 0x00000040): The Win32 Time Service, as specified in [MS-W32T], is present on the server.

FC (DS_CLOSEST_FLAG, 0x00000080): The server is in the same site as the client. This is a hint to the client that it is well-connected to the server in terms of speed.

FW (DS_WRITABLE_FLAG, 0x00000100): Indicates that the server is not an RODC. As described in section 3.1.1.1.9, all NC replicas hosted on an RODC do not accept originating updates.

FGT (DS_GOOD_TIMESERV_FLAG, 0x00000200): The server is a reliable time server.

FN (DS_NDNC_FLAG, 0x00000400): The NC is an application NC.

FSS (DS_SELECT_SECRET_DOMAIN_6_FLAG, 0x00000800): The server is an RODC.

FFS (DS_FULL_SECRET_DOMAIN_6_FLAG, 0x00001000): The server is a writable DC, not running Windows 2000 Server operating system or Windows Server 2003 operating system.

FWS (DS_WS_FLAG, 0x00002000): The Active Directory Web Service, as specified in [MS-ADDM], is present on the server.

FW8 (DS_DS_8_FLAG, 0x00004000): The server is not running Windows 2000 operating system, Windows Server 2003, Windows Server 2008 operating system, or Windows Server 2008 R2 operating system.

FW9 (DS_DS_9_FLAG, 0x00008000): The server is not running Windows 2000, Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, or Windows Server 2012 operating system.

FDNS (DS_DNS_CONTROLLER_FLAG, 0x20000000): The server has a DNS name.

FDM (DS_DNS_DOMAIN_FLAG, 0x40000000): The NC is a default NC.

FF (DS_DNS_FOREST_FLAG, 0x80000000): The NC is the forest root.

X: Reserved for future expansion. The server MUST return zero, and the client MUST ignore.

6.3.1.3 Operation Code

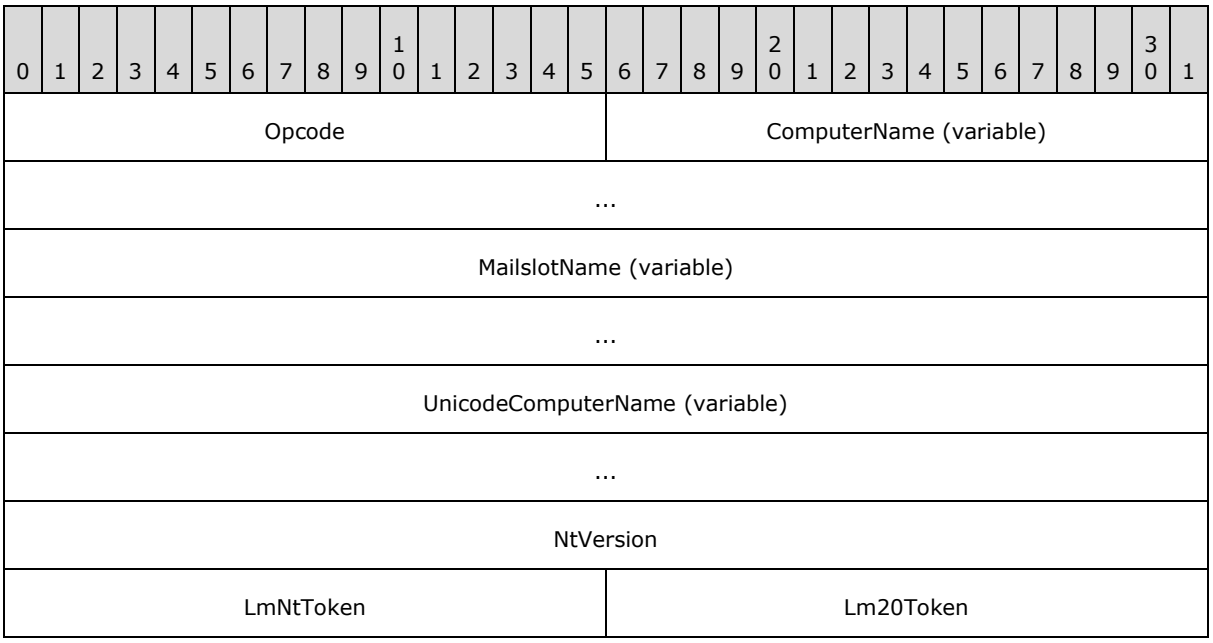
Operation code set in the request and response of an LDAP ping (section 6.3.3) or a mailslot ping (section 6.3.5).

Symbolic name	Value (Associated packet format)
LOGON_PRIMARY_QUERY	7 (section 6.3.1.4)
LOGON_PRIMARY_RESPONSE	12 (section 6.3.1.5)
LOGON_SAM_LOGON_REQUEST	18 (section 6.3.1.6)
LOGON_SAM_LOGON_RESPONSE	19 (section 6.3.1.8)
LOGON_SAM_PAUSE_RESPONSE	20 (section 6.3.1.8)

Symbolic name	Value (Associated packet format)
LOGON_SAM_USER_UNKNOWN	21 (section 6.3.1.8)
LOGON_SAM_LOGON_RESPONSE_EX	23 (section 6.3.1.9)
LOGON_SAM_PAUSE_RESPONSE_EX	24 (section 6.3.1.8)
LOGON_SAM_USER_UNKNOWN_EX	25 (section 6.3.1.8)

6.3.1.4 NETLOGON_LOGON_QUERY

The format of a mailslot ping as documented in section 6.3.5. This can be used if a PDC is required.



Opcode (2 bytes): Operation code (see section 6.3.1.3). Set to LOGON_PRIMARY_QUERY.

ComputerName (variable): Null-terminated ASCII value of the NETBIOS name of the client. This field SHOULD contain at least one character: the null terminator.

MailslotName (variable): Null-terminated ASCII value of the name of the mailslot on which the client listens. This field is always aligned to an even byte boundary, with padding (bytes of value 0) to the next even byte boundary as necessary.

UnicodeComputerName (variable): Null-terminated Unicode value of the NETBIOS name of the client. This field SHOULD contain at least one character: the null terminator. Each Unicode value is encoded as 2 bytes.

NtVersion (4 bytes): NETLOGON_NT_VERSION options (see 6.3.1.1).

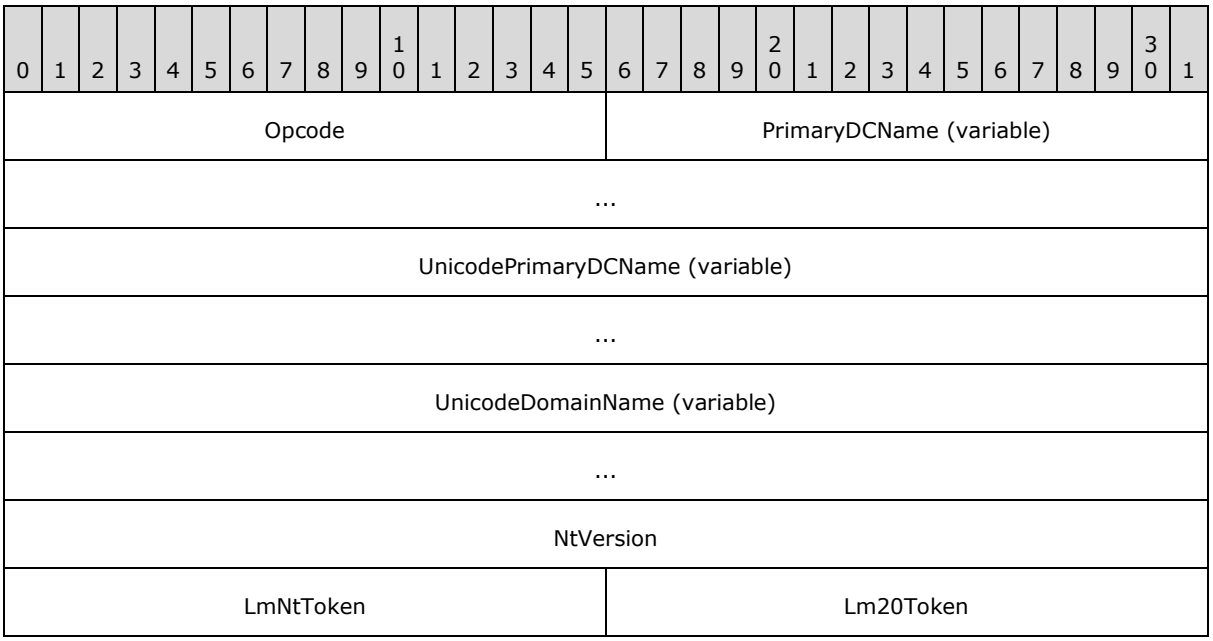
LmNtToken (2 bytes): This MUST be set to 0xFFFF.

Lm20Token (2 bytes): This MUST be set to 0xFFFF.

Note All multibyte quantities are represented in little-endian byte order.

6.3.1.5 NETLOGON_PRIMARY_RESPONSE

The NETLOGON_PRIMARY_RESPONSE structure is the PDC server's response to a mailslot ping (section 6.3.5).



Opcode (2 bytes): Operation code (see section 6.3.1.3). Set to LOGON_PRIMARY_RESPONSE.

PrimaryDCName (variable): Null-terminated ASCII value of the NetBIOS name of the server. This field SHOULD contain at least one character: the null terminator.

UnicodePrimaryDCName (variable): Null-terminated Unicode value of the NetBIOS name of the server. This field SHOULD contain at least one character: the null terminator. Each Unicode value is encoded as 2 bytes. This field is always aligned to an even byte boundary, with padding (bytes of value 0) to the next even byte boundary as necessary.

UnicodeDomainName (variable): Null-terminated Unicode value of the NetBIOS name of the NC. This field MUST contain at least one character: the null terminator. Each Unicode value is encoded as 2 bytes.

NtVersion (4 bytes): NETLOGON_NT_VERSION Options (see section 6.3.1.1).

LmNtToken (2 bytes): This MUST be set to 0xFFFF.

Lm20Token (2 bytes): This MUST be set to 0xFFFF.

Note All multibyte quantities are represented in little-endian byte order.

6.3.1.6 NETLOGON_SAM_LOGON_REQUEST

The format of a mailslot ping as documented in section 6.3.5.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Opcode																RequestCount															
UnicodeComputerName (variable)																															
...																															
UnicodeUserName (variable)																															
...																															
MailslotName (variable)																															
...																															
AllowableAccountControlBits																															
DomainSidSize																															
DomainSid (variable)																															
...																															
NtVersion																															
LmNtToken																Lm20Token															

Opcode (2 bytes): Operation code (see section 6.3.1.3). Set to LOGON_SAM_LOGON_REQUEST.

RequestCount (2 bytes): A **USHORT** that contains the number of times the user has repeated this request.

UnicodeComputerName (variable): Null-terminated Unicode value of the NETBIOS name of the client. This field MUST contain at least one character: the null terminator. Each Unicode value is encoded as 2 bytes.

UnicodeUserName (variable): Null-terminated Unicode value of the account name of the user being queried. This field MUST contain at least one character: the null terminator. Each Unicode value is encoded as 2 bytes.

MailslotName (variable): Null-terminated ASCII value of the name of the mailslot the client listens on.

AllowableAccountControlBits (4 bytes): Represents the userAccountControl attribute of an account.

DomainSidSize (4 bytes): A **DWORD** that contains the size of the DomainSid field.

DomainSid (variable): The SID of the domain, specified as a SID structure, which is defined in [MS-DTYP] section 2.4.2. Its length is defined in the **DomainSidSize** field. This field is padded as necessary so that it is aligned on a DWORD boundary.

NtVersion (4 bytes): NETLOGON_NT_VERSION Options (see 6.3.1.1).

LmNtToken (2 bytes): This MUST be set to 0xFFFF.

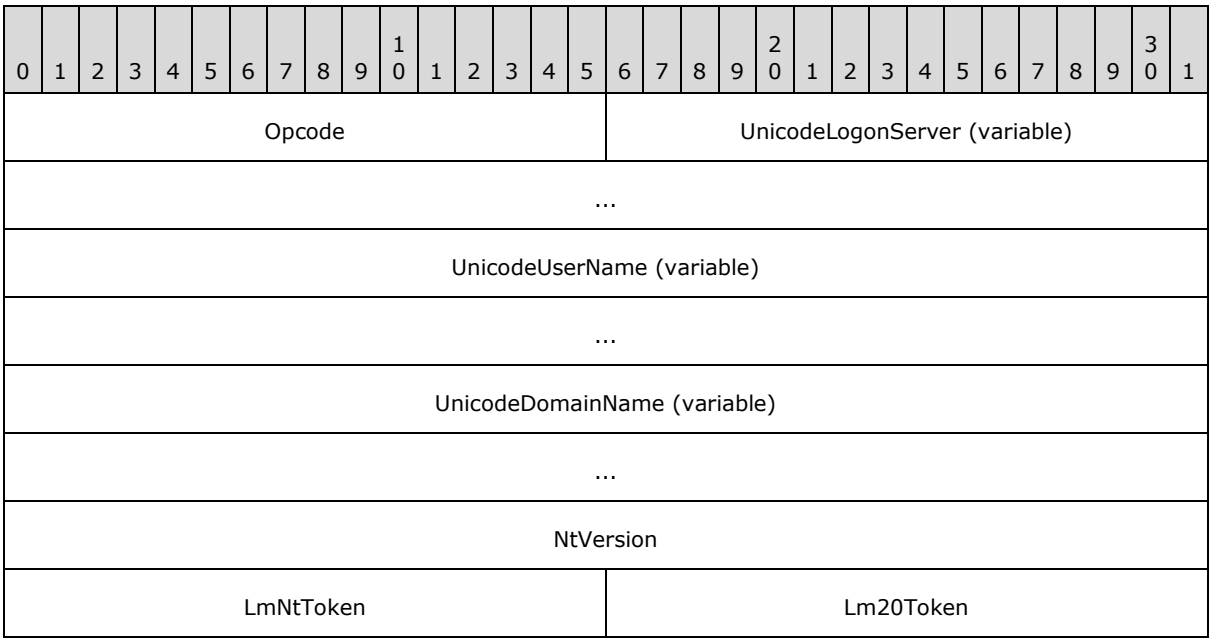
Lm20Token (2 bytes): This MUST be set to 0xFFFF.

Note Except as noted earlier in this section, there is no padding for alignment. Therefore, except as otherwise specified, all fields after **MailslotName** can occur on odd byte boundaries.

All multibyte quantities are represented in little-endian byte order.

6.3.1.7 NETLOGON_SAM_LOGON_RESPONSE_NT40

The NETLOGON_SAM_LOGON_RESPONSE_NT40 structure is the server's response to an LDAP ping (section 6.3.3) or a mailslot ping (section 6.3.5).



Opcode (2 bytes): Operation code (see section 6.3.1.3).

UnicodeLogonServer (variable): Null-terminated Unicode value of the NetBIOS name of the server. This field MUST contain at least one character: the null terminator. Each Unicode value is encoded as 2 bytes.

UnicodeUserName (variable): Null-terminated Unicode value of the name of the user copied directly from the client's request. This field MUST contain at least one character: the null terminator. Each Unicode value is encoded as 2 bytes.

UnicodeDomainName (variable): Null-terminated Unicode value of the NetBIOS name of the NC. This field MUST contain at least one character: the null terminator. Each Unicode value is encoded as 2 bytes.

NtVersion (4 bytes): Set to NETLOGON_NT_VERSION_1.

LmNtToken (2 bytes): This MUST be set to 0xFFFF.

Lm20Token (2 bytes): This MUST be set to 0xFFFF.

Note All multibyte quantities are represented in little-endian byte order.

6.3.1.8 NETLOGON_SAM_LOGON_RESPONSE

The NETLOGON_SAM_LOGON_RESPONSE structure is the first extended version of the server's response to an LDAP ping (section 6.3.3) or a mailslot ping (section 6.3.5).

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
Opcode																UnicodeLogonServer (variable)															
...																															
UnicodeUserName (variable)																															
...																															
UnicodeDomainName (variable)																															
...																															
DomainGuid																															
...																															
...																															
...																															
NullGuid																															
...																															
...																															
...																															
DnsForestName (variable)																															
...																															
DnsDomainName (variable)																															

...	
DnsHostName (variable)	
...	
DcIpAddress	
Flags	
NtVersion	
LmNtToken	Lm20Token

Opcode (2 bytes): Operation code (see section 6.3.1.3).

UnicodeLogonServer (variable): Null-terminated Unicode value of the NetBIOS name of the server. This field always contains at least one character: the null terminator. Each Unicode value is encoded as 2 bytes.

UnicodeUserName (variable): Null-terminated Unicode value of the name of the user copied directly from the client's request. This field always contains at least one character: the null terminator. Each Unicode value is encoded as 2 bytes.

UnicodeDomainName (variable): Null-terminated Unicode value of the NetBIOS name of the NC. This field always contains at least one character: the null terminator. Each Unicode value is encoded as 2 bytes.

DomainGuid (16 bytes): The value of the NC's **GUID** attribute specified as a GUID structure, which is defined in [MS-DTYP] section 2.3.4.

NullGuid (16 bytes): A NULL GUID. The GUID structure is defined in [MS-DTYP] section 2.3.4. Always set zero values for all fields in the GUID structure.

DnsForestName (variable): UTF-8 encoded value of the DNS forest name, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

DnsDomainName (variable): UTF-8 encoded value of the DNS NC name, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

DnsHostName (variable): UTF-8 encoded value of the DNS server name, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

DcIpAddress (4 bytes): The domain controller IP address, as specified in [RFC791].

Flags (4 bytes): DS_FLAG Options (see section 6.3.1.2).

NtVersion (4 bytes): Set to NETLOGON_NT_VERSION_1 | NETLOGON_NT_VERSION_5.

LmNtToken (2 bytes): This MUST be set to 0xFFFF.

Lm20Token (2 bytes): This MUST be set to 0xFFFF.

Note All multibyte quantities are represented in little-endian byte order.

6.3.1.9 NETLOGON_SAM_LOGON_RESPONSE_EX

The NETLOGON_SAM_LOGON_RESPONSE_EX structure is the second extended version of the server's response to an LDAP ping (section 6.3.3) or a mailslot ping (section 6.3.5).

ClientSiteName (variable)	
...	
DcSockAddrSize	DcSockAddr
...	
...	
...	
...	NextClosestSiteName (variable)
...	
NtVersion	
LmNtToken	Lm20Token

Opcode (2 bytes): Operation code (see section 6.3.1.3).

Sbz (2 bytes): This MUST be set to 0.

Flags (4 bytes): DS_FLAG Options (see section 6.3.1.2).

DomainGuid (16 bytes): The value of the NC's GUID attribute specified as a GUID structure, which is defined in [MS-DTYP] section 2.3.4.

DnsForestName (variable): UTF-8 encoded value of the DNS name of the forest, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

DnsDomainName (variable): UTF-8 encoded value of the DNS name of the NC, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

DnsHostName (variable): UTF-8 encoded value of the DNS name of the server, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

NetbiosDomainName (variable): UTF-8 encoded value of the NetBIOS name of the NC, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

NetbiosComputerName (variable): UTF-8 encoded value of the NetBIOS name of the server, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

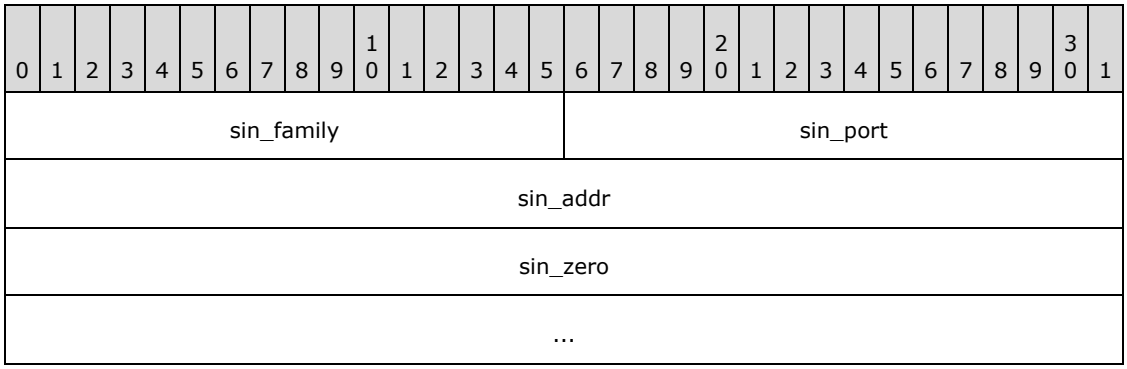
UserName (variable): UTF-8 encoded value of the user specified in the client's request, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

DcSiteName (variable): UTF-8 encoded value of the site name of the server, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

ClientSiteName (variable): UTF-8 encoded value of the site name of the client, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

DcSockAddrSize (1 byte): A **CHAR** that contains the size of the server's IP address. This field is included only if the client specifies NETLOGON_NT_VERSION_5EX_WITH_IP in the request.

DcSockAddr (16 bytes): The domain controller IPv4 address, structured as shown in the following diagram. This field is included only if the client specifies NETLOGON_NT_VERSION_5EX_WITH_IP in the request.



sin_family (2 bytes): The socket family, represented in little-endian byte order. The value SHOULD always be AF_INET (that is, 2).

sin_port (2 bytes): The socket port, represented in little-endian byte order. The value SHOULD always be zero.

sin_addr (4 bytes): The socket address, represented in big-endian byte order. The value is an IPv4 address. If the domain controller does not have an IPv4 address, this value SHOULD be 127.0.0.1.

sin_zero (8 bytes): Reserved. MUST be set to zero when sending and ignored on receipt.

NextClosestSiteName (variable): This field is included only if the client specifies NETLOGON_NT_VERSION_WITH_CLOSEST_SITE in the request, and if the responding DC has DC functional level DS_BEHAVIOR_WIN2008 or greater. When included, NextClosestSiteName contains the name of the site that is closest by cost to ClientSiteName without being equal to it. The site name is UTF-8 encoded, compressed as specified in [RFC1035] section 4.1.4. To get the decompressed string, see section 6.3.7.

NtVersion (4 bytes): NETLOGON_NT_VERSION_1 | NETLOGON_NT_VERSION_5EX.

LmNtToken (2 bytes): This MUST be set to 0xFFFF.

Lm20Token (2 bytes): This MUST be set to 0xFFFF.

Note All multibyte quantities are represented in little-endian byte order.

6.3.1.10 DNSRegistrationSettings

DNSRegistrationSettings is an abstract type containing the following fields.

PerformDynamicRegistration: A Boolean that indicates whether the DC registers DNS records on a periodic basis, as specified by [RFC2136]. Default value of this flag is true. If set to false, the DC does not itself register any DNS records.

AvoidDNSRecordsList: This is a list of zero or more of the following mnemonics. Presence of a specific mnemonic on the list is an instruction to the DC to skip the registration of the associated DNS record as part of dynamic DNS registration. By default this list is empty.

Mnemonic	DNS Record Type	Associated DNS Record
LdapIpAddress	A	<DnsDomainName>
Ldap	SRV	_ldap._tcp.<DnsDomainName>
LdapAtSite	SRV	_ldap._tcp.<SiteName>._sites.<DnsDomainName>
Pdc	SRV	_ldap._tcp.pdc._msdcs.<DnsDomainName>
Gc	SRV	_ldap._tcp.gc._msdcs.<DnsForestName>
GcAtSite	SRV	_ldap._tcp.<SiteName>._sites.gc._msdcs.<DnsForestName>
DcByGuid	SRV	_ldap._tcp.<DomainGuid>.domains._msdcs.<DnsForestName>
GcIpAddress	A	_gc._msdcs.<DnsForestName>
DsaCname	CNAME	<DsaGuid>._msdcs.<DnsForestName>
Kdc	SRV	_kerberos._tcp.dc._msdcs.<DnsDomainName>
KdcAtSite	SRV	_kerberos._tcp.dc._msdcs.<SiteName>._sites.<DnsDomainName>
Dc	SRV	_ldap._tcp.dc._msdcs.<DnsDomainName>
DcAtSite	SRV	_ldap._tcp.<SiteName>._sites.dc._msdcs.<DnsDomainName>
Rfc1510Kdc	SRV	_kerberos._tcp.<DnsDomainName>
Rfc1510KdcAtSite	SRV	_kerberos._tcp.<SiteName>._sites.<DnsDomainName>
GenericGc	SRV	_gc._tcp.<DnsForestName>
GenericGcAtSite	SRV	_gc._tcp.<SiteName>._sites.<DnsForestName>
Rfc1510UdpKdc	SRV	_kerberos._udp.<DnsDomainName>
Rfc1510Kpwd	SRV	_kpasswd._tcp.<DnsDomainName>
Rfc1510UdpKpwd	SRV	_kpasswd._udp.<DnsDomainName>

where

<DnsDomainName> = FQDN of the default NC of the DC

<DnsForestName> = FQDN of the forest root domain NC

<SiteName> = Site for which the record is being registered

<DsaGuid> = DSA GUID

<DomainGuid> = objectGuid of the root object of the default NC

DynamicRegistrationRefreshInterval: The time interval in minutes after which the DC re-registers DNS records if dc.dnsRegistrationSettings.PerformDynamicRegistration is true. The default value is 60.

SRVRecordWeight: Specifies the value of the **Weight** field for all DNS SRV records ([RFC2782]) that are registered by the DC. The default value is 100.

SRVRecordPriority: Specifies the value of the **Priority** field for all DNS SRV records ([RFC2782]) that are registered by the DC. The default value is 0.

DNSRecordTTL: Specifies the value of the **TTL** field for all DNS records ([RFC2782]) that are registered by the DC. The default value is 600 seconds.

PerformAutoSiteCoverage: A Boolean that indicates whether the DC registers records for any additional sites that do not have any DCs in them. Implementations can choose any algorithm to determine which DCs cover the sites that don't already have coverage. The choice of algorithm does not affect client interoperability.The default value of this flag is true.

SitesForDCRecordsList: A list of site names. This list instructs the DC to register the DNS records that are registered for the default NC (see section 6.3.2.3) for all the listed sites. By default this list is empty.

SitesForGCRecordsList: A list of site names. This list instructs the DC to register the DNS records that are registered for the GC server (see section 6.3.2.3) for all the listed sites. By default this list is empty.

SitesForNDNCRecordsList: A list of site names. This list instructs the DC to register the DNS records that are registered for an application NC (see section 6.3.2.3) for all the listed sites. By default this list is empty.

Each of the above fields can be configured by an implementation-dependent mechanism. On Windows Server operating system, these can also be configured at the following registry key path:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Netlogon\Parameters
```

If a value is present under this key, it takes precedence over any value set by an implementation-dependent configuration mechanism. The following table describes the name of the registry key value for each field, the registry type and the range for each setting:

Field	Registry Value Name	RegistryType	Range/Acceptable Values
PerformDynamicRegistration	UseDynamicDns	REG_DWORD	Enabled = 1, Disabled = 0
AvoidDNSRecordsList	DnsAvoidRegisterRecords	REG_SZ	List of space delimited mnemonics mentioned in the table of mnemonics above.
DynamicRegistrationRefreshInterval	DnsRefreshInterval	REG_DWORD	MinValue = 0, MaxValue =

Field	Registry Value Name	RegistryType	Range/Acceptable Values
			4294967200
SRVRecordWeight	LdapSrvWeight	REG_DWORD	MinValue = 0, MaxValue = 65535
SRVRecordPriority	LdapSrvPriority	REG_DWORD	MinValue = 0, MaxValue = 65535
DNSRecordTTL	DnsTtl	REG_DWORD	MinValue = 0, MaxValue = 2147483647
PerformAutoSiteCoverage	AutoSiteCoverage	REG_DWORD	Enabled = 1, Disabled = 0
SitesForDCRecordsList	SiteCoverage	REG_SZ	List of space delimited site names.
SitesForGCRecordsList	GcSiteCoverage	REG_SZ	List of space delimited site names.
SitesForNDNCRecordsList	NdncSiteCoverage	REG_SZ	List of space delimited site names.

6.3.2 DNS Record Registrations

If dc.dnsRegistrationSettings.PerformDynamicRegistration is true, a DC performs dynamic registration of DNS records (as specified by [RFC2136]) at a periodic interval (see section 6.3.2.1.1). Additionally, a DC performs the DNS record registration on demand when requested by the Netlogon Remote Protocol as described in [MS-NRPC] section 3.5.4.9.1.

6.3.2.1 Timers

6.3.2.1.1 Register DNS Records Timer

This timer controls how often a DC registers DNS records if configured to do so on a periodic basis. If dc.dnsRegistrationSettings.PerformDynamicRegistration is true, this timer is configured to signal an event every dc.dnsRegistrationSettings.DynamicRegistrationRefreshInterval minutes. At each timer event, the DC registers the DNS records described in SRV Records (section 6.3.2.3) and Non-SRV Records (section 6.3.2.4), unless explicitly excluded via dc.dnsRegistrationSettings.AvoidDNSRecordsList.

6.3.2.2 Non-Timer Events

There is one non-timer event, Force Register DNS Records, in the Active Directory System (beyond those non-timer events specified in the underlying protocol documents).

6.3.2.2.1 Force Register DNS Records Non-Timer Event

This event can be triggered by another system to cause the DC to register DNS records.

When this event occurs, the DC registers the DNS records described in SRV Records (section 6.3.2.3) and Non-SRV Records (section 6.3.2.4), unless explicitly excluded via dc.dnsRegistrationSettings.AvoidDNSRecordsList.

6.3.2.3 SRV Records

The SRV DNS Resource Record for specifying the location of services is specified in [RFC2782]. An **SRV record** maps the name of a service to the DNS name of a server that offers that service.

The creation of DNS Resource Records is specified in [RFC2136].

The name of an SRV Resource Record is in the following form:

- Service.Proto.Name TTL Class SRV Priority Weight Port Target

A client queries for these records by sending a DNS SRV query [RFC2782] to a DNS server.

Non-RODC server

If the DC is a non-RODC with default NC X (and NC X's GUID is G) in forest Z, then it registers SRV records with Service.Proto.Name equal to the following.

```
_ldap._tcp.X
_ldap._tcp.dc._msdcs.X
_ldap._tcp.G. domains._msdcs.Z
_kerberos._tcp.X
_kerberos._udp.X
_kerberos._tcp.dc._msdcs.X
_kpasswd._tcp.X
_kpasswd._udp.X
```

In addition, the DC registers site-specific records for the following sites:

The site that the DC is in (see sections 6.1.1.2.2.1 and 6.1.1.2.2.1.2).

The sites listed in dc.dnsRegistrationSettings.SitesForDCRecordsList.

If dc.dnsRegistrationSettings.PerformAutoSiteCoverage is true, the additional sites that should be covered by this DC as determined by the implementation's chosen algorithm.

For each site Y_i in the above list of sites, the DC registers SRV records with Service.Proto.Name equal to the following.

```
_ldap._tcp.Yi._sites.X
_ldap._tcp.Yi._sites.dc._msdcs.X
_kerberos._tcp.Yi._sites.X
_kerberos._tcp.Yi._sites.dc._msdcs.X
```

RODC Server

If the DC is an RODC with default NC X (and NC X's GUID is G) in site Y and in forest Z, then it registers SRV records with Service.Proto.Name equal to the following.

```
_ldap._tcp.Y._sites.X
_ldap._tcp.Y._sites.dc._msdcs.X
_kerberos._tcp.Y._sites.X
_kerberos._tcp.Y._sites.dc._msdcs.X
```

Non-RODC GC server

If the DC is also a non-RODC GC server, then it registers SRV records with Service.Proto.Name equal to the following.

```
_ldap._tcp.gc._msdcs.Z
_gc._tcp.Z
```

In addition, the DC registers site specific records for the following sites:

The site that the DC is in (see sections 6.1.1.2.2.1 and 6.1.1.2.2.1.2).

The sites listed in dc.dnsRegistrationSettings.SitesForGCRecordsList.

If dc.dnsRegistrationSettings.PerformAutoSiteCoverage is true, the additional sites that should be covered by this domain controller as determined by the implementation's chosen algorithm.

For each site Y_i in the above list of sites, the DC registers SRV records with Service.Proto.Name equal to the following:

```
_ldap._tcp.Yi._sites.gc._msdcs.Z
_gc._tcp.Yi._sites.Z
```

RODC GC server

If the DC is also an RODC GC server, then it registers SRV records with Service.Proto.Name equal to the following.

```
_ldap._tcp.Y._sites.gc._msdcs.Z
_gc._tcp.Y._sites.Z
```

PDC

If the DC is also holds the PDC Emulator FSMO role for its default NC, then it registers SRV records with Service.Proto.Name equal to the following.

```
_ldap._tcp.pdc._msdcs.X
```

Application NC host

If the DC also hosts application NCs, then for each application NC A_i, it registers SRV records with Service.Proto.Name equal to the following.

```
_ldap._tcp.Ai
```

In addition, the DC also registers site-specific records for the following sites:

The site that the DC is in (see sections 6.1.1.2.2.1 and 6.1.1.2.2.1.2).

The sites listed in dc.dnsRegistrationSettings.SitesForNDNCRecordsList.

If dc.dnsRegistrationSettings.PerformAutoSiteCoverage is true, the additional sites that should be covered by this domain controller as determined by the implementation's chosen algorithm.

For each application NC A_i and each site Y_i in the above list of sites, the DC registers SRV records with Service.Proto.Name equal to the following:

```
_ldap._tcp.Yi._sites.Ai
```

Example: If a DC with default NC:

```
X = na.fabrikam.com
```

is in site:

```
Y = site1
```

and forest:

```
Z = fabrikam.com
```

and NC X's GUID is:

```
G = 52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8
```

then its record of type _ldap._tcp.Y._sites.dc._msdcs.X has:

```
Service.Proto.Name =
    _ldap._tcp.site1._sites.dc._msdcs.na.fabrikam.com
```

and its record of type _ldap._tcp.G.domains._msdcs.Z has:

```
Service.Proto.Name =
```

```
_ldap._tcp.52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8.domains._msdcs.fabrikam.com
```

The following table describes the other fields of each SRV record registered by a server.

Field	Value
TTL	Set to dc.dnsRegistrationSettings.DNSRecordTTL.
Class	Set to IN.
SRV	Set to SRV.
Priority	Set to dc.dnsRegistrationSettings.SRVRecordPriority.
Weight	Set to dc.dnsRegistrationSettings.SRVRecordWeight.
Port	Set to 389 for LDAP service. Set to 3268 for GC service. Set to 88 for Kerberos KDC service. Set to 464 for Kerberos Password Change service.
Target	Set to the fully qualified DNS name of the server.

6.3.2.4 Non-SRV Records

In addition to SRV records, a DC also registers CNAME [RFC1034] and type A [RFC1034] DNS records.

A CNAME record acts as an alias for a DNS hostname and has the following form:

- Name TTL class type RDATA

A client queries for these records by sending a DNS A, CNAME, or * query [RFC1034] to a DNS server.

If a server is a DC in forest Z, and its **DSA GUID** is G, then the server registers a CNAME record with Name field set to G._msdcs.Z. This name is called the DC's GUID-based DNS name.

Example: If a DC is in forest:

```
Z = fabrikam.com
```

and its DSA GUID is:

```
G = 52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8
```

then it registers a CNAME record with:

```
Name =
    52f6c43b-99ec-4040-a2b0-e9ebf2ec02b8._msdcs.fabrikam.com
```

The following table describes the other fields of each CNAME record registered by a server.

Field	Value
TTL	Set to dc.dnsRegistrationSettings.DNSRecordTTL.
Class	Set to IN.
Type	Set to CNAME.
RDATA	Set to the fully qualified DNS name of the server.

A type A record associates an IP address with a name and takes the form:

- Name TTL class type RDATA

A client queries for these records by sending a DNS A or * query [RFC1034] to a DNS server.

If a server is a DC with default NC X in forest Z, then it publishes a type A record with **Name** field X. If the DC is a GC server, it also publishes a type A record with **Name** field gc._msdcs.Z.

Example: If a DC has default NC:

```
X = na.fabrikam.com
```

and is in forest:

```
Z = fabrikam.com
```

then it registers a type A record with:

```
Name = na.fabrikam.com
```

If the DC is a GC server, it registers a type A record with:

```
Name = gc._msdcs.fabrikam.com
```

The following table describes the other fields of each type A record registered by a server.

Field	Value
TTL	Set to dc.dnsRegistrationSettings.DNSRecordTTL.
Class	Set to IN.
Type	Set to A.
RDATA	Set to the IP address of the server used for DC functions.

6.3.3 LDAP Ping

This topic describes the usage of LDAP to verify the aliveness of the domain controller and also check whether the domain controller matches a specific set of requirements. This operation is commonly referred to as LDAP ping.

An LDAP rootDSE search (section 3.1.1.3.2) that retrieves the rootDSE attribute netlogon (section 3.1.1.3.2.14) triggers the following processing on the server: Syntactic validation of the filter as specified in section 6.3.3.1 and construction of a DC response to the search request as specified in sections 6.3.3.2 and 6.3.3.3.

The LDAP search filter included in the SearchRequest is a one-level AND of equalityMatch tests of the following elements:

DnsDomain: The DNS name of an NC (default NC or application NC).

Host: The NetBIOS name of the client.

DnsHostName: The fully qualified domain name (FQDN) of the client.

> **Note** The DnsHostName element is not sent by Windows clients from Windows 2000 operating system through Windows 7 operating system and Windows Server 2008 R2 operating system.

User: The sAMAccountName of an account in the domain specified by DnsDomain, DomainSid, or DomainGuid.

AAC: Represents the userAccountControl attribute of an account.

DomainSid: The SID of a domain.

DomainGuid: The GUID of a domain.

NtVer: NETLOGON_NT_VERSION Options (see section 6.3.1.1).

Example:

(&(DnsDomain=abcde.corp.microsoft.com)(Host=abcdefgh-dev)(User=abcdefgh-dev$)(AAC=\80\00\00\00)(DomainGuid=\3b\b0\21\ca\d3\6d\d1\11\8a\7d\b8\df\b1\56\87\1f)(NtVer=\06\00\00\00))

Network payload:

```
A0 84 00 00 00 A8 A3 84 00 00 00 25 04 09 44    ?...¨£?...%..D
6E 73 44 6F 6D 61 69 66 04 18 61 62 63 64 65    nsDomain..abcde
2E 63 6F 72 70 2E 6D 69 63 72 6F 73 6F 66 74    .corp.microsoft
2E 63 6F 6D A3 84 00 00 00 14 04 04 48 6F 73    .com£?......Hos
74 04 0C 61 62 63 64 65 66 67 68 2D 64 65 76    t..abcdefgh-dev
A3 84 00 00 00 15 04 04 55 73 65 72 04 0D 61    £?......User..a
62 63 64 65 66 67 68 2D 64 65 76 24 A3 84 00    bcdefgh-dev$£?.
00 00 0B 04 03 41 41 43 04 04 80 00 00 00 A3    .....AAC..?...£?
84 00 00 00 1E 04 0A 44 6F 6D 61 69 6E 47 75    .......DomainGu
69 64 04 10 3B B0 21 CA D3 6D D1 11 8A 7D B8    id..;°!ÊÓmÑ.?}¸
DF B1 56 87 1F A3 84 00 00 00 0D 04 05 4E 74    ß±V?.£?......Nt
56 65 72 04 04 06 00 00 00 30 84 00 00 00 0A    Ver......0?....
04 08 6E 65 74 6C 6F 67 6F 6E                   ..netlogon
```

6.3.3.1 Syntactic Validation of the Filter

If any of the elements is specified more than once, then the filter is invalid.

If the value of the string passed with DomainGuid has a different size than the size of GUID ([MS-DTYP] section 2.3.4), then the filter is invalid.

If the numeric value of the string passed with AAC is longer than the largest unsigned integer that can be represented in a **DWORD** or has an unsupported bit set, then the filter is invalid.

If the numeric value of the string passed with NtVer is longer than the largest unsigned integer that can be represented in a **DWORD** or has an unsupported bit set, then the filter is invalid.

The response of the DC for the invalid filter case is documented in section 6.3.3.3.

The DC MUST ignore any unrecognized filter elements.

6.3.3.2 Domain Controller Response to an LDAP Ping

Let reqGuidNC be set as follows:

- If the filter does not include the (DomainGuid=domainGuid) clause, reqGuidNC is set to NULL.
- If the filter includes the (DomainGuid=domainGuid) clause:
  - If domainGuid is not a valid GUID, the response of the DC is documented in section 6.3.3.3.
  - If there is no NC hosted by the server whose GUID is domainGuid, the response of the DC is documented in section 6.3.3.3.
  - Otherwise, reqGuidNC is set to the NC hosted by the server whose GUID is domainGuid.

Let reqDnsNC be set as follows:

- If the filter does not include the (DnsDomain=dnsDomain) clause:
  - If reqGuidNC is NULL, reqDnsNC is set to the default NC hosted by the server.
  - If reqGuidNC is not NULL, reqDnsNC is set to NULL.
- If the filter includes the (DnsDomain=dnsDomain) clause:
  - If dnsDomain is empty, the response of the DC is documented in section 6.3.3.3.
  - If there is no NC hosted by the server whose DNS name is dnsDomain, the response of the DC is documented in section 6.3.3.3.
  - Otherwise, reqDnsNC is set to the NC hosted by the server whose DNS name is dnsDomain.

Let reqNCUsed be set as follows:

- If reqGuidNC is NULL, then reqNCUsed is set to reqDnsNC.
- If reqDnsNC is NULL, then reqNCUsed is set to reqGuidNC.
- If neither reqGuidNC nor reqDnsNC are NULL, then reqNCUsed is set to either reqGuidNC or reqDnsNC. The protocol does not specify which value is used, nor that a DC is consistent in which value is used.

Let reqSidNC be set as follows:

- If the filter does not include the (DomainSid=domainSid) clause, reqSidNC is set to NULL.
- If the filter includes the (DomainSid=domainSid) clause:
  - If domainSid is not a valid sid, the response of the DC is documented in section 6.3.3.3.
  - If there is no NC hosted by the server whose Sid is domainSid, the response of the DC is documented in section 6.3.3.3.
  - If domainSid is not equal to the SID of NC reqNCUsed, the response of the DC is documented in section 6.3.3.3.
  - Otherwise, reqSidNC is set to the NC hosted by the server whose SID is domainSid.

Let u be set as follows:

- If the filter does not include the (User=user) clause, then u is set to NULL.
- If filter includes the (User=user) clause, then u is set to the supplied value.

Let x be as follows:

- Let y be an object in NC reqNCUsed where y!sAMAccountName = u.
  - If there is no such object y, then x is set to NULL.
  - If there is an object y, x is set as:
    - Let aac be set as follows:
      - If the filter does not include the (AAC=aac) clause, then aac is set to 0.
      - If filter includes the (AAC = aac) clause, then aac is set to the supplied value.
    - Let uac be set to y!userAccountControl.
      - If uac has the USER_ACCOUNT_DISABLED ([MS-SAMR] section 2.2.1.12) bit set, then let x be equal to NULL.
      - If (aac & uac & USER_TEMP_DUPLICATE_ACCOUNT | USER_NORMAL_ACCOUNT | USER_INTERDOMAIN_TRUST_ACCOUNT | USER_WORKSTATION_TRUST_ACCOUNT | USER_SERVER_TRUST_ACCOUNT [MS-SAMR] section 2.2.1.12) is zero, then let x be equal to NULL. The effect of doing this is so that the server only checks USER_TEMP_DUPLICATE_ACCOUNT | USER_NORMAL_ACCOUNT | USER_INTERDOMAIN_TRUST_ACCOUNT | USER_WORKSTATION_TRUST_ACCOUNT | USER_SERVER_TRUST_ACCOUNT bits.
      - Otherwise, set x to y.

Let s be set as follows:

- Let sno be a subnet object in the Subnets Container (section 6.1.1.2.2.2) where sno!name represents the range of IP addresses, which includes the client's IP address (see section 6.1.1.2.2.2.1).
  - If there is no such object sno, then s is set to NULL.

- If there is an object sno, s is set as follows:
  - If sno!siteObject has a value, let so be the site object referred to by this attribute value (see section 6.1.1.2.2.2.1). Set s to so!name.
  - If sno!siteObject does not contain a value, set s to NULL.

Note In Windows, the server computes the client's IP address from the client's socket address. If the NtVer filter element has the NETLOGON_NT_VERSION_5EX or NETLOGON_NT_VERSION_5EX_WITH_IP bit set, and if the client's site cannot be computed from the client's socket address, then the server computes the client's IP address by using either the FQDN of the client, which is found in the DnsHostName filter element (if present), or the NetBIOS name of the client, which is found in the Host filter element (section 6.3.3). The server then uses the IP address to determine the site.

Let v be the NtVer requested by the client in the search filter.

- If the server is configured to respond to ping requests in the form of a NETLOGON_SAM_LOGON_RESPONSE_NT40 structure, and v does not have the NETLOGON_NT_VERSION_AVOID_NT4EMUL bit set (for an informative example of how and why this is configured in the Windows implementation, see [MSKB-298713]), the server uses the NETLOGON_SAM_LOGON_RESPONSE_NT40 structure to send the response.
- Else, if v has the NETLOGON_NT_VERSION_5EX or NETLOGON_NT_VERSION_5EX_WITH_IP bit set, the server uses the NETLOGON_SAM_LOGON_RESPONSE_EX structure to send the response.
- Else, if v has the NETLOGON_NT_VERSION_5 bit set, the server uses the NETLOGON_SAM_LOGON_RESPONSE structure to send the response.
- For all other cases, the server uses the NETLOGON_SAM_LOGON_RESPONSE_NT40 structure to send the response.

Let t be set as follows:

- When the NetLogon service is in a paused state, if v does not have the NETLOGON_NT_VERSION_PDC bit set or the server is not a PDC, let t be 1.
- If the value of rootDSE attribute isSynchronized (see section 3.1.1.3) is false, let t be 1.
- When the NetLogon RPC server is not initialized, if v does not have the NETLOGON_NT_VERSION_LOCAL bit set, let t be 1.
- If the FRS service is in a paused state, let t be 1.
- Otherwise, let t be 0.

After the preceding processing has occurred, if the server has not responded to an invalid filter (as documented in section 6.3.3.3), the server returns an LDAP SearchResultEntry to the client with the following form:

- The ObjectName of the SearchResultEntry is NULL and the attribute list contains one attribute. This attribute is named "Netlogon" and its value is a little-endian octet string packed in NETLOGON_SAM_LOGON_RESPONSE_EX, NETLOGON_SAM_LOGON_RESPONSE, or NETLOGON_SAM_LOGON_RESPONSE_NT40, depending on value v.
  - If the server uses NETLOGON_SAM_LOGON_RESPONSE_EX to pack the value, it does the following:

OperationCode: Set to LOGON_SAM_PAUSE_RESPONSE_EX if t is equal to 1. Set to LOGON_SAM_USER_UNKNOWN_EX if u is not NULL, but x is NULL. Set to LOGON_SAM_LOGON_RESPONSE_EX in other cases.

Flags:

Bit values are taken from DS_FLAGS in section 6.3.1.2.

- If the server holds the PDC FSMO role (see section 3.1.1.1.11), the DS_PDC_FLAG bit is set.
- If the server is a global catalog server, the DS_GC_FLAG bit is set. This bit is set if and only if the isGlobalCatalogReady attribute on the rootDSE is true (see section 3.1.1.3.2.10).
- If the server is a KDC, the DS_KDC_FLAG bit is set.
- If the server is running the Win32 Time Service, as specified in [MS-W32T] and indicated by bit field A in the ServiceBits flag in the NetLogon Remote Protocol ([MS-NRPC] section 3.5.1), the DS_TIMESERV_FLAG bit is set.
- If the server is in the same site as the client, the DS_CLOSEST_FLAG bit is set.
- If the server is not an RODC, the DS_WRITABLE_FLAG bit is set. [MS-DRSR] section 5.7, AmIRODC, explains how to determine if a DC is an RODC.
- If the server is configured to be a reliable time source (the way in which the configuration can be done is outside the scope of the state model and is implementation-dependent) as indicated by bit field B in the ServiceBits flag in the NetLogon Remote Protocol ([MS-NRPC] section 3.5.1), the DS_GOOD_TIMESERV_FLAG bit is set.
- If the DnsDomain value specified in the search filter is an application NC, the DS_NDNC_FLAG bit is set.
- If the server is an RODC, the DS_SELECT_SECRET_DOMAIN_6_FLAG bit is set.
- If the server is a writable DC and not running Windows 2000 Server operating system or Windows Server 2003 operating system, the DS_FULL_SECRET_DOMAIN_6_FLAG bit is set.
- If the server is running the Active Directory Web Service, as specified in [MS-ADDM] and indicated by the bit field C in the ServiceBits flag in the Netlogon Remote Protocol ([MS-NRPC] section 3.5.1), the DS_WS_FLAG bit is set.
- If the server is running Windows Server 2012 operating system or Windows Server 2012 R2 operating system, the DS_DS_8_FLAG bit is set.
- If the server is running Windows Server 2012 R2, the DS_DS_9_FLAG bit is set.
- Always set the DS_LDAP_FLAG and DS_DS_FLAG bits.
- All the other bits of DS_FLAG are set to 0.

DomainGuid: Set to the GUID of NC reqNCUsed.

DnsForestName: Set to the DNS name of the forest.

DnsDomainName: Set to the DNS name of the NC reqNCUsed.

DnsHostName: Set to the DNS name of the server.

NetbiosDomainName: Set to the NetBIOS name of the NC reqNCUsed.

NetbiosComputerName: Set to the NetBIOS name of the server.

UserName: Set to u.

DcSiteName: Set to the site name of the server.

ClientSiteName: Set to the site s.

DcSockAddrSize: Set to the size of the server's IP address.

SockAddr: Set to the IP address of the server.

NextClosestSiteName: If v has NETLOGON_NT_VERSION_WITH_CLOSEST_SITE and the DC has DC functional level DS_BEHAVIOR_WIN2008 or greater, use IDL_DRSQuerySitesByCost ([MS-DRSR] section 4.1.16) to find the site C that is closest to ClientSiteName but not equal to ClientSiteName, and set this field to C. Otherwise omit this field.

NtVersion: If the NextClosestSiteName field is set, set this field to {NETLOGON_NT_VERSION_1, NETLOGON_NT_VERSION_WITH_CLOSEST_SITE, NETLOGON_NT_VERSION_5EX}; otherwise set this field to {NETLOGON_NT_VERSION_1, NETLOGON_NT_VERSION_5EX}.

LmNtToken: Always set to 0xFFFF.

Lm20Token: Always set to 0xFFFF.

- If the server uses NETLOGON_SAM_LOGON_RESPONSE to pack the value, it does the following:

  OperationCode: Set to LOGON_SAM_PAUSE_RESPONSE if t is equal to 1. Set to LOGON_SAM_USER_UNKNOWN if u is not NULL, but x is NULL. Set to LOGON_SAM_LOGON_RESPONSE in other cases.

  UnicodeLogonServer: Set to the NetBIOS name of the server.

  UnicodeUserName: Set to u.

  UnicodeDomainName: Set to the NetBIOS name of the domain.

  DomainGuid: Set to the GUID of the domain.

  SiteGuid: Always set to NULL GUID.

  DnsForestName: Set to the DNS name of the forest.

  DnsDomainName: Set to the DNS name of the domain.

  DnsHostName: Set to the DNS name of the server.

  DcIpAddress: Set to the IP address of the server.

  Flags: If the server is a PDC, bit DS_PDC_FLAG is set; bit DS_DS_FLAG is always set; all the other bits of DS_FLAG are set to 0.

  NtVersion: Set to NETLOGON_NT_VERSION_1 | NETLOGON_NT_VERSION_5.

  LmNtToken: Always set to 0xFFFF.

Lm20Token: Always set to 0xFFFF.

- If the server uses NETLOGON_SAM_LOGON_RESPONSE_NT40 to pack the value, it does the following:

  OperationCode: If t is 1, set to LOGON_SAM_PAUSE_RESPONSE. Else, if u is not NULL, but x is NULL, set to LOGON_SAM_USER_UNKNOWN. If none of the preceding conditions are met, set to LOGON_SAM_LOGON_RESPONSE.

  UnicodeLogonServer: Set to the NetBIOS name of the server.

  UnicodeUserName: Set to u.

  UnicodeDomainName: Set to the NetBIOS name of the domain.

  NtVersion: Set to NETLOGON_NT_VERSION_1.

  LmNtToken: Always set to 0xFFFF.

  Lm20Token: Always set to 0xFFFF.

LdapResult of SearchResultDone entry is set to 0 (*success*).

6.3.3.3 Response to Invalid Filter

If the filter is not syntactically valid for any of the cases specified in the preceding sections, the server returns an LDAP SearchResultEntry with the following form:

The ObjectName of the SearchResultEntry is NULL. Attribute of SearchResultEntry is NULL. And LdapResult of SearchResultDone entry is set to 0 (*success*).

6.3.4 NetBIOS Broadcast and NBNS Background

If a server is in a domain whose NetBIOS name is d, it registers <d>[1C] records, and <d>[1B] records if it is a PDC, to the NBNS(WINS) server. A client can retrieve those records by either broadcasting or querying against NBNS(WINS) directly.

For more information, see [RFC1001] and [RFC1002].

6.3.5 Mailslot Ping

This section describes the usage of mailslot messages to verify the aliveness of the DC and also to check whether that DC matches a specific set of requirements. This operation is commonly referred to as a mailslot ping.

The server creates a mailslot (as specified in [MS-MAIL] section 3.2.4.1) with the name \\mailslot\net\netlogon and listens to this mailslot [MS-MAIL] section 3.2.4.2. If the opcode of the mailslot message (hereafter in this section referred to simply as "message") is set to LOGON_PRIMARY_QUERY, it interprets the message as a NETLOGON_LOGON_QUERY structure; otherwise, it interprets the message as a NETLOGON_SAM_LOGON_REQUEST.

The server then completes the following processing:

If the opcode is set to LOGON_PRIMARY_QUERY and the server is not the PDC, the DC ignores the message without sending a response back to the client. If the opcode is set to LOGON_SAM_LOGON_REQUEST and NtVer is not NETLOGON_NT_VERSION_5, the DC ignores the message without sending a response back to the client. The server determines whether or not it is

the PDC by calling the IsEffectiveRoleOwner(roleObject(Default NC, PdcEmulationMasterRole)) function. If the function returns true, the server is the PDC, otherwise it is not. See section 3.1.1.5.1.8 for more information.

If **DomainSidSize** is not zero, it checks whether the default NC has the same SID; if it does not, the server ignores the message without sending a response back to the client.

If **UnicodeUserName** is specified, it is processed in the same way as the User value in section 6.3.3.2.

Let v be the NtVer requested by the client.

- If dc.nt4EmulatorEnabled is TRUE, and v does not have the NETLOGON_NT_VERSION_AVOID_NT4EMUL bit set, the server uses the NETLOGON_SAM_LOGON_RESPONSE_NT40 structure to send the response.
- Else, if v has the NETLOGON_NT_VERSION_5EX or NETLOGON_NT_VERSION_5EX_WITH_IP bit set, the server uses the NETLOGON_SAM_LOGON_RESPONSE_EX structure to send the response.
- Else, if v has the NETLOGON_NT_VERSION_5 bit set, the server uses the NETLOGON_SAM_LOGON_RESPONSE structure to send the response.
- Else, if v has the NETLOGON_NT_VERSION_PDC bit set, the server uses the NETLOGON_PRIMARY_RESPONSE structure to send the response.
- For all other cases, the server uses the NETLOGON_SAM_LOGON_RESPONSE_NT40 structure to send the response.

Let t be 0.

- When the NetLogon service is in a paused state, if v does not have the NETLOGON_NT_VERSION_PDC bit set or server is not a PDC, let t be 1.
- If the value of rootDSE attributes isSynchronized (see section 3.1.1.3) is false, let t be 1.
- When the NetLogon RPC server is not initialized, if v does not have the NETLOGON_NT_VERSION_LOCAL bit set, let t be 1.
- If the FRS is in a paused state, let t be 1.

Then, the server sends a response back to the mailslot named in the client's request. The response message is packed in the NETLOGON_SAM_LOGON_RESPONSE structure, the NETLOGON_PRIMARY_RESPONSE structure, or the NETLOGON_SAM_LOGON_RESPONSE_NT40 structure, depending on the value of v.

- If the server uses NETLOGON_SAM_LOGON_RESPONSE to pack the value, it does the following:

  OperationCode: Set to LOGON_SAM_PAUSE_RESPONSE if t is equal to 1. Set to LOGON_SAM_USER_UNKNOWN if UnicodeUserName is not NULL, but x is NULL. Set to LOGON_SAM_LOGON_RESPONSE in other cases.

  UnicodeLogonServer: Set to the NetBIOS name of the server.

  UnicodeUserName: Set to UnicodeUserName filed in the request NETLOGON_SAM_LOGON_REQUEST message.

  UnicodeDomainName: Set to the NetBIOS name of the domain.

DomainGuid: Set to the GUID of the domain.

SiteGuid: Always set to NULL GUID.

DnsForestName: Set to the DNS name of the forest.

DnsDomainName: Set to the DNS name of the domain.

DnsHostName: Set to the DNS name of the server.

DcIpAddress: Set to the IP address of the server.

Flags: If the server is a PDC, bit DS_PDC_FLAG is set; bit DS_DS_FLAG is always set; all the other bits of DS_FLAG are set to 0.

NtVersion: Set to NETLOGON_NT_VERSION_1 | NETLOGON_NT_VERSION_5.

LmNtToken: Always set to 0xFFFF.

Lm20Token: Always set to 0xFFFF.

- If the server uses NETLOGON_SAM_LOGON_RESPONSE_NT40 to pack the value, it does the following:

  OperationCode: If t is 1, set to LOGON_SAM_PAUSE_RESPONSE. Else, if UnicodeUserName is not NULL, but x is NULL, set to LOGON_SAM_USER_UNKNOWN. If none of the preceding conditions are met, set to LOGON_SAM_LOGON_RESPONSE.

  UnicodeLogonServer: Set to the NetBIOS name of the server.

  UnicodeUserName: Set to UnicodeUserName filed in the request NETLOGON_SAM_LOGON_REQUEST message.

  UnicodeDomainName: Set to the NetBIOS name of the domain.

  NtVersion: Set to NETLOGON_NT_VERSION_1.

  LmNtToken: Always set to 0xFFFF.

  Lm20Token: Always set to 0xFFFF.

- If the server uses NETLOGON_PRIMARY_RESPONSE to pack the value, it does the following:

  OperationCode: If t is 1, set to LOGON_SAM_PAUSE_RESPONSE. Else, if UnicodeUserName is not NULL, but x is NULL, set to LOGON_SAM_USER_UNKNOWN. If none of the preceding conditions are met, set to LOGON_PRIMARY_RESPONSE.

  PrimaryDCName: Set to the ASCII value of the NetBIOS name of the server.

  UnicodePrimaryDCName: Set to the Unicode value of the NetBIOS name of the server.

  UnicodeDomainName: Set to the NetBIOS name of the domain.

  NtVersion: Set to NETLOGON_NT_VERSION_1.

  LmNtToken: Always set to 0xFFFF.

  Lm20Token: Always set to 0xFFFF.

6.3.6 Locating a Domain Controller

There are two ways to locate a domain controller: DNS-based discovery and NetBIOS-based discovery.

6.3.6.1 DNS-Based Discovery

For DNS-based discovery, the client machine can issue the following DNS queries:

- To locate an LDAP server hosting NC N, the client machine issues a DNS query for the SRV record _ldap._tcp.N, constructed from the NC name (N).
- To locate an LDAP server hosting NC N in site Y, the client machine issues a DNS query for the SRV record _ldap._tcp.Y._sites.N, constructed from the NC name (N) and the site name (Y).
- To locate domain controller (DC) hosting NC N, the client machine issues a DNS query for the SRV record _ldap._tcp.dc._msdcs.N, constructed from the NC name (N).
- To locate a DC hosting NC N in site Y, the client machine issues a DNS query for the SRV record _ldap._tcp.Y._sites.dc._msdcs.N, constructed from the NC name (N) and the site name (Y).
- To locate a DC hosting default NC X whose GUID is G in forest Z, the client machine issues a DNS query for the SRV record _ldap._tcp.G.domains._msdcs.Z, constructed from the default NC's GUID (G) and the forest name (Z).
- To locate a DC that is hosting default NC X and that is also a PDC, the client machine issues a DNS query for the SRV record _ldap._tcp.pdc._msdcs.X, constructed from the NC name (X).
- To locate a DC in forest Z that is a GC server, the client machine issues a DNS query for the SRV record _gc._tcp.Z, constructed from the forest name (Z).
- To locate DC in forest Z, site Y that is a GC server, the client machine issues a DNS query for the SRV record _gc._tcp.Y._sites.Z, constructed from the forest name (Z) and the site name (Y).
- To locate a server that is running the Kerberos Key Distribution Center service over TCP for default NC X, the client machine issues a DNS query for the SRV record _kerberos._tcp.X, constructed from the default NC name (X).
- To locate a server that is running the Kerberos Key Distribution Center service over UDP for default NC X, the client machine issues a DNS query for the SRV record _kerberos._udp.X, constructed from the default NC name (X).
- To locate a server in site Y that is running the Kerberos Key Distribution Center service over TCP for default NC X, the client machine issues a DNS query for the SRV record _kerberos._tcp.Y._sites.X, constructed from the default NC name (X) and the site name (Y).
- To locate a DC that is running the Kerberos Key Distribution Center service over TCP and that also hosts default NC X, the client machine issues a DNS query for the SRV record _kerberos.tcp.dc._msdcs.X, constructed from the default NC name (X).
- To locate a DC in site Y that is running the Kerberos Key Distribution Center service over TCP and that also hosts default NC X, the client machine issues a DNS query for the SRV record _kerberos.tcp.Y._sites.dc._msdcs.X, constructed from the default NC name (X) and the site name (Y).

- To locate a server that is running the Kerberos Password Change service over TCP for default NC X, the client machine issues a DNS query for the SRV record _kpasswd._tcp.X, constructed from the default NC name (X).
- To locate a server that is running the Kerberos Password Change service over UDP for default NC X, the client machine issues a DNS query for the SRV record _kpasswd._udp.X, constructed from the default NC name (X).

The DNS query returns a list of SRV records that match this query. The target field of the SRV record contains the FQDN of the server.

Upon receiving the DNS query results, the client machine retrieves the IP addresses corresponding to each server (via DNS A/AAAA queries) and sends an LDAP ping to the retrieved addresses in weighted random order [RFC2782]. If a server has multiple IP addresses, the client pings all of them before pinging the next server in the weighted random order. The client attempts the intended protocol request to the first server address that responds to the ping.

6.3.6.2 NetBIOS-Based Discovery

To locate a domain controller using NetBIOS-based discovery, the client either queries a Windows Internet Name Service (WINS) server or performs broadcasting. To find a domain controller in domain fabrikam, the client either sends a NetBIOS name query for <fabrikam>[1C] to the WINS server or broadcasts for <fabrikam>[1C] record. And if the client wants to find a primary domain controller, it issues a name query for <fabrikam>[1B] to the WINS server or broadcasts for <fabrikam>[1B] record.

Upon receiving the list of matching records from WINS or broadcasting, the client either contacts servers (attempts the intended protocol request) or sends a mailslot ping (section 6.3.5) to servers first, and then attempts the intended protocol request to a server that responded to the ping.

6.3.7 Name Compression and Decompression

The server can choose any compression algorithm, as long as the compressed stream can be decompressed using the following name decompression algorithm. When the server compresses the names for the LDAP ping response, if compression fails, the response of the server is documented in "Response to Invalid filter" (section 6.3.3.3). When the server compresses the names for the mailslot ping response, if compression fails, the server does not send any response back to the client.

Name Decompression Algorithm

```
--
--  On Entry: InputBuffer - a buffer of compressed data, treated as
--                bytes
--            InputBufferSize - The number of bytes in the InputBuffer
--            StringCount - number of strings needed to be
--                decompressed from InputBuffer
--            Current - Index into the buffer that contains the first
--                byte of the compressed strings
--
--  On Exit:  OutputBuffers - an array of decompressed strings
--            Success -  Set to TRUE if decompression succeeds, set to
--                FALSE if decompression fails.
--            Current - Index of the byte in the message succeeding the last
--                byte of the compressed string block
```

```
SET deCompressedCount = 0
SET localCurrent = 0
FOR i = 1 to StringCount
    SET dnsNameLen = 0
    SET firstLabel = 0
    allocate a buffer s[InputBufferSize]
    WHILE Current < InputBufferSize
        SET labelSize = InputBuffer[Current]
        IF labelSize == '\0' THEN
            s[dnsNameLen] = '\0'
            OutputBuffers[deCompressedCount] = s
            deCompressedCount++
            Current++
            BREAK
        ELSE IF (labelSize & 0xC0) != 0 THEN
            Current++
            localCurrent = Current + 1
            labelSize = InputBuffer[Current]
            IF labelSize > InputBufferSize THEN
                Success = FALSE
                RETURN
            END IF
            Current = labelSize
            CONTINUE
        ELSE
            IF (labelSize + Current) >= InputBufferSize THEN
                Success = FALSE
                RETURN
            END IF
            IF firstLabel == 0 THEN
                firstLabel = 1
            ELSE
                s[dnsNameLen] = '.'
                dnsNameLen++
            ENDIF


            Append
              substring InputBuffer[Current + 1, Current + labelSize]
                to s
            dnsNameLen += labelSize
            IF localCurrent != 0 THEN
                Current = localCurrent
                localCurrent = 0
            ELSE
                Current = Current + 1 + labelSize
            END IF
        END IF
    END WHILE
    If i <> deCompressedCount THEN
        Success = FALSE
        RETURN
    ENDIF
END FOR
Success = TRUE
```

```
RETURN
```

6.3.8 AD LDS DC Publication

If an AD LDS DC is running on a computer joined to an AD DS domain, the AD LDS DC SHOULD (if certain conditions are met, as described later in this section) create a serviceConnectionPoint object in the AD DS forest of the domain to which it is joined. Clients MAY use this serviceConnectionPoint object to locate this AD LDS DC.

Let O be the msDS-ServiceConnectionPointPublicationService object in the AD LDS forest whose DN is "CN=SCP Publication Service" relative to the nTDSService object in the config NC (the DN of the nTDSService object is "CN=Directory Service, CN=Windows NT, CN=Services" relative to the root of the config NC).

An AD LDS DC SHOULD create (or update, if the object already exists) a serviceConnectionPoint object unless one of the following conditions is true:

- O (the msDS-ServiceConnectionPointPublicationService object defined previously) exists and O!Enabled = false.
- O exists and O!msDS-DisableForInstances contains the DN of the nTDSDSA object of the replica.

If the LDAP add or modify operation to create or update the serviceConnectionPoint object fails for any reason, including lack of permission to create or update the serviceConnectionPoint object, the AD LDS DC SHOULD retry periodically until the operation succeeds.

The created (or updated) serviceConnectionPoint object S satisfies the following:

- If O exists and O!msDS-SCPContainer is non-null, then the DN of S is "CN={**dsaGuid**}" relative to O!msDS-SCPContainer, where **dsaGuid** is the DC's DSA GUID. Otherwise, the DN of S is "CN={**dsaGuid**}" relative to the computer object of the machine running AD LDS.
- S!serviceDNSNameType = "A"
- S!serviceClassName = "LDAP"
- S!serviceDNSName is the DNS name of the computer on which the AD LDS DC is running.
- S!serviceBindingInformation contains two values, "ldap://**dnsName**:**ldapPort**" and "ldaps://**dnsName**:**ldapsPort**", where **dnsName** is the DNS name of the computer on which the AD LDS DC is running, **ldapPort** is the port on which the AD LDS DC is listening for LDAP requests, and **ldapsPort** is the port on which the AD LDS DC is listening for SSL/TLS-protected LDAPS requests.
- S!keywords contains the following values:
  - The DSA GUID.
  - For each value of the supportedCapabilities attribute of the rootDSE, a string containing that value.
  - The string "site:**siteName**" where **siteName** is the name of the site in which the AD LDS DC is located.
  - The string "instance:**instanceName**" where **instanceName** is a name configured for this AD LDS DC, unique among all AD LDS DCs on the machine running the DC.

- If this AD LDS DC has the Schema Master FSMO role, the string "fsmo:schema".
- If the AD LDS DC has the Domain Naming FSMO role, the string "fsmo:naming".
- For each NC-replica on the AD LDS DC, excluding the NC-replica of the schema NC:
  - The string "partition:**ncName**" where **ncName** is the DN of the NC.
  - The NC GUID (that is, the value of the objectGUID attribute for the root of the NC).
- If O exists, the values (if any) present on O!keywords. (See section 6.1.1.2.4.1.5.)

For example, suppose an AD LDS replica is running on a computer whose DNS name is "adlds-01.fabrikam.com", has a DSA GUID of {d07c66ed-b55e-4472-b09c-1ae35980}, possesses both FSMO roles, and has a single application NC whose name is "CN=FirstAppNC" and whose GUID is {32079ab-9e49-4c4e-ad36-0f2b8a63f12b}. Further assume that it is listening on ports 50000 and 50001 for LDAP and LDAPS traffic, respectively, is located in a site named "Default-First-Site-Name", has an instance name of "TestInstance", and there are no keywords on O!keywords. The resulting serviceConnectionPoint object could be as follows (depending on the DN and GUID of the config NC).

```
S!serviceDnsNameType = "A"
S!serviceClassName = "LDAP"
S!serviceDNSName = "adlds-01.fabrikam.com"
S!serviceBindingInformation = {
    "ldap://adlds-01.fabrikam.com:50000",
    "ldaps://adlds-01.fabrikam.com:50001"
    }
S!keywords = {
    "d07c66ed-b55e-4472-b09c-1ae35980",
    "1.2.840.113556.1.4.1851",
    "1.2.840.113556.1.4.1791",
    "site:Default-First-Site-Name",
    "instance:TestInstance",
    "fsmo:schema",
    "fsmo:naming",
    "partition:CN=FirstAppNC",
    "32079ab-9e49-4c4e-ad36-0f2b8a63f12b",
    "partition:CN=Configuration,CN={FD783EE9-0216-4B83-8A2A-
        60E45AECCB81}",
    "23b65d43-a701-44b9-9e04-a6555df722eb"
    }
```

6.4 Domain Join

A machine is said to be "joined to a domain" if certain state exists on the machine and in the domain NC. The necessary state is specified in the remainder of this section. The state enables the machine and the domain to mutually authenticate using various protocols (for example, [MS-NRPC]).

6.4.1 State of a Machine Joined to a Domain

The following variables are part of the state of any machine joined to a domain:

- *domain-secret*: An even-numbered sequence of bytes, with no embedded zero values, containing the secret shared between the machine and the domain. There are no minimum or maximum

length constraints imposed on *domain-secret*; implementations MUST NOT assume any such limitations.

- *machine-account-name*: The sAMAccountName of the machine's computer object within the domain.
- *domain-name*: A tuple containing:
  - netbios: The NetBIOS name of the domain
  - dns: The fully qualified DNS name of the domain

  If the domain has a DNS name, domain-name.dns contains it. If the domain has a NetBIOS name, domain-name.netbios contains it. The value of at least one of these variables is not NULL.
- *domain-locator*: Implementation-specific state sufficient to locate a domain controller of the domain. If the implementation is capable of locating a domain controller given domain-name, then domain-locator can be NULL.
- *supported-encryption-types*: A set of encryption algorithms that can be used by the Key Distribution Center (KDC) to generate tickets for the machine account. This value can be NULL if the machine supports default encryption types used by a given implementation of the KDC.

The specific choices made in implementing a machine joined to a domain (for example, for representing these variables and for generating names) are outside the state model. For Windows, machine-account-name equals the machine name (result of GetComputerName) with "$" appended, and domain-locator is NULL.

6.4.2 State in an Active Directory Domain

A machine *m* that is a member of an Active Directory domain *d* has a corresponding object *o* in *d*'s domain NC. The object o is called the *machine account* of the joined machine *m*. The objectClass attribute of *o* contains the class computer. In addition to objectClass, the following attributes of *o* are significant to the membership of *m* in *d*:

- userAccountControl
- sAMAccountName
- unicodePwd
- dNSHostName
- servicePrincipalName
- msDs-supportedEncryptionTypes

The syntax and other details of these attributes are documented in [MS-ADA1], [MS-ADA2], and [MS-ADA3].

The following predicates are satisfied by the joined machine *m*'s state and the state of object *o*:

- the domain *d*'s NetBIOS name equals *m.domain-name.netbios*
- the domain *d*'s fully qualified DNS name equals *m.domain-name.dns*
- *o*!userAccountControl & ADS_UF_WORKSTATION_TRUST_ACCOUNT ≠ 0

- *o*!sAMAccountName equals *m.machine-account-name*
- *o*!unicodePwd equals *m.domain-secret*
- *o*!msDs-supportedEncryptionTypes equals m.supported-encryption-types, in the format specified in [MS-KILE] section 2.2.6. This attribute may not be set if m.supported-encryption-types is NULL.

Section 6.1.1.2.1.1.4 specifies the representation of a domain's NetBIOS name. A domain's fully qualified DNS name is derived from the DN of its root object, as specified in section 3.1.1.1.5.

The specific choices made in implementing a machine joined to a domain (for example, for maintaining these variables) are outside the state model. Windows may periodically update m.domain-secret on the client machine and o.domain-secret in the Windows Active Directory. This behavior is not required for a functional domain join.

6.4.3 Relationship to Protocols

A joined machine's domain-secret can be used by the Netlogon, NTLM, and Kerberos authentication protocols as a parameter for machine authentication to the domain. A joined machine's supported-encryption-types can be used by the Netlogon and Kerberos authentication protocol as a parameter for machine authentication to the domain. Further Netlogon, NTLM, and Kerberos authentication protocol documentation can be found in [MS-NRPC], [MS-NLMP], and [MS-KILE], respectively.

6.5 Unicode String Comparison

This section specifies how the Unicode sort methods specified in [MS-UCODEREF] are utilized to perform comparisons of Unicode strings.

6.5.1 String Comparison by Using Sort Keys

To compare strings, the implementer needs to get a "sort key" for each string (see [MSASRT]). A binary comparison of the sort keys can then be used to arrange the strings in any desired order.

This section utilizes the GetWindowsSortKey and CompareSortKeys procedures, which are specified in [MS-UCODEREF].

The flags that need to be passed to GetWindowsSortKey depend on the comparison being performed. This is specified in the following table.

Comparison being performed	Flags (from [MS-UCODEREF])
UnicodeString Comparison Rule (section 3.1.1.2.2.4.13) LDAP_SERVER_SORT_OID sorting (section 3.1.1.3.4.1.13), except for phonetic display name sort	NORM_IGNORECASE | NORM_IGNOREKANATYPE | NORM_IGNORENONSPACE | NORM_IGNOREWIDTH
Phonetic display name sort (section 3.1.1.3.4.1.13)	NORM_IGNORECASE | NORM_IGNORENONSPACE

In order to compare two strings, StringA and StringB, the following procedure is used. The value of flags is as specified in the table above. The value of LCID is the locale identifier (section 2.2.1) for the locale being used to compare the strings. To determine what value to pass for LCID, see sections 3.1.1.2.2.4.13 and 3.1.1.3.4.1.13. Note that when performing phonetic display name sort, LCID must be set equal to "1.2.840.113556.1.4.1538" (the Japanese sort order).

```
set SortKeyA to call GetWindowsSortKey(StringA, LCID, flags)
set SortKeyB to call GetWindowsSortKey(StringB, LCID, flags)
set Result to call CompareSortKeys(SortKeyA, SortKeyB)
if Result is "SortKeyA is equal to SortKeyB"
 StringA is considered equal to StringB
else if Result is "SortKeyA is less than SortKeyB"
 StringA is sorted prior to StringB
else
 assert Result must be "SortKeyA is greater than SortKeyB"
 StringA is sorted after StringB
endif
```

Any sorting mechanism may be used to arrange these strings by comparing their sort keys.

6.6 Claims IDL

For ease of implementation, the full IDL for the data types used for claims is provided as follows, where "ms-dtyp.idl" is the IDL found in [MS-DTYP] Appendix A.

```
import "ms-dtyp.idl";

[ uuid (BBA9CB76-EB0C-462C-AA1B-5D8C34415701),
  version(1.0),
  pointer_default(unique)
]
interface Claims
{
    typedef [string] wchar_t *CLAIM_ID;
    typedef [string] wchar_t **PCLAIM_ID;

    typedef enum _CLAIM_TYPE
    {
        CLAIM_TYPE_INT64 = 1,
        CLAIM_TYPE_UINT64 = 2,
        CLAIM_TYPE_STRING = 3,
        CLAIM_TYPE_BOOLEAN = 6

    } CLAIM_TYPE, *PCLAIM_TYPE;

    typedef enum _CLAIMS_SOURCE_TYPE
    {
        CLAIMS_SOURCE_TYPE_AD = 1,
        CLAIMS_SOURCE_TYPE_CERTIFICATE
    } CLAIMS_SOURCE_TYPE;

    typedef enum _CLAIMS_COMPRESSION_FORMAT
    {
        COMPRESSION_FORMAT_NONE = 0,
        COMPRESSION_FORMAT_LZNT1 = 2,
        COMPRESSION_FORMAT_XPRESS = 3,
        COMPRESSION_FORMAT_XPRESS_HUFF = 4
    } CLAIMS_COMPRESSION_FORMAT;

    typedef struct _CLAIM_ENTRY
    {
      CLAIM_ID Id;
      CLAIM_TYPE Type;
```

```
  [switch_is(Type), switch_type(CLAIM_TYPE)]
    union
    {
    [case(CLAIM_TYPE_INT64)]
      struct
      {
        [range(1, 10*1024*1024)] ULONG ValueCount;
        [size_is(ValueCount)] LONG64* Int64Values;
      };
    [case(CLAIM_TYPE_UINT64)]
      struct
      {
        [range(1, 10*1024*1024)] ULONG ValueCount;
        [size_is(ValueCount)] ULONG64* Uint64Values;
      };
    [case(CLAIM_TYPE_STRING)]
      struct
      {
        [range(1, 10*1024*1024)] ULONG ValueCount;
        [size_is(ValueCount), string] LPWSTR* StringValues;
      };
    [case(CLAIM_TYPE_BOOLEAN)]
      struct
      {
        [range(1, 10*1024*1024)] ULONG ValueCount;
        [size_is(ValueCount)] ULONG64* BooleanValues;
      };
    [default]
      ;
  } Values;
} CLAIM_ENTRY,
 *PCLAIM_ENTRY;

typedef struct _CLAIMS_ARRAY
{
    CLAIMS_SOURCE_TYPE                          usClaimsSourceType;
    ULONG                                       ulClaimsCount;
    [size_is(ulClaimsCount)] PCLAIM_ENTRY       ClaimEntries;
} CLAIMS_ARRAY, *PCLAIMS_ARRAY;

typedef struct _CLAIMS_SET
{
    ULONG                                           ulClaimsArrayCount;
    [size_is(ulClaimsArrayCount)] PCLAIMS_ARRAY     ClaimsArrays;
    USHORT                                          usReservedType;
    ULONG                                           ulReservedFieldSize;
    [size_is(ulReservedFieldSize)]  BYTE            *ReservedField;
} CLAIMS_SET, *PCLAIMS_SET;

typedef struct _CLAIMS_SET_METADATA
{
    ULONG                           ulClaimsSetSize;
    [size_is(ulClaimsSetSize)] BYTE *ClaimsSet;
    CLAIMS_COMPRESSION_FORMAT       usCompressionFormat;
    ULONG                           ulUncompressedClaimsSetSize;
    USHORT                          usReservedType;
    ULONG                           ulReservedFieldSize;
    [size_is(ulReservedFieldSize)]  BYTE  *ReservedField;
} CLAIMS_SET_METADATA, *PCLAIMS_SET_METADATA;
```

```
}
```

7 Change Tracking

This section identifies changes that were made to the [MS-ADTS] protocol document between the January 2013 and August 2013 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.

- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1 Introduction	Updated content for Windows Server 2012 R2 operating system.	Y	Content updated.
1.2.1 Normative References	Added reference [MS-GPSB].	Y	Content updated.
2.2.9 Search Flags	Updated content for Windows Server 2012 R2.	Y	Content updated.
2.2.12 Group Type Flags	67778 Changed cross-reference from [MS-ADA1] to internal topic "Constraints".	N	Content updated.
3.1.1.1.5.1 Tombstone Lifetime and Deleted-Object Lifetime	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.1.9 DCs, usn Counters, and the Originating Update Stamp	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.1.16 Delayed Link Processing	Revised product version information.	Y	Product behavior note updated.
3.1.1.2.1	Updated content for Windows Server	Y	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
Schema NC	2012 R2.		updated.
3.1.1.2.2.2 LDAP Representations	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.2.3 Attributes	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.2.3.3 Property Set	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.1.1.3 Attributes	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.1.1.4 Classes	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.1.1.5 Auxiliary Classes	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.1.2.4 Alternative Forms of DNs	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.1.3.1 Search Filters	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.1.3.2 Selection Filters	Added section with content for Windows Server 2012 R2.	Y	New content added.
3.1.1.3.1.5.1 unicodePwd	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.1.6 Dynamic Objects	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.2 rootDSE Attributes	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.2.25 domainControllerFunctionality	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.2.26 domainFunctionality	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.2.27 forestFunctionality	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.2.40 spnRegistrationResult	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.3 rootDSE Modify Operations	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.3.10	Updated content for Windows Server	Y	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
fixupInheritance	2012 R2.		updated.
3.1.1.3.3.14 schemaUpgradeInProgress	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.3.30 dumpLinks	Added section with content for Windows Server 2012 R2.	Y	New content added.
3.1.1.3.3.31 schemaUpdateIndicesNow	Added section with content for Windows Server 2012 R2.	Y	New content added.
3.1.1.3.3.32 null	Added section with content for Windows Server 2012 R2.	Y	New content added.
3.1.1.3.4 LDAP Extensions	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.1 LDAP Extended Controls	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.1.3 LDAP_SERVER_DIRSYNC_OID	67428 Clarified that search results must always include objectGuid and instanceType attributes for each object.	Y	Content updated.
3.1.1.3.4.1.3 LDAP_SERVER_DIRSYNC_OID	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.1.5 LDAP_SERVER_EXTENDED_DN_OID	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.1.6 LDAP_SERVER_GET_STATS_OID	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.1.9 LDAP_SERVER_NOTIFICATION_OID	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.1.13 LDAP_SERVER_SORT_OID and LDAP_SERVER_RESP_SORT_OID	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.1.23 LDAP_SERVER_RODC_DCPROMO_OID	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.1.29 LDAP_SERVER_DIRSYNC_EX_OID	67428 Clarified the case in which the LDAP_SERVER_DIRSYNC_EX_OID control returns unchanged attributes.	Y	Content updated.
3.1.1.3.4.1.30 LDAP_SERVER_UPDATE_STATS_OID	67421 Changed OID values in the statistic	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
	names table.		
3.1.1.3.4.1.32.1 Require Sort Index	67430 Modified hintValue information and included an example snippet.	Y	Content updated.
3.1.1.3.4.1.32.2 Soft Size Limit	67430 Modified hintValue information and included an example snippet.	Y	Content updated.
3.1.1.3.4.1.34 LDAP_SERVER_SET_OWNER_OID	Added section with content for Windows Server 2012 R2.	Y	New content added.
3.1.1.3.4.1.35 LDAP_SERVER_BYPASS_QUOTA_OID	Added section with content for Windows Server 2012 R2.	Y	New content added.
3.1.1.3.4.2 LDAP Extended Operations	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.2.5 LDAP_SERVER_BATCH_REQUEST_OID	67386 Revised the definition for LDAP_SERVER_BATCH_REQUEST_OID.	Y	Content updated.
3.1.1.3.4.2.5 LDAP_SERVER_BATCH_REQUEST_OID	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.3 LDAP Capabilities	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.4 LDAP Matching Rules (extensibleMatch)	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.4.4 LDAP_MATCHING_RULE_DN_WITH_DATA	Added section with content for Windows Server 2012 R2.	Y	New content added.
3.1.1.3.4.5 LDAP SASL Mechanisms	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.6 LDAP Policies	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.3.4.7 LDAP Configurable Settings	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.4.4 Extended Access Checks	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.5.1.3 Uniqueness Constraints	Added section with content for Windows Server 2012 R2.	Y	New content added.
3.1.1.5.1.7	67696	Y	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
Urgent Replication	Modified information on the lockoutTime attribute.		updated.
3.1.1.5.2 Add Operation	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.5.2.1 Security Considerations	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.5.2.2 Constraints	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.5.2.5 Quota Calculation	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.5.2.8.1 Constraints	67598 Clarified crossRef validation for the case of an existing crossRef.	Y	Content updated.
3.1.1.5.3 Modify Operation	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.5.3.1 Security Considerations	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.5.3.2 Constraints	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.5.3.7.2 Undelete Constraints	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.5.5.5 Constraints	67611 Amended the list of constraints.	Y	Content updated.
3.1.1.9 Optional Features	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.10.1 Forest Revision	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.10.2 RODC Revision	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.10.3 Domain Revision	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.11.1 Informative Overview	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.11.1.3 Claim Source	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.11.1.4 Claims Issuance	Updated content for Windows Server 2012 R2.	Y	Content updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3.1.1.11.2.1 GetClaimsForPrincipal	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.11.2.4 GetConstructedClaims	Added section with content for Windows Server 2012 R2.	Y	New content added.
3.1.1.11.2.11 TransformClaimsOnTrustTraversal	67417 Changed the trustDsName procedure's status value ERROR_NO_CONFIGURATION to ERROR_DS_OBJ_NOT_FOUND.	Y	Content updated.
3.1.1.11.2.12 GetClaimsTransformationRulesXml	67417 Changed the error return value from ERROR_NO_CONFIGURATION to ERROR_DS_OBJ_NOT_FOUND.	Y	Content updated.
3.1.1.11.2.17 ValidateClaimDefinition	Updated content for Windows Server 2012 R2.	Y	Content updated.
3.1.1.11.2.18 GetAuthSiloClaim	Added section with content for Windows Server 2012 R2.	Y	New content added.
5.1.1.1.1 Simple Authentication	Clarified user account lockout and password policy on simple bind.	Y	Content updated.
5.1.1.1.2 SASL Authentication	Updated content for Windows Server 2012 R2.	Y	Content updated.
5.1.1.1.3 Sicily Authentication	Updated content for Windows Server 2012 R2.	Y	Content updated.
5.1.2.1 Using SASL	Updated content for Windows Server 2012 R2.	Y	Content updated.
5.1.3.2.1 Control Access Rights	Updated content for Windows Server 2012 R2.	Y	Content updated.
5.1.3.2.2 Validated Writes	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.1.2.1.1 Cross-Ref Objects	67605 Expanded the DNS name structure held by dnsRoot when Enabled equals false.	Y	Content updated.
6.1.1.2.2.1.2.1.2 Connection Object	Changed "domains" to "sites".	N	Content updated.
6.1.1.2.4.1.2 dSHeuristics	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.1.2.4.1.2	Clarified the availability of the	Y	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
dSHeuristics	DoNotVerifyUPNUniqueness heuristic.		updated.
6.1.1.2.7.78 DS-Read-Partition-Secrets	Added section with content for Windows Server 2012 R2.	Y	New content added.
6.1.1.2.7.79 DS-Write-Partition-Secrets	Added section with content for Windows Server 2012 R2.	Y	New content added.
6.1.1.2.7.80 DS-Set-Owner	Added section with content for Windows Server 2012 R2.	Y	New content added.
6.1.1.2.7.81 DS-Bypass-Quota	Added section with content for Windows Server 2012 R2.	Y	New content added.
6.1.1.5.3 Sam Server Object	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.1.6.13 Read-Only Domain Controllers	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.1.6.14 Enterprise Read-Only Domain Controllers	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.3.4 Security Considerations	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.4.1 ntMixedDomain	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.4.2 msDS-Behavior-Version: DC Functional Level	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.4.3 msDS-Behavior-Version: Domain NC Functional Level	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.4.4 msDS-Behavior-Version: Forest Functional Level	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.6.3 Prerequisites/Preconditions	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.6.7.3 msDs-supportedEncryptionTypes	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.6.7.4 msDS-TrustForestTrustInfo	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.1.6.7.9	Updated content for Windows Server	Y	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
trustAttributes	2012 R2.		updated.
6.1.6.9.1.2 Kerberos Usages of trustAuthInfo Attributes	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.3.1.2 DS_FLAG Options Bits	67419 Removed Windows Server 2012 from list of operating systems not running for the DS_FLAG bit FW8 (DS_DS_8_FLAG, 0x00004000).	N	Content updated.
6.3.1.2 DS_FLAG Options Bits	Updated content for Windows Server 2012 R2.	Y	Content updated.
6.3.3.2 Domain Controller Response to an LDAP Ping	Updated content for Windows Server 2012 R2.	Y	Content updated.

8 Index

E

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W