[MS-DFSNM]:
Distributed File System (DFS):
Namespace Management Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
03/14/2007	1.0		Version 1.0 release
04/10/2007	1.1		Version 1.1 release
05/18/2007	1.2		Version 1.2 release
06/08/2007	1.2.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
07/10/2007	1.2.2	Editorial	Revised and edited the technical content.
08/17/2007	1.3	Minor	Updated the technical content.
09/21/2007	1.4	Minor	Updated the technical content.
10/26/2007	1.4.1	Editorial	Revised and edited the technical content.
01/25/2008	2.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

This document is a specification of the Distributed File System (DFS): Namespace Management Protocol.

This protocol provides a **remote procedure call (RPC)** interface for administering **DFS** configurations. The client is an application that issues method calls on the RPC interface to administer DFS. The server is a DFS service that implements support for this RPC interface for administering DFS.

1.1 Glossary

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

8.3 Name
Access Control List (ACL)
Active Directory (AD)
Active Directory Domain Services (AD DS)
Authentication Level
Binary Large Object (BLOB)
Coordinated Universal Time (UTC)
Directory Object (or Object)
Directory Service (DS)
Distributed File System (DFS)
Distributed File System (DFS) Client
Distributed File System (DFS) Client Target Failback
Distributed File System (DFS) Client Target Failover
Distributed File System (DFS) In-Site Referral Mode
Distributed File System (DFS) Link
Distributed File System (DFS) Link Target
Distributed File System (DFS) Metadata
Distributed File System (DFS) Namespace
Distributed File System (DFS) Path
Distributed File System (DFS) Referral
Distributed File System (DFS) Referral Site Costing
Distributed File System (DFS) Root
Distributed File System (DFS) Root Target
Distributed File System (DFS) Root Scalability Mode
Domain
Domain Controller (DC)
Endpoint
FILETIME
Globally Unique Identifier (GUID)
GUIDString
Interface Definition Language (IDL)
Lightweight Directory Access Protocol (LDAP)
Little-Endian
Member Server
Microsoft Interface Definition Language (MIDL)
NetBIOS Name
Opnum
Primary Domain Controller (PDC)
Relative Distinguished Name (RDN)
Remote Procedure Call (RPC)
RPC Protocol Sequence

RPC Transport
Server Message Block (SMB)
Share
Site
System Access Control List (SACL)
Unicode
Universal Naming Convention (UNC)
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

Access Based Directory Enumeration (ABDE) mode: A mode where the server filters directory entries based on the access permissions of the client. In a **DFS** scenario, **ABDE** is enabled on the **DFS root target share** to prevent a user from seeing another user's home directory. The **DFS namespace** administrator can create a **DFS link** for a user (or user group), and a user is granted appropriate rights to the **DFS link**.

Clustered DFS Namespace: A **stand-alone DFS namespace** that is hosted on a file server cluster.

Distributed File System (DFS) Namespace Name: The second pathname component of a **DFS path**. For example, in the **DFS path** \\MyDomain\MyDfs\MyDir, the **DFS namespace name** is MyDfs.

Distributed File System (DFS) Server: A server computer running the **DFS** service, which responds to **DFS referral** requests, as specified in [MS-DFSC], as well as to the DFS: Namespace Management Protocol. Also used interchangeably to refer to the **DFS** service itself.

Distributed File System (DFS) Target: Either a **DFS root target** server or a **DFS link target** server.

Distinguished Name (DN): The unique identifier of an **object** in the **Active Directory Domain Services (AD DS)**. For more information, see [MS-ADTS].

Domain-Based DFS Namespace: A **DFS namespace** that has configuration information stored in **AD**. The **DFS namespace** may span a distributed system that is organized hierarchically into logical domains, wherein each logical domain may have a primary controller, such as a **domain controller** used in **AD**. The path to access a **domain-based DFS namespace** starts with the host **Active Directory Domain Services (AD DS)** domain name. A **domain-based DFS namespace** can have multiple **DFS root targets**, which offers high availability and load sharing at the **DFS root** level.

Domainv1-Based DFS Namespace: A type of **domain-based DFS namespace** that has its **DFS metadata** stored in **AD** in an ftDfs type **object**.

Domainv2-Based DFS Namespace: A type of **domain-based DFS namespace** that has its **DFS metadata** stored in the form of individual **LDAP** entries, with one **LDAP** entry per **DFS link**. Each **LDAP** entry contains the **DFS metadata**, such as targets, properties, and other information, that corresponds to that entity.

Host Name: Name of a computer used for identification and access purposes by both humans and other computers in a network.

Share Name: The name of a **share**.

Site Cost (or AD DS Site Cost): An **Active Directory Domain Services (AD DS)** administrator-defined numerical value meant to indicate the bandwidth or actual monetary cost of transmitting data between two **AD DS sites**. Only a comparison between two **site cost** values is meaningful, with a lower **AD DS site** preferred to a higher **AD DS site cost**.

Stand-Alone DFS Namespace: A **DFS namespace** that has **DFS metadata** stored locally on the host server. The path to access the **DFS root** or a **DFS link** starts with the **DFS root target host name**. A **stand-alone DFS namespace** has only one **DFS root target**. Stand-alone **DFS roots** are not fault-tolerant; when the **DFS root target** is unavailable, the entire **DFS namespace** is inaccessible. Stand-alone **DFS roots** can be made fault-tolerant by creating them on clustered file servers.

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

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

[MS-ADA2] Microsoft Corporation, "Active Directory Schema Attributes M", July 2006.

[MS-ADA3] Microsoft Corporation, "Active Directory Schema Attributes N-Z", July 2006.

[MS-ADSC] Microsoft Corporation, "Active Directory Schema Classes", July 2006.

[MS-ADTS] Microsoft Corporation, "Active Directory Technical Specification", July 2006.

[MS-DFSC] Microsoft Corporation, "Distributed File System (DFS): Referral Protocol Specification", July 2006.

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

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

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

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", July 2006.

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", July 2006.

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Version 2.0 Protocol Specification", July 2006.

[MS-SRVS] Microsoft Corporation, "Server Service Remote Protocol Specification", July 2006.

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

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

[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

[UNICODE] The Unicode Consortium, "Unicode Home Page", 2006, http://www.unicode.org/

[X680] ITU-T, "Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", Recommendation X.680, July 2002, http://www.itu.int/rec/T-REC-X.680/en

Note There is a charge to download the specification.

[XML] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C Recommendation, September 2006, http://www.w3.org/TR/REC-xml

[XMLSCHEMA] World Wide Web Consortium, "XML Schema", September 2005, http://www.w3.org/2001/XMLSchema

1.2.2 Informative References

[MSDFS] Microsoft Corporation, "How DFS Works", March 2003, http://technet2.microsoft.com/WindowsServer/en/library/a9096e88-1634-4da6-b820-537341d349061033.mspx

[NOVELL] Chappell, L.A. and Hakes, D.E., "Novell's Guide to NetWare LAN Analysis, 2nd Edition", Novell Press, June 1994, ISBN: 0782113621.

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

[RFC2165] Veizades, J., Guttman, E., Perkins, C., and Kaplan, S., "Service Location Protocol", RFC 2165, June 1997, http://www.ietf.org/rfc/rfc2165.txt

[RFC2518] Goland, Y., Whitehead, E., Faizi, A., Carter, S., and Jensen, D., "HTTP Extensions for Distributed Authoring—WebDAV", RFC 2518, February 1999, http://www.ietf.org/rfc/rfc2518.txt

[RFC3530] Shepler, S., et al., "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003, http://www.ietf.org/rfc/rfc3530.txt

1.3 Protocol Overview (Synopsis)

The DFS: Namespace Management Protocol is one of a collection of protocols that group **shares** located on different servers by combining various storage media into a single logical namespace. The **DFS namespace** is a virtual view of the share. When a user views the namespace, the directories and files in it appear to reside on a single share. Users can navigate the namespace without needing to know the server names or shares hosting the data. DFS also provides redundancy of namespace service.

Access to a DFS namespace requires the **DFS client**. The DFS client uses the **DFS Referral** Protocol, as specified in [MS-DFSC], to ascertain the existence of the DFS namespace and to determine the shares to access on servers that participate in the DFS namespace. The DFS Referral Protocol navigates through the DFS namespace by appropriately issuing referral requests to a **domain controller (DC)** or to a **DFS root target** server to resolve the original path to a share on a server that actually contains the data being accessed. For more information on DFS and the DFS client, see [MSDFS]. For more information on how the DFS Referral Protocol operates within the

context of the **Server Message Block (SMB)** Protocol, as specified in [MS-SMB], which is the transport for DFS referrals. For more information on DFS referrals, see [MS-DFSC] section 2.

DFS namespace information, such as name, **DFS link** name, **DFS link target**, and so on, is stored in the **DFS metadata** of the namespace. Depending on where the DFS metadata is stored, the DFS namespace is said to be "domain-based" or "stand-alone".

- **Domain-Based DFS Namespace**: A well-known container in the **domain** directory, known as the DFS configuration container, holds the DFS metadata for a domain-based DFS namespace. An **Active Directory Domain Services (AD DS)** "**object**" exists for each domain-based DFS namespace in the DFS configuration container. DFS metadata of a domain-based DFS namespace is stored as a **binary large object (BLOB)** in an attribute of the per DFS namespace AD DS object. A domain-based DFS namespace can have no more than one DFS root target. The **DFS root** name of a domain-based DFS namespace has the AD DS domain as its first component. A DFS client issues a referral request to a DC in order to identify the DFS root targets of the DFS namespace.
- **Stand-Alone DFS Namespace**: DFS metadata is stored in an implementation-specific format on the DFS root target server itself. A stand-alone DFS namespace supports only one DFS root target. The DFS root name of a stand-alone DFS namespace has a **host name** as its first component. A DFS client issues referral requests to the DFS root target server to access the DFS namespace. A stand-alone DFS namespace may be clustered to provide high availability of the DFS namespace.<1>

The DFS: Namespace Management Protocol is used to configure DFS services. This protocol is used primarily by administrative applications that run on client computers to connect and configure **DFS servers**. It consists of the RPC methods that can be issued from an administrative client computer to the protocol server on a DC or a DFS root target server. An administrator can use this protocol to perform various DFS namespace administration operations, such as creating or deleting a DFS namespace, adding or removing DFS root targets, adding or removing DFS links, and adding or removing targets to an existing link. The DFS: Namespace Management Protocol includes the following:

- Eleven basic methods for configuring stand-alone DFS namespaces and domain-based DFS namespaces, as specified in section 3.2.5.1.
- Four methods that support extended access to configurations of a DFS namespace, as specified in section 3.2.5.2.
- Three methods for configuring root targets in a **domainv1-based DFS namespace**, as specified in section 3.2.5.3.
- Three methods for configuring a stand-alone DFS namespace, as specified in section 3.2.5.4.
- Two methods relating to the association between a DFS server and the DC used by a domain-based DFS namespace, as specified in section 3.2.5.5.

Much of the configuration information that is communicated through this protocol is marshaled through two unions: **DFS_INFO_STRUCT** and **DFS_INFO_ENUM_STRUCT**. The usage model of these unions is for the client to specify a *Level* parameter to determine which union case to use. Each level corresponds to a specific DFS_INFO_*n* structure, where *n* is the level number. Arrays of DFS_INFO_*n* structures are marshaled using DFS_INFO_*n*_CONTAINER structures. Levels 1, 2, 3, 4, 5, 6, 8, and 9 are common, and are shared across both the **DFS_INFO_STRUCT** and **DFS_INFO_ENUM_STRUCT** unions. Levels 7, 50, 100, 101, 102, 103, 104, 105, 106, 107, and 150 are unique to the **DFS_INFO_STRUCT** union, and Levels 200 and 300 are unique to the **DFS_INFO_ENUM_STRUCT** union.

While a number of methods use the common configuration information structures, not all methods support all levels. The following table lists the levels used in the **DFS_INFO_STRUCT** and **DFS_INFO_ENUM_STRUCT** unions, their singleton and array structures, and the methods with which the level can be used.

Level	Structure	Array Structure	NetrDfs GetInfo	NetrDfs Enum	NetrDfs SetInfo	NetrDfs SetInfo2	NetrDfs EnumEx
1	DFS_INFO_1	DFS_INFO_1_ CONTAINER	X	X			X
2	DFS_INFO_2	DFS_INFO_2_ CONTAINER	X	X			X
3	DFS_INFO_3	DFS_INFO_3_ CONTAINER	X	X			X
4	DFS_INFO_4	DFS_INFO_4_ CONTAINER	X	X			X
5	DFS_INFO_5	DFS_INFO_5_ CONTAINER	X	X			X
6	DFS_INFO_6	DFS_INFO_6_ CONTAINER	X	X			X
7	DFS_INFO_7	N/A	X				
8	DFS_INFO_8	DFS_INFO_8_ CONTAINER	X	X			X
9	DFS_INFO_9	DFS_INFO_9_ CONTAINER	X	X			X
50	DFS_INFO_50	N/A	X				
100	DFS_INFO_100	N/A	X		X	X	
101	DFS_INFO_101	N/A			X	X	
102	DFS_INFO_102	N/A			X	X	
103	DFS_INFO_103	N/A			X	X	
104	DFS_INFO_104	N/A			X	X	
105	DFS_INFO_105	N/A			X	X	

Level	Structure	Array Structure	NetrDfs GetInfo	NetrDfs Enum	NetrDfs SetInfo	NetrDfs SetInfo2	NetrDfs EnumEx
106	DFS_INFO_106	N/A			X	X	
107	DFS_INFO_107	N/A			X	X	
150	DFS_INFO_150	N/A	X		X	X	
200	DFS_INFO_200	DFS_INFO_200_CONTAINER					X
300	DFS_INFO_300	DFS_INFO_300_CONTAINER		X			X

1.4 Relationship to Other Protocols

The DFS: Namespace Management Protocol is used to configure and administer DFS namespaces. It depends on RPC for its transport.

The DFS: Namespace Management Protocol is the preferred and recommended method of performing DFS namespace operations. This protocol is used in many operations, for example, creating a new DFS namespace or adding or removing **DFS links** or DFS link targets. All of these operations require updating the DFS metadata of a DFS namespace.

The DFS Referral Protocol, as specified in [MS-DFSC], accesses the DFS metadata of a DFS namespace for providing DFS referral responses. The DFS clients issue DFS referral requests to verify the existence of a DFS namespace and to identify the targets of a **DFS path**, as specified in [MS-DFSC]. The DFS Referral Protocol permits DFS clients to navigate the DFS namespace and to locate the share on a server that contains the required data. The DFS Referral Protocol is implemented as a set of SMB Protocol extensions to commands such as TRANS2_GET_DFS_REFERRAL to request DFS referrals from DCs, as specified in [MS-SMB].

After a DFS path is resolved to a **DFS target** using the DFS Referral Protocol, a client accesses resources on the server identified by the DFS target using a resource access protocol, such as the following:

- SMB, as specified in [MS-SMB].
- SMB2, as specified in [MS-SMB2].
- Network file system (NFS), as specified in [RFC3530].
- Network Control Protocol (NCP), as specified in [NOVELL].
- Web Distributed Authoring and Versioning (WebDAV), as specified in [RFC2518].

A resource access protocol implementation uses name resolution protocols, such as DNS (as specified in [RFC1034]) or SLP (as specified in [RFC2165]), to resolve DFS target host names. The DFS root target MUST reside on a server that is accessible through SMB (as specified in [MS-SMB]) or SMB2 (as specified in [MS-SMB2]) while a link target MAY reside on a server that is accessible through any resource access protocol for which appropriate client-side software exists.<2>

The DFS metadata of a domain-based DFS namespace is stored in AD DS. A DFS server uses the **Lightweight Directory Access Protocol (LDAP)**, as specified in [RFC2251], to access the DFS metadata from AD DS for use with both the DFS: Namespace Management Protocol and the DFS Referral Protocol.

1.5 Prerequisites/Preconditions

The DFS: Namespace Management Protocol is an RPC interface and, as a result, has prerequisites common to RPC interfaces. These prerequisites are specified in [MS-RPCE].

Before a client invokes this protocol, it must obtain the name of a server that supports DFS services and RPC.

To avoid conflicts between updates to DFS metadata:

- There MUST be, at most, one client at a time modifying the metadata for a given DFS namespace.
- A domain-based DFS server MUST perform all DFS metadata updates to the **primary domain controller (PDC)** independently of the **DFS root scalability mode** setting of the DFS namespace.<3>

1.6 Applicability Statement

The DFS: Namespace Management Protocol is appropriate for managing a domain-based DFS namespace or a stand-alone DFS namespace.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- Supported transports: The DFS: Namespace Management Protocol uses RPC over SMB1.*x* or SMB2 as its only supported transport. For more information on transport specifications, see section 2.1.
- Protocol versions: The RPC interface for this protocol has a single version number of 3.0. This protocol MAY be extended without altering the version number by adding RPC methods to the interface, with **opnums** positioned numerically beyond those defined in this specification. A client determines whether such methods are supported by attempting to invoke the method; if the method is not supported, the RPC server MUST return an "Opnum out of range" error. RPC versioning and capacity negotiation in this situation is specified in [C706] and [MS-RPCE].<4>
- Security and authentication methods, as specified in [MS-RPCE].
- Capability negotiation: None.

1.8 Vendor-Extensible Fields

The DFS: Namespace Management Protocol does not define any vendor-extensible fields.

1.9 Standards Assignments

No standards assignments have been received for the RPC interface **universally unique identifier (UUID)** or well-known pipe name described in this document. All values used in these extensions are in private ranges, as specified in [MS-RPCE] and [MS-SMB].

2 Messages

The following sections specify how DFS: Namespace Management Protocol messages are transported and common DFS: Namespace Management Protocol data types.

2.1 Transport

The DFS: Namespace Management Protocol uses RPC over SMB, as specified in [MS-RPCE]. SMB provides commands, such as TRANS2_GET_DFS_REFERRAL, to request DFS referrals from DCs, as specified in [MS-SMB] and [MS-SMB2].

This protocol uses a **well-known endpoint**, \\PIPE\NETDFS, for RPC over SMB. The RPC interface uses transport-level authentication, as specified in [MS-RPCE]. **DFS** is not directly involved in authentication, however, the DFS service must verify if the user has administrator privileges to the namespace. The authenticated RPC interface allows RPC to negotiate the use of authentication and the **authentication level** on behalf of the client and server, as specified in [MS-RPCE] section 3.3.3.5.3. The server MUST find the security context indicated by the auth_context_id in the sec_trailer of the request and MUST ask the security provider that created the security context to retrieve the client identity.

The UUID of the RPC interface for the DFS: Namespace Management Protocol is 4FC742E0-4A10-11CF-8273-00AA004AE673. The RPC interface version number is 3.0.

This protocol allows any user to establish a connection to a DFS server. It uses the underlying RPC protocol to retrieve the identity of the caller that made the request, as specified in [MS-RPCE], section 3.3.3.5.3. The RPC server SHOULD use this identity to verify method-specific access.

2.2 Message Syntax

The following sections specify how DFS: Namespace Management Protocol messages are transported and DFS: Namespace Management Protocol message syntax.

2.2.1 Common Conventions

Unless otherwise specified, all strings in this protocol are null-terminated strings of UTF-16, as specified in [UNICODE] characters. Backslashes (\) in string descriptions are literal characters.

Constructs of the form "<value>" in strings are placeholders to be replaced with client- or server-specified values. For example, the string description "\\<servername>\<share>" would take the form "\\myserver\myshare" when populated with the values "myserver" for the <servername> placeholder and "myshare" for the <share> placeholder.

A number of string formats are common to many of the data types and methods in this protocol. To avoid repetition, this section describes the specific formats.

2.2.1.1 Host Name

A host name refers to a DC host name, a DFS root target server host name, or a DFS link target server host name.

While resource access protocols, such as SMB, SMB2, NFS, NCP, or WebDAV, place restrictions on host names, at the level of the DFS: Namespace Management Protocol and the DFS Referral Protocol, host names are handled as string literals.

Within the DFS: Namespace Management Protocol, a valid host name is any sequence of **Unicode** characters, with two exceptions:

- A host name MUST NOT contain a backslash (\).
- A host name MUST NOT contain a null character (\0).

2.2.1.2 Share Name

Unless specified otherwise, a **share name** is a null-terminated Unicode character string whose format depends on the actual file server protocol used to access the share. Examples of file server protocols include SMB (as referenced in [MS-SMB]), NFS (as referenced in [RFC3530]), and NCP (as referenced in [NOVELL]).

2.2.1.3 AD DS Domain Name

Unless specified otherwise, an AD DS domain name is a null-terminated Unicode character string consisting of the name of an AD DS domain in an AD DS forest. This can be either a **NetBIOS name** or a fully qualified name, as specified in [MS-ADTS].

2.2.1.4 UNC Path

A **Universal Naming Convention (UNC)** path can be used to access network resources and is a null-terminated Unicode character string whose format is as follows:

\\<hostname>\<sharename>[\<objectname>]*

where:

- <hostname> is the host name of a server or the AD DS domain name of an AD DS domain hosting resource; it may also be an IP address.
- <sharename> is the share or the resource accessed.
- <objectname> is the name of an object; this name depends on the actual resource accessed.

The notation "[\<objectname>]*" indicates that zero or more object names may exist in the path, and each <objectname> is separated from the immediately preceding <objectname> with a backslash path separator. In a UNC path used to access files and directories in an SMB share, for example, <objectname> may be the name of a file or a directory.

The <hostname>, <sharename>, and <objectname> are referred to as "pathname components" or "path components". A valid UNC path consists of two or more path components. The <hostname> is referred to as the "first pathname component", the <sharename> as the "second pathname component", and so on. The last component of the path is also referred to as the "leaf component".

The protocol used to access the resource, and the type of resource being accessed, define the size and valid characters for a path component. The only limitation that DFS places on path components is that they must be at least one character in length and must not contain a backslash or null.

2.2.1.5 DFS Root

A DFS root has one of the following UNC path formats.

```
\\<ServerName>\<DFSName>
```

```
\\<DomainName>\<DFSName>
```

where:

<ServerName> is the host name (as specified in section 2.2.1.1) of a DFS root target (as specified in section 2.2.1.7) of the DFS namespace.

<DomainName> is the AD DS domain name (as specified in section 2.2.1.3) of the domain hosting the domain-based DFS namespace.

<DFSName> is the DFS namespace name. A stand-alone DFS namespace can be referred to only by the first format. A domain-based DFS namespace can be referred to in either format, with the second format preferred.

2.2.1.6 DFS Link

A DFS link has one of the following UNC path formats.

```
\\<ServerName>\<DFSName>\<LinkPath>
\\<DomainName>\<DFSName>\<LinkPath>
```

where:

<ServerName> is the host name of a DFS root target of the DFS namespace.

<DomainName> is the AD DS domain name of the domain hosting the domain-based DFS namespace.

<DFSName> is the DFS namespace name.

<LinkPath> is the path of the DFS link relative to the DFS root target share. A stand-alone DFS namespace can be referred to only by the first format. A domain-based DFS namespace can be referred to in either format, with the second format preferred.

2.2.1.7 DFS Root Target

A DFS root target is a UNC path with the following format.

\\<servername>\<sharename>

where:

<servername> is the host name of a DFS root target server.

<sharename> is the share name corresponding to a DFS namespace on the DFS root target server.

2.2.1.8 DFS Link Target

A DFS link target is any UNC path that resolves to a directory.

2.2.1.9 DFS Target

A DFS target is either a DFS root target or a DFS link target.

2.2.2 Common Data Types

In addition to RPC base types and definitions, as specified in [C706] and [MS-RPCE], the following sections use the definitions of DWORD, **GUID**, and WCHAR, as specified in [MS-DTYP]. Any remaining data types in this section are defined in the **Interface Definition Language (IDL)** specification for this RPC interface.

2.2.2.1 NET_API_STATUS

The **NET_API_STATUS** type is an unsigned, 32-bit integer value representing the return code from an RPC method.

This type is declared as follows:

```
typedef DWORD NET_API_STATUS;
```

This protocol uses Microsoft Win32 error codes. The values are taken from the Windows error number space, as specified in [MS-ERREF]. Vendors SHOULD reuse those values with their indicated meanings. Choosing any other value runs the risk of collisions in the future.

2.2.2.2 NETDFS_SERVER_OR_DOMAIN_HANDLE

The **NETDFS_SERVER_OR_DOMAIN_HANDLE** is a pointer to a Unicode string representing a host name for an RPC method.

This type is declared as follows:

```
typedef WCHAR* NETDFS_SERVER_OR_DOMAIN_HANDLE;
```

2.2.2.3 DFS_INFO_STRUCT

The **DFS_INFO_STRUCT** union relates to the **NetrDfsGetInfo**, **NetrDfsSetInfo**, and **NetrDfsSetInfo2** methods when used to retrieve or set the configuration of the DFS server. The usage model of this union is for the client to specify a *Level* parameter to determine which case of the **DFS_INFO_STRUCT** to use.

The **DFS_INFO_STRUCT** union has the following format.

```
typedef
[switch_type(unsigned long)]
union _DFS_INFO_STRUCT {
  [case(1)]
    DFS_INFO_1* DfsInfo1;
```

```
  [case(2)]
    DFS_INFO_2* DfsInfo2;
  [case(3)]
    DFS_INFO_3* DfsInfo3;
  [case(4)]
    DFS_INFO_4* DfsInfo4;
  [case(5)]
    DFS_INFO_5* DfsInfo5;
  [case(6)]
    DFS_INFO_6* DfsInfo6;
  [case(7)]
    DFS_INFO_7* DfsInfo7;
  [case(8)]
    DFS_INFO_8* DfsInfo8;
  [case(9)]
    DFS_INFO_9* DfsInfo9;
  [case(50)]
    DFS_INFO_50* DfsInfo50;
  [case(100)]
    DFS_INFO_100* DfsInfo100;
  [case(101)]
    DFS_INFO_101* DfsInfo101;
  [case(102)]
    DFS_INFO_102* DfsInfo102;
  [case(103)]
    DFS_INFO_103* DfsInfo103;
  [case(104)]
    DFS_INFO_104* DfsInfo104;
  [case(105)]
    DFS_INFO_105* DfsInfo105;
  [case(106)]
    DFS_INFO_106* DfsInfo106;
  [case(107)]
    DFS_INFO_107* DfsInfo107;
  [case(150)]
    DFS_INFO_150* DfsInfo150;
} DFS_INFO_STRUCT;
```

DfsInfo1: The **DFS_INFO_1** structure contains the name of a DFS root or DFS link. For more information on the specifications, see section 2.2.3.1.

DfsInfo2: The **DFS_INFO_2** structure contains information for a DFS root or DFS link. For more information on specifications, see section 2.2.3.2.

DfsInfo3: The **DFS_INFO_3** structure contains information for a DFS root or DFS link. For more information on specifications, see section 2.2.3.3.

DfsInfo4: The **DFS_INFO_4** structure contains information for a DFS root or DFS link. For more information on specifications, see section 2.2.3.4.

DfsInfo5: The **DFS_INFO_5** structure contains information for a DFS root or DFS link. For more information on specifications, see section 2.2.3.5.

DfsInfo6: The **DFS_INFO_6** structure contains information for a DFS root or DFS link. For more information on specifications, see section 2.2.3.6.

DfsInfo7: The **DFS_INFO_7** structure contains information about a DFS root or DFS link. For more information on specifications, see section 2.2.3.7.

DfsInfo8: The **DFS_INFO_8** structure contains information about a DFS root or DFS link. For more information on specifications, see section 2.2.3.8.

DfsInfo9: The **DFS_INFO_9** structure contains information about a DFS root or DFS link. For more information on specifications, see section 2.2.3.9.

DfsInfo50: The **DFS_INFO_50** structure contains information about a DFS root or DFS link. For more information on specifications, see section 2.2.3.10.

DfsInfo100: The **DFS_INFO_100** structure contains a comment associated with a DFS root or DFS link. For more information on specifications, see section 2.2.5.1.

DfsInfo101: The **DFS_INFO_101** structure describes the storage state on a DFS root, DFS link, DFS root target, or DFS link target. For more information on specifications, see section 2.2.4.1.

DfsInfo102: The **DFS_INFO_102** structure contains a time-out value for a DFS root or DFS link. For more information on specifications, see section 2.2.4.2.

DfsInfo103: The **DFS_INFO_103** structure contains properties that set specific behaviors for a DFS root or DFS link. For more information on specifications, see section 2.2.4.3.

DfsInfo104: The **DFS_INFO_104** structure contains the priority of a DFS root target or DFS link target. For more information on specifications, see section 2.2.4.4.

DfsInfo105: The **DFS_INFO_105** structure contains information about a DFS root or DFS link, including comment, state, time-out, and DFS behaviors that property flags specify. For more information on specifications, see section 2.2.4.5.

DfsInfo106: The **DFS_INFO_106** structure contains the storage state and priority for a DFS root target or DFS link target. For more information on specifications, see section 2.2.4.6.

DfsInfo107: The **DFS_INFO_107** structure contains the storage state and priority for a DFS root target or DFS link target. For more information on specifications, see section 2.2.4.7.

DfsInfo150: The **DFS_INFO_150** structure contains the self-relative security descriptor associated with the DFS link. For more information on specifications, see section 2.2.5.2.

2.2.2.4 DFS_INFO_ENUM_STRUCT

The **DFS_INFO_ENUM_STRUCT** union relates to the **NetrDfsEnum** and **NetrDfsEnumEx** methods when used to enumerate the configuration of the DFS server. The usage model of this union is for the client to specify a *Level* parameter to determine which case of the DFS_INFO_ENUM_STRUCT to use.

The **DFS_INFO_ENUM_STRUCT** union structure has the following format.

```
typedef struct _DFS_INFO_STRUCT {
  DWORD Level;
  [swtich_is(Level)] union {
    [case(1)]
      DFS_INFO_1_CONTAINER* DfsInfo1Container;
    [case(2)]
      DFS_INFO_2_CONTAINER* DfsInfo2Container;
```

```
    [case(3)]
      DFS_INFO_3_CONTAINER* DfsInfo3Container;
    [case(4)]
      DFS_INFO_4_CONTAINER* DfsInfo4Container;
    [case(5)]
      DFS_INFO_5_CONTAINER* DfsInfo5Container;
    [case(6)]
      DFS_INFO_6_CONTAINER* DfsInfo6Container;
    [case(8)]
      DFS_INFO_8_CONTAINER* DfsInfo8Container;
    [case(9)]
      DFS_INFO_9_CONTAINER* DfsInfo9Container;
    [case(200)]
      DFS_INFO_200_CONTAINER* DfsInfo200Container;
    [case(300)]
      DFS_INFO_300_CONTAINER* DfsInfo300Container;
  } DfsInfoContainer;
} DFS_INFO_STRUCT;
```

Level: A parameter whose value specifies the case of the **DFS_INFO_ENUM_STRUCT** union structure to use.

DfsInfo1Container: The **DFS_INFO_1_CONTAINER** structure contains an array of the names of DFS roots or DFS links. For more information, see section 2.2.6.1.

DfsInfo2Container: The **DFS_INFO_2_CONTAINER** structure contains an array of information for DFS roots or DFS links. For more information, see section 2.2.6.2.

DfsInfo3Container: The **DFS_INFO_3_CONTAINER** structure contains an array of information for DFS roots or DFS links. For more information, see section 2.2.6.3.

DfsInfo4Container: The **DFS_INFO_4_CONTAINER** structure contains an array of information for DFS roots or DFS links. For more information, see section 2.2.6.4.

DfsInfo5Container: The **DFS_INFO_5_CONTAINER** structure contains an array of information for DFS roots or DFS links. For more information, see section 2.2.6.5.

DfsInfo6Container: The **DFS_INFO_6_CONTAINER** structure contains an array of information for DFS roots or DFS links. For more information, see section 2.2.6.6.

DfsInfo8Container: The **DFS_INFO_8_CONTAINER** structure contains an array of information for DFS roots or DFS links. For more information, see section 2.2.6.7.

DfsInfo9Container: The **DFS_INFO_9_CONTAINER** structure contains an array of information for DFS roots or DFS links. For more information, see section 2.2.6.8.

DfsInfo200Container: The **DFS_INFO_200_CONTAINER** structure contains an array of the names of domain-based DFS namespaces in a domain-based DFS. For more information, see section 2.2.6.9.

DfsInfo300Container: The **DFS_INFO_300_CONTAINER** structure contains an array of the DFS roots hosted on a server. For more information, see section 2.2.6.10.

2.2.2.5 DFS_STORAGE_INFO

The **DFS_STORAGE_INFO** structure relates to the **NetrDfsEnum**, **NetrDfsEnumEx**, and **NetrDfsGetInfo** methods when used to enumerate DFS links and DFS targets in a namespace or to get information about a DFS link. The structure contains information about the target of a DFS root or DFS link.

The **DFS_STORAGE_INFO** structure has the following format.

```
typedef struct _DFS_STORAGE_INFO {
  unsigned long State;
  [string] WCHAR* ServerName;
  [string] WCHAR* ShareName;
} DFS_STORAGE_INFO;
```

State: Refers to the **State** field of **DFS_INFO_106**. For more information, see section 2.2.4.6.

ServerName: The pointer to a null-terminated Unicode string containing the DFS target host name.

ShareName: The pointer to a null-terminated Unicode string containing the DFS target share name.

DFS_INFO_3 and **DFS_INFO_4** structures contain one or more **DFS_STORAGE_INFO** structures, one for each DFS target.

2.2.2.6 DFS_STORAGE_INFO_1

The **DFS_STORAGE_INFO_1** structure relates to the **NetrDfsEnum**, **NetrDfsEnumEx**, and **NetrDfsGetInfo** methods when used to enumerate DFS links and targets in a namespace or to get information about a DFS link. The structure contains data about a DFS target, including the host name and share name, as well as the target state and priority. For more information on prioritization, see section 2.2.2.7.

The **DFS_STORAGE_INFO_1** structure has the following format.

```
typedef struct _DFS_STORAGE_INFO_1 {
  unsigned long State;
  [string] WCHAR* ServerName;
  [string] WCHAR* ShareName;
  DFS_TARGET_PRIORITY TargetPriority;
} DFS_STORAGE_INFO_1;
```

State: Refers to the State field of **DFS_INFO_106**. For more information, see section 2.2.4.6.

ServerName: A pointer to a null-terminated Unicode string containing the DFS target host name.

ShareName: A pointer to a null-terminated Unicode string containing the DFS target share name.

TargetPriority: A **DFS_TARGET_PRIORITY** structure containing the priority class and priority rank.

2.2.2.7 DFS_TARGET_PRIORITY

The **DFS_TARGET_PRIORITY** structure relates to the **NetrDfsSetInfo** and **NetrDfsSetInfo2** methods when used to set the priority of a DFS target in referrals from a server. It also relates to the **DFS_STORAGE_INFO_1** structure that the **NetrDfsEnum**, **NetrDfsEnumEx**, and **NetrDfsGetInfo** methods return. The structure defines the priority of a DFS target. The DFS targets can be prioritized independently of **site cost**. The DFS target priority is manually assigned to link targets and root targets and allows for load balancing of clients.

The **DFS_TARGET_PRIORITY** structure has the following format.

```
typedef struct _DFS_TARGET_PRIORITY {
  DFS_TARGET_PRIORITY_CLASS TargetPriorityClass;
  unsigned short TargetPriorityRank;
  unsigned short Reserved;
} DFS_TARGET_PRIORITY;
```

TargetPriorityClass: The **DFS_TARGET_PRIORITY_CLASS** enumeration value that specifies the priority class of the target. For more information, see section 2.2.2.8.

TargetPriorityRank: The priority rank of the target, ranging in value from 0x0000 to 0x001F, where 0x0000 is the highest rank. Priority ranks apply only within a priority class, not across priority classes.

Reserved: MUST be set to 0 by the sender and ignored by the receiver.

2.2.2.8 DFS_TARGET_PRIORITY_CLASS

The **DFS_TARGET_PRIORITY_CLASS** enumeration relates to the **NetrDfsSetInfo** and **NetrDfsSetInfo2** methods when used to set the priority of DFS targets in referrals from a server. For more information on prioritization, see section 2.2.2.7. The enumeration defines five possible DFS target priority class settings.

```
typedef  enum _DFS_TARGET_PRIORITY_CLASS
{
  DfsInvalidPriorityClass = -1,
  DfsSiteCostNormalPriorityClass = 0,
  DfsGlobalHighPriorityClass = 1,
  DfsSiteCostHighPriorityClass = 2,
  DfsSiteCostLowPriorityClass = 3,
  DfsGlobalLowPriorityClass = 4
} DFS_TARGET_PRIORITY_CLASS;
```

DfsInvalidPriorityClass: This is not a valid priority class.

DfsSiteCostNormalPriorityClass: The default or "normal" site cost priority class for a DFS target.

DfsGlobalHighPriorityClass: The highest priority class for a DFS target. Targets assigned to this class receive global preference.

DfsSiteCostHighPriorityClass: The highest site cost priority class for a DFS target. Targets assigned to this class receive the highest preference among targets of the same site cost for a given DFS client.

DfsSiteCostLowPriorityClass: The lowest site cost priority class for a DFS target. Targets assigned to this class receive the least preference among targets of the same site cost for a given DFS client.

DfsGlobalLowPriorityClass: The lowest priority class level for a DFS target. Targets assigned to this class receive the least preference globally.

The underlying data type of this enumeration is long integer.

The order of priority classes, from highest to lowest, is as follows:

- DfsGlobalHighPriorityClass
- DfsSiteCostHighPriorityClass
- DfsSiteCostNormalPriorityClass
- DfsSiteCostLowPriorityClass
- DfsGlobalLowPriorityClass

Server targets are initially grouped into global high-priority, normal-priority, and global low-priority classes. The normal-priority class is then subdivided, based on AD DS site cost, into site cost high-priority, site cost normal-priority, and site-cost low-priority classes.

For example, all server targets with a site cost value of 0 are grouped into site cost high-, normal-, and low-priority classes. Then, all server targets with higher site costs are likewise separated into site cost high-, normal-, and low-priority classes. Thus, a server target with a site cost value of 0 and a site cost low-priority class is still ranked higher than a server target with a site cost value of 1 and a site cost high-priority class.

Be aware that the value for a "normal-priority class" is set to 0 even though it is lower in priority than DfsGlobalHighPriorityClass and DfsSiteCostHighPriorityClass. This is the default priority class setting. For added granularity, priority rank can be used to discriminate within a priority class.

2.2.2.9 DFSM_ROOT_LIST

The **DFSM_ROOT_LIST** structure relates to the **NetrDfsAdd2**, **NetrDfsAddFtRoot**, and **NetrDfsSetInfo2** methods when used to add a DFS link or a DFS root target, or to modify the configuration of a domain-based DFS namespace. The structure contains an array of **DFSM_ROOT_LIST_ENTRY** structures, each of which contains information about a DFS root target.

The **DFSM_ROOT_LIST** structure has the following format.

```
typedef struct _DFSM_ROOT_LIST {
  DWORD cEntries;
  [size_is(cEntries)] DFSM_ROOT_LIST_ENTRY Entry[];
} DFSM_ROOT_LIST;
```

cEntries: The number of DFS targets. The value of this member indicates the size of the array in the Entry member.

Entry: An array of **DFSM_ROOT_LIST_ENTRY** structures. Each structure provides information about one DFS target. For more information, see section 2.2.2.10.

2.2.2.10 DFSM_ROOT_LIST_ENTRY

The **DFSM_ROOT_LIST_ENTRY** structure relates to the **NetrDfsAdd2**, **NetrDfsAddFtRoot**, and **NetrDfsSetInfo2** methods when used to add a DFS link or a DFS root target, or to modify the configuration of a domain-based DFS namespace. The structure contains information about a DFS root target.

The **DFSM_ROOT_LIST_ENTRY** structure has the following format.

```
typedef struct _DFSM_ROOT_LIST_ENTRY {
  [string, unique] WCHAR* ServerShare;
} DFSM_ROOT_LIST_ENTRY;
```

ServerShare: Specifies a DFS root target.

2.2.2.11 DFS_NAMESPACE_VERSION_ORIGIN

The **DFS_NAMESPACE_VERSION_ORIGIN** is an enumeration that relates to the **NetrDfsGetSupportedNamespaceVersion** method when used to determine the supported DFS metadata version number.

The **DFS_NAMESPACE_VERSION_ORIGIN** enumeration has the following format.

```
typedef  enum _DFS_NAMESPACE_VERSION_ORIGIN
{
  DFS_NAMESPACE_VERSION_ORIGIN_COMBINED = 0,
  DFS_NAMESPACE_VERSION_ORIGIN_SERVER,
  DFS_NAMESPACE_VERSION_ORIGIN_DOMAIN
} DFS_NAMESPACE_VERSION_ORIGIN;
```

DFS_NAMESPACE_VERSION_ORIGIN_COMBINED: The maximum version that a server and the AD DS domain that it is a member of can support.

DFS_NAMESPACE_VERSION_ORIGIN_SERVER: The maximum version that a server can support.

DFS_NAMESPACE_VERSION_ORIGIN_DOMAIN: The maximum version that the AD DS domain can support.

2.2.2.12 DFS_SUPPORTED_NAMESPACE_VERSION_INFO

The **DFS_SUPPORTED_NAMESPACE_VERSION_INFO** structure relates to the **NetrDfsGetSupportedNamespaceVersion** method when used to determine the domain-based or standalone-based DFS major and minor version information.

The **DFS_SUPPORTED_NAMESPACE_VERSION_INFO** structure has the following format.

```
typedef struct _DFS_SUPPORTED_NAMESPACE_VERSION_INFO {
  unsigned long DomainDfsMajorVersion;
  unsigned long DomainDfsMinorVersion;
  unsigned long long DomainDfsCapabilities;
  unsigned long StandaloneDfsMajorVersion;
  unsigned long StandaloneDfsMinorVersion;
  unsigned long long StandaloneDfsCapabilities;
} DFS_SUPPORTED_NAMESPACE_VERSION_INFO,
 *PDFS_SUPPORTED_NAMESPACE_VERSION_INFO;
```

DomainDfsMajorVersion: A value containing the major version number of the DFS metadata format supported by a domain-based DFS namespace.

DomainDfsMinorVersion: A value containing the minor version number of the DFS metadata format supported by a domain-based DFS namespace.

DomainDfsCapabilities: A value containing the capability information of a domain-based DFS namespace.

StandaloneDfsMajorVersion: A value containing the major version number of a stand-alone DFS namespace.

StandaloneDfsMinorVersion: A value containing the minor version number of a stand-alone DFS namespace.

StandaloneDfsCapabilities: A value containing the capability information of a stand-alone DFS namespace.

2.2.3 Get Info Data Types

The structures in this section relate to the **NetrDfsGetInfo**, **NetrDfsEnum**, and **NetrDfsEnumEx** methods when used to retrieve information about the DFS server configuration. The usage model of these structures is for the client to specify a *Level* parameter to indicate which case of the **DFS_INFO_STRUCT** to use.

2.2.3.1 DFS_INFO_1

The **DFS_INFO_1** structure contains the name of a DFS root or DFS link.

The **DFS_INFO_1** structure has the following format.

```
typedef struct _DFS_INFO_1 {
  [string] WCHAR* EntryPath;
} DFS_INFO_1;
```

EntryPath: The pointer to a DFS root or a DFS link path.

2.2.3.2 DFS_INFO_2

The **DFS_INFO_2** structure contains information for a DFS root or DFS link.

The **DFS_INFO_2** structure has the following format.

```
typedef struct _DFS_INFO_2 {
  [string] WCHAR* EntryPath;
  [string] WCHAR* Comment;
  DWORD State;
  DWORD NumberOfStorages;
} DFS_INFO_2;
```

EntryPath: A pointer to a DFS root or a DFS link path.

Comment: A pointer to a null-terminated Unicode string containing a comment associated with the DFS root or DFS link that is for informational purposes. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality.

State: This field has both the state of the DFS root or DFS link as well as a bit field specifying whether the DFS namespace is stand-alone or domain-based.

The DFS_VOLUME_STATES bitmask (0x0000000F) MUST be used to extract the following DFS root or DFS link state from this field.

Value	Meaning
DFS_VOLUME_STATE_OK 0x00000001	The specified DFS root or DFS link is in the normal state. This state MUST NOT be set to this value.
DFS_VOLUME_STATE_INCONSISTENT 0x00000002	The internal DFS database is inconsistent with the specified DFS root or DFS link. Attempts to repair the inconsistency have failed. This is a read-only state and MUST NOT be set by clients.
DFS_VOLUME_STATE_OFFLINE 0x00000003	The specified DFS root or DFS link is offline or unavailable.
DFS_VOLUME_STATE_ONLINE 0x00000004	The specified DFS root or DFS link is available.

The DFS_VOLUME_FLAVORS bitmask (0x00000300) MUST be used to extract the following DFS namespace flavor from this field.

Value	Meaning
DFS_VOLUME_FLAVOR_STANDALONE 0x00000100	Stand-alone DFS namespace.
DFS_VOLUME_FLAVOR_AD_BLOB 0x00000200	Domainv1-based or **domainv2-based DFS namespace**.

NumberOfStorages: Number of DFS targets for the root or link.

2.2.3.3 DFS_INFO_3

The **DFS_INFO_3** structure contains information for a DFS root or a DFS link.

The **DFS_INFO_3** structure has the following format.

```
typedef struct _DFS_INFO_3 {
  [string] WCHAR* EntryPath;
  [string] WCHAR* Comment;
  DWORD State;
  DWORD NumberOfStorages;
  [size_is(NumberOfStorages)] DFS_STORAGE_INFO* Storage;
} DFS_INFO_3;
```

EntryPath: Pointer to a DFS root or DFS link path.

Comment: A pointer to a null-terminated Unicode string containing a comment associated with the DFS root or DFS link that is for informational purposes. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality.

State: Refers to the State field of **DFS_INFO_2**. For more information, see section 2.2.3.2.

NumberOfStorages: The number of DFS targets for this root or link.

Storage: A pointer to an array of **DFS_STORAGE_INFO** structures containing information about each target. For more information, see section 2.2.2.5). The NumberOfStorages member specifies the number of structures within this storage array.

2.2.3.4 DFS_INFO_4

The **DFS_INFO_4** structure contains information for a DFS root or a DFS link.

The **DFS_INFO_4** structure has the following format.

```
typedef struct _DFS_INFO_4 {
  [string] WCHAR* EntryPath;
  [string] WCHAR* Comment;
  DWORD State;
  unsigned long Timeout;
  GUID Guid;
  DWORD NumberOfStorages;
  [size_is(NumberOfStorages)] DFS_STORAGE_INFO* Storage;
} DFS_INFO_4;
```

EntryPath: A pointer to a DFS root or a DFS link path.

Comment: A pointer to a null-terminated Unicode string containing a comment associated with the DFS root or DFS link that is for informational purposes. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality.

State: Refers to the State field of **DFS_INFO_2**. For more information, see section 2.2.3.2.

Timeout: The time-out, in seconds, associated with the root or link and used in a DFS referral response to a DFS client.

Guid: The GUID of this root or link.

NumberOfStorages: The number of DFS targets for this root or link. There are no protocol-specified restrictions on the number of targets for a root or link.

Storage: A pointer to an array of **DFS_STORAGE_INFO** structures containing information about each target. For more information, see section 2.2.2.5). The NumberOfStorages member specifies the number of structures within this storage array.

2.2.3.5 DFS_INFO_5

The **DFS_INFO_5** structure contains information for a DFS root or a DFS link.

The **DFS_INFO_5** structure has the following format.

```
typedef struct _DFS_INFO_5 {
  [string] WCHAR* EntryPath;
  [string] WCHAR* Comment;
  DWORD State;
  unsigned long Timeout;
  GUID Guid;
  unsigned long PropertyFlags;
  unsigned long MetadataSize;
  DWORD NumberOfStorages;
} DFS_INFO_5;
```

EntryPath: A pointer to a DFS root or a DFS link path.

Comment: A pointer to a null-terminated Unicode string containing a comment associated with the DFS root or DFS link that is for informational purposes. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality.

State: Refers to the State field of **DFS_INFO_2**. For more information, see section 2.2.3.2.

Timeout: The time-out, in seconds, associated with the root or link and used in a DFS referral response to a DFS client.

Guid: The GUID of this root or link.

PropertyFlags: A bit field in which each bit is responsible for a specific property applicable to the entire DFS namespace, the DFS root, or an individual DFS link, depending on the actual property. Any combination of bits is allowed, unless indicated otherwise. The following are valid bit definitions for this field.

Value	Meaning
DFS_PROPERTY_FLAG_INSITE_REFERRALS 0x00000001	When set, indicates that **DFS in-site referral mode** is enabled.
DFS_PROPERTY_FLAG_ROOT_SCALABILITY 0x00000002	When set, indicates DFS root scalability mode is enabled. This flag is valid only for the DFS root of a domain-based DFS namespace.
DFS_PROPERTY_FLAG_SITE_COSTING 0x00000004	When set, indicates **DFS referral site costing** is enabled. This flag is valid only for a DFS root.

Value	Meaning
DFS_PROPERTY_FLAG_TARGET_FAILBACK 0x00000008	When set, indicates **DFS client target failback** is enabled.
DFS_PROPERTY_FLAG_CLUSTER_ENABLED 0x00000010	When set, indicates **clustered DFS namespace** is enabled.
DFS_PROPERTY_FLAG_ABDE 0x00000020	When set, enables **Access Based Directory Enumeration (ABDE) mode** on a domainv2-based DFS namespace or a stand-alone DFS namespace.<5>

MetadataSize: The size, in bytes, of the DFS metadata of the DFS namespace. For a DFS link, this MUST be 0.

NumberOfStorages: the number of DFS targets for this root or link.

2.2.3.6 DFS_INFO_6

The **DFS_INFO_6** structure contains information for a DFS root or a DFS link.

The **DFS_INFO_6** structure has the following format.

```
typedef struct _DFS_INFO_6 {
  [string] WCHAR* EntryPath;
  [string] WCHAR* Comment;
  DWORD State;
  unsigned long Timeout;
  GUID Guid;
  unsigned long PropertyFlags;
  unsigned long MetadataSize;
  DWORD NumberOfStorages;
  [size_is(NumberOfStorages)] DFS_STORAGE_INFO_1* Storage;
} DFS_INFO_6;
```

EntryPath: A pointer to a DFS root or a DFS link path.

Comment: A pointer to a null-terminated Unicode string containing a comment associated with the DFS root or DFS link that is for informational purposes. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality.

State: Refers to the State field of **DFS_INFO_2**. For more information, see section 2.2.3.2.

Timeout: The time-out, in seconds, associated with the root or link and used in a DFS referral response to a DFS client.

Guid: The GUID of this root or link.

PropertyFlags: Refers to the PropertyFlags field of **DFS_INFO_5**. For more information, see section 2.2.3.5.

MetadataSize: The size of the DFS metadata of the DFS namespace. This MUST be 0 for a DFS link.

NumberOfStorages: The number of DFS targets for this root or link. The protocol imposes no restrictions on the number of roots or links.

Storage: A pointer to an array of **DFS_STORAGE_INFO_1** structures containing information about each target. The NumberOfStorages member specifies the number of structures within this storage array.<6>

2.2.3.7 DFS_INFO_7

The **DFS_INFO_7** structure contains information about a DFS root or a DFS link.

The **DFS_INFO_7** structure has the following format.

```
typedef struct _DFS_INFO_7 {
  GUID GenerationGuid;
} DFS_INFO_7;
```

GenerationGuid: This GUID is modified each time DFS metadata is updated.

This data type is used to detect when the metadata of a DFS namespace has changed. It MUST be supported for domain-based DFS namespaces. It MAY be supported for stand-alone DFS namespaces; a null GUID (all 128-bits are 0) MUST be returned if this is not supported.<7>

2.2.3.8 DFS_INFO_8

The **DFS_INFO_8** structure contains information for a DFS root or a DFS link.

The **DFS_INFO_8** structure has the following format.

```
typedef struct _DFS_INFO_8 {
  [string] WCHAR* EntryPath;
  [string] WCHAR* Comment;
  DWORD State;
  unsigned long Timeout;
  GUID Guid;
  unsigned long PropertyFlags;
  unsigned long MetadataSize;
  PSECURITY_DESCRIPTOR pSecurityDescriptor;
  DWORD NumberOfStorages;
} DFS_INFO_8,
 *LPDFS_INFO_8;
```

EntryPath: A pointer to a DFS root or a DFS link path.

Comment: A pointer to a null-terminated Unicode string containing a comment associated with the DFS root or DFS link that is for informational purposes. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality.

State: Refers to the State field of **DFS_INFO_2**. For more information, see section 2.2.3.2.

Timeout: The time-out, in seconds, associated with the root or link and used in a DFS referral response to a DFS client.

Guid: The GUID of this root or link.

PropertyFlags: Refers to the PropertyFlags field of **DFS_INFO_5**. For more information, see section 2.2.3.5.

MetadataSize: The size, in bytes, of the DFS metadata of the DFS namespace. For a DFS link, this MUST be 0.

pSecurityDescriptor: A self-relative security descriptor to be associated with a DFS link. For more information on security descriptors, see [MS-DTYP] section **2.4.6**.

NumberOfStorages: The number of DFS targets for this root or link. The protocol imposes no restrictions on the number of roots or links.

2.2.3.9 DFS_INFO_9

The **DFS_INFO_9** structure contains information for a DFS root or a DFS link.

The **DFS_INFO_9** structure has the following format.

```
typedef struct _DFS_INFO_9 {
  [string] WCHAR* EntryPath;
  [string] WCHAR* Comment;
  DWORD State;
  unsigned long Timeout;
  GUID Guid;
  unsigned long PropertyFlags;
  unsigned long MetadataSize;
  PSECURITY_DESCRIPTOR pSecurityDescriptor;
  DWORD NumberOfStorages;
  [size_is(NumberOfStorages)] DFS_STORAGE_INFO_1* Storage;
} DFS_INFO_9,
 *LPDFS_INFO_9;
```

EntryPath: A pointer to a DFS root or a DFS link path.

Comment: Pointer to a null-terminated Unicode string containing a comment associated with the DFS root or DFS link that is for informational purposes. There are no protocol-specified restrictions on the length or content of this string. The comment is meant for human consumption and has no effect on server functionality.

State: Refers to the State field of **DFS_INFO_2**. For more information, see section 2.2.3.2.

Timeout: The time-out, in seconds, associated with the root or link and used in a DFS referral response to a DFS client.

Guid: The GUID of this root or link.

PropertyFlags: Refers to the PropertyFlags field of **DFS_INFO_5**. For more information, see section 2.2.3.5.

MetadataSize: The size, in bytes, of the DFS metadata of the DFS namespace. For a DFS link, this MUST be 0.

pSecurityDescriptor: A self-relative security descriptor to be associated with DFS link. For more information on security descriptors, see [MS-DTYP] section **2.4.6**.

NumberOfStorages: The number of DFS targets for this root or link. The protocol imposes no restrictions on the number of roots or links.

Storage: A pointer to an array of **DFS_STORAGE_INFO_1** structures containing information about each target. The NumberOfStorages member specifies the number of structures within this storage array.<8>

For information on target priority rank and class information, see 2.2.2.6.

2.2.3.10 DFS_INFO_50

The **DFS_INFO_50** structure is used to get the DFS metadata version and the capability information of an existing DFS namespace.

The **DFS_INFO_50** structure has the following format.

```
typedef struct _DFS_INFO_50 {
  unsigned long NamespaceMajorVersion;
  unsigned long NamespaceMinorVersion;
  unsigned __int64 NamespaceCapabilities;
} DFS_INFO_50;
```

NamespaceMajorVersion: A value containing the major version number used to determine the DFS metadata format supported in a domain-based DFS namespace or a stand-alone DFS namespace.<9>

NamespaceMinorVersion: Value containing the minor version number used to determine the DFS metadata format supported in a domain-based DFS namespace or stand-alone DFS namespace.<10>

NamespaceCapabilities: A value containing the capability information of a DFS namespace.

Value	Meaning
DFS__NAMESPACE_CAPABILITY_ABDE 0x0000000000000001	The specified DFS root or DFS link that supports the ABDE flag.<11>

2.2.4 Set Info Data Types

The structures in this section relate to the **NetrDfsSetInfo** and **NetrDfsSetInfo2** methods when used to retrieve or set the configuration of the DFS server. The usage model of these structures is for the client to specify a *Level* parameter to indicate which **DFS_INFO_STRUCT** case to use.

2.2.4.1 DFS_INFO_101

The **DFS_INFO_101** structure describes the storage state on a DFS root, DFS link, DFS root target, or DFS link target.

The **DFS_INFO_101** structure has the following format.

```
typedef struct _DFS_INFO_101 {
  unsigned long State;
} DFS_INFO_101;
```

State: The state of the root, link, root target, or link target.

The following table lists the valid states that can be set for a DFS root or a DFS link.

Value	Meaning
DFS_VOLUME_STATE_OK 0x00000001	The specified DFS root or DFS link is in the normal state. This state MUST NOT be set to this value.
DFS_VOLUME_STATE_OFFLINE 0x00000003	The specified DFS root or DFS link is offline or unavailable.
DFS_VOLUME_STATE_ONLINE 0x00000004	The specified DFS root or DFS link is available.
DFS_VOLUME_STATE_RESYNCHRONIZE 0x00000010	Forces a resynchronization on the DFS root. This flag is valid only for a DFS root. This operation is an incremental synchronization that picks up only changed objects in the metadata.
DFS_VOLUME_STATE_STANDBY 0x00000020	Sets a root volume to standby mode. This flag is valid only for a clustered DFS root.
FS_VOLUME_STATE_FORCE_SYNC 0x00000040	Forces a full resynchronization operation on the DFS root target of a specified domainv2-based DFS namespace or stand-alone DFS namespace to identify DFS links that have been added or deleted. This is not supported on a domainv1-based DFS namespace. DFS links MUST NOT be specified.

Clients MUST NOT set the state to DFS_VOLUME_STATE_INCONSISTENT (0x00000002) because this is a read-only state set by the server. DFS_VOLUME_STATES (0x0000000F) is not relevant here, because it is a mask used when reading the volume state, not for setting it.

The following table lists the valid states that can be set for a DFS root target or a DFS link target.

Value	Meaning
DFS_STORAGE_STATE_OFFLINE 0x00000001	The DFS storage is offline.
DFS_STORAGE_STATE_ONLINE 0x00000002	The DFS storage is online.

2.2.4.2 DFS_INFO_102

The **DFS_INFO_102** structure contains a time-out value for a DFS root or a DFS link.

The **DFS_INFO_102** structure has the following format.

```
typedef struct _DFS_INFO_102 {
  unsigned long Timeout;
} DFS_INFO_102;
```

Timeout: The time-out, in seconds, associated with the root or link and used in a DFS referral response to a DFS client.

2.2.4.3 DFS_INFO_103

The **DFS_INFO_103** structure contains properties that set specific behaviors for a DFS root or a DFS link.

The **DFS_INFO_103** structure has the following format.

```
typedef struct _DFS_INFO_103 {
  unsigned long PropertyFlagMask;
  unsigned long PropertyFlags;
} DFS_INFO_103;
```

PropertyFlagMask: Indicates which bits in the **PropertyFlags** field are valid.

PropertyFlags: A bit field in which each bit is responsible for a specific property applicable to the whole DFS namespace, the DFS root, or an individual DFS link, depending on the actual property. Any combination of bits is allowed, unless indicated otherwise. The following are valid bit definitions for this field.

Value	Meaning
DFS_PROPERTY_FLAG_INSITE_REFERRALS 0x00000001	When set, enables **DFS in-site referral mode**. Valid for domain and stand-alone DFS roots and links.
DFS_PROPERTY_FLAG_ROOT_SCALABILITY 0x00000002	When set, enables DFS root scalability mode. This flag is valid only for the DFS root of a domain-based DFS namespace.
DFS_PROPERTY_FLAG_SITE_COSTING 0x00000004	When set, enables DFS referral site costing. This flag is valid only for a DFS root.
DFS_PROPERTY_FLAG_TARGET_FAILBACK 0x00000008	When set, enables **DFS client target failback**. Valid for domain and stand-alone DFS roots and links.
DFS_PROPERTY_FLAG_CLUSTER_ENABLED 0x00000010	When set, indicates a clustered DFS namespace.
DFS_PROPERTY_FLAG_ABDE 0x00000020	When set, enables ABDE mode on a domainv2-based DFS namespace or stand-alone DFS namespace.<12>

2.2.4.4 DFS_INFO_104

The **DFS_INFO_104** structure contains the priority of a DFS root target or a DFS link target.

The **DFS_INFO_104** structure has the following format.

```
typedef struct _DFS_INFO_104 {
  DFS_TARGET_PRIORITY TargetPriority;
} DFS_INFO_104;
```

TargetPriority: A **DFS_TARGET_PRIORITY** structure that indicates the priority rank and priority class of a target. For more information on prioritization, see section 2.2.2.7.

2.2.4.5 DFS_INFO_105

The **DFS_INFO_105** structure contains information about a DFS root or DFS link, including comment, state, time-out, and DFS behaviors specified by property flags.

The **DFS_INFO_105** structure has the following format.

```
typedef struct _DFS_INFO_105 {
  [string] WCHAR* Comment;
  DWORD State;
  unsigned long Timeout;
  unsigned long PropertyFlagMask;
  unsigned long PropertyFlags;
} DFS_INFO_105;
```

Comment: A pointer to a null-terminated Unicode string containing a comment associated with the DFS root or DFS link that is for informational purposes. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality.

State: The following table lists the valid states that can be set for the root or link.

Value	Meaning
0x00000000	Indicates that the existing state MUST NOT be changed.
DFS_VOLUME_STATE_OK 0x00000001	The specified DFS root or DFS link is in the normal state. This state MUST NOT be set to this value.
DFS_VOLUME_STATE_OFFLINE 0x00000003	The specified DFS root or link is offline or unavailable.
DFS_VOLUME_STATE_ONLINE 0x00000004	The specified DFS root or link is available.

Timeout: The time-out, in seconds, associated with the root or link and used in a DFS referral response to a DFS client.

PropertyFlagMask: Indicates which bits in the **PropertyFlags** field are valid.

PropertyFlags: Refers to the **PropertyFlags** field of **DFS_INFO_103**, as specified in section 2.2.4.3.

2.2.4.6 DFS_INFO_106

The **DFS_INFO_106** structure contains the storage state and priority of a DFS root target or a DFS link target. For more information on prioritization, see section 2.2.2.7.

The **DFS_INFO_106** structure has the following format.

```
typedef struct _DFS_INFO_106 {
  DWORD State;
  DFS_TARGET_PRIORITY TargetPriority;
} DFS_INFO_106;
```

State: The state of the target. Contains one of the following valid state values.

Value	Meaning
DFS_STORAGE_STATE_OFFLINE 0x00000001	This target is offline and unavailable for use.
DFS_STORAGE_STATE_ONLINE 0x00000002	This target is online and available for use.

TargetPriority: A **DFS_TARGET_PRIORITY** structure that indicates the priority class and rank of the DFS target.

2.2.4.7 DFS_INFO_107

The **DFS_INFO_107** structure contains information about a DFS link, including comment, state, time-out, security descriptor, and DFS behaviors specified by property flags.

The **DFS_INFO_107** structure has the following format.

```
typedef struct _DFS_INFO_107 {
  [string] WCHAR* Comment;
  DWORD State;
  unsigned long Timeout;
  unsigned long PropertyFlagMask;
  unsigned long PropertyFlags;
  PSECURITY_DESCRIPTOR pSecurityDescriptor;
} DFS_INFO_107;
```

Comment: A pointer to a null-terminated Unicode string containing a comment associated with the DFS root or DFS link that is for informational purposes. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality.

State: The state of the root or link. The following table lists the valid states that can be set.

Value	Meaning
0x00000000	Indicates that the existing state MUST NOT be changed.
DFS_VOLUME_STATE_OK 0x00000001	The specified DFS root or DFS link is in the normal state. This state MUST NOT be set to this value.
DFS_VOLUME_STATE_OFFLINE 0x00000003	The specified DFS root or DFS link is offline or unavailable.
DFS_VOLUME_STATE_ONLINE 0x00000004	The specified DFS root or DFS link is available.

Timeout: The time-out, in seconds, associated with the root or link and used in a DFS referral response to a DFS client.

PropertyFlagMask: Indicates which bits in the **PropertyFlags** field are valid.

PropertyFlags: Refers to the **PropertyFlags** field of **DFS_INFO_3**, as specified in section 2.2.4.3.

pSecurityDescriptor: A pointer to a self-relative security descriptor associated with DFS. For more information on security descriptors, see [MS-DTYP] section **2.4.6**.

2.2.5 Special Info Data Types

The structures in this section relate to the **NetrDfsEnum**, **NetrDfsEnumEx**, **NetrDfsGetInfo**, **NetrDfsSetInfo**, and **NetrDfsSetInfo2** methods when used to retrieve or set the DFS server configuration. The usage model of these structures is for the client to specify a *Level* parameter to indicate which case of the **DFS_INFO_STRUCT** to use.

2.2.5.1 DFS_INFO_100

The **DFS_INFO_100** structure relates to the **NetrDfsGetInfo**, **NetrDfsSetInfo**, and **NetrDfsSetInfo2** methods when used to retrieve or set comment text about a DFS root or a DFS link. The structure contains a comment associated with a DFS root or a DFS link.

The **DFS_INFO_100** structure has the following format.

```
typedef struct _DFS_INFO_100 {
  [string] WCHAR* Comment;
} DFS_INFO_100;
```

Comment: A pointer to a null-terminated Unicode string containing a comment associated with the DFS root or DFS link that is for informational purposes. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality.

2.2.5.2 DFS_INFO_150

The **DFS_INFO_150** structure relates to the **NetrDfsGetInfo**, **NetrDfsSetInfo**, and **NetrDfsSetInfo2** methods when used to retrieve or set security descriptors associated with a DFS link. The structure contains the self-relative security descriptor associated with a DFS link.

The **DFS_INFO_150** structure has the following format.

```
typedef struct _DFS_INFO_150 {
  unsigned long SdLengthReserved;
  PSECURITY_DESCRIPTOR pSecurityDescriptor;
} DFS_INFO_150;
```

SdLengthReserved: The length, in bytes, of the buffer that the **pSecurityDescriptor** field points to.

pSecurityDescriptor: A self-relative security descriptor associated with DFS. For more information on security descriptors, see [MS-DTYP] section **2.4.6**.

2.2.5.3 DFS_INFO_200

The **DFS_INFO_200** structure relates to the **NetrDfsEnumEx** method when used to enumerate all of the domain-based DFS namespace in a domain. The structure contains the name of a domain-based DFS namespace. The **DFS_INFO_200** structure has the following format.

```
typedef struct _DFS_INFO_200 {
  [string] WCHAR* FtDfsName;
} DFS_INFO_200;
```

FtDfsName: A pointer to a DFS root path.

2.2.5.4 DFS_INFO_300

The **DFS_INFO_300** structure relates to the **NetrDfsEnum** and **NetrDfsEnumEx** methods when used to enumerate DFS roots hosted on a server. The structure contains the name and type (domain-based or stand-alone) of a DFS namespace. The **DFS_INFO_300** structure has the following format.

```
typedef struct _DFS_INFO_300 {
  DWORD Flags;
  [string] WCHAR* DfsName;
} DFS_INFO_300;
```

Flags: This value specifies the type of the DFS namespace. This MUST have one of the following two permitted values.

Value	Meaning
DFS_VOLUME_FLAVOR_STANDALONE 0x00000100	Stand-alone DFS namespace.
DFS_VOLUME_FLAVOR_AD_BLOB 0x00000200	Domain-based DFS namespace.

DfsName: A pointer to a DFS root path.

2.2.6 Enum Info Data Types

The structures in this section relate to the **NetrDfsEnum** and **NetrDfsEnumEx** methods when used to enumerate and retrieve the configuration of the DFS server. The usage model of these structures is for the client to specify a *Level* parameter to indicate which case of the **DFS_INFO_ENUM_STRUCT** to use.

2.2.6.1 DFS_INFO_1_CONTAINER

The **DFS_INFO_1_CONTAINER** structure contains an array of **DFS_INFO_1** structures. The **DFS_INFO_1_CONTAINER** structure has the following format.

```
typedef struct _DFS_INFO_1_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] DFS_INFO_1* Buffer;
} DFS_INFO_1_CONTAINER;
```

EntriesRead: The number of elements in the array.

Buffer: The array of **DFS_INFO_1** structures.

2.2.6.2 DFS_INFO_2_CONTAINER

The **DFS_INFO_2_CONTAINER** structure contains an array of **DFS_INFO_2** structures. The **DFS_INFO_2_CONTAINER** structure has the following format.

```
typedef struct _DFS_INFO_2_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] DFS_INFO_2* Buffer;
} DFS_INFO_2_CONTAINER;
```

EntriesRead: The number of elements in the array.

Buffer: The array of **DFS_INFO_2** structures.

2.2.6.3 DFS_INFO_3_CONTAINER

The **DFS_INFO_3_CONTAINER** structure contains an array of **DFS_INFO_3** structures. The **DFS_INFO_3_CONTAINER** structure has the following format.

```
typedef struct _DFS_INFO_3_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] DFS_INFO_3* Buffer;
} DFS_INFO_3_CONTAINER;
```

EntriesRead: The number of elements in the array.

Buffer: The array of **DFS_INFO_3** structures.

2.2.6.4 DFS_INFO_4_CONTAINER

The **DFS_INFO_4_CONTAINER** structure contains an array of **DFS_INFO_4** structures. The **DFS_INFO_4_CONTAINER** structure has the following format.

```
typedef struct _DFS_INFO_4_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] DFS_INFO_4* Buffer;
} DFS_INFO_4_CONTAINER;
```

EntriesRead: The number of elements in the array.

Buffer: The array of **DFS_INFO_4** structures.

2.2.6.5 DFS_INFO_5_CONTAINER

The **DFS_INFO_5_CONTAINER** structure contains an array of **DFS_INFO_5** structures. The **DFS_INFO_5_CONTAINER** structure has the following format.

```
typedef struct _DFS_INFO_5_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] DFS_INFO_5* Buffer;
} DFS_INFO_5_CONTAINER;
```

EntriesRead: The number of elements in the array.

Buffer: The array of **DFS_INFO_5** structures.

2.2.6.6 DFS_INFO_6_CONTAINER

The **DFS_INFO_6_CONTAINER** structure contains an array of **DFS_INFO_6** structures. The **DFS_INFO_6_CONTAINER** structure has the following format.

```
typedef struct _DFS_INFO_6_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] DFS_INFO_6* Buffer;
} DFS_INFO_6_CONTAINER;
```

EntriesRead: The number of elements in the array.

Buffer: The array of **DFS_INFO_6** structures.

2.2.6.7 DFS_INFO_8_CONTAINER

The **DFS_INFO_8_CONTAINER** structure contains an array of **DFS_INFO_8** structures. The **DFS_INFO_8_CONTAINER** structure has the following format.

```
typedef struct _DFS_INFO_8_CONTAINER {
  DWORD EntriesRead;
```

```
  [size_is(EntriesRead)] LPDFS_INFO_8 Buffer;
} DFS_INFO_8_CONTAINER,
 *LPDFS_INFO_8_CONTAINER;
```

EntriesRead: The number of **DFS_INFO_8** elements in the array.

Buffer: The array of **DFS_INFO_8** structures.

2.2.6.8 DFS_INFO_9_CONTAINER

The **DFS_INFO_9_CONTAINER** structure contains an array of **DFS_INFO_9** structures. The **DFS_INFO_9_CONTAINER** structure has the following format.

```
typedef struct _DFS_INFO_9_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPDFS_INFO_9 Buffer;
} DFS_INFO_9_CONTAINER,
 *LPDFS_INFO_9_CONTAINER;
```

EntriesRead: The number of **DFS_INFO_9** elements in the array.

Buffer: The array of **DFS_INFO_9** structures.

2.2.6.9 DFS_INFO_200_CONTAINER

The **DFS_INFO_200_CONTAINER** structure contains an array of **DFS_INFO_200** structures. The **DFS_INFO_200_CONTAINER** structure has the following format.

```
typedef struct _DFS_INFO_200_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] DFS_INFO_200* Buffer;
} DFS_INFO_200_CONTAINER;
```

EntriesRead: The number of elements in the array.

Buffer: The array of **DFS_INFO_200** structures.

2.2.6.10 DFS_INFO_300_CONTAINER

The **DFS_INFO_300_CONTAINER** structure contains an array of **DFS_INFO_300** structures. The **DFS_INFO_300_CONTAINER** structure has the following format.

```
typedef struct _DFS_INFO_300_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] DFS_INFO_300* Buffer;
} DFS_INFO_300_CONTAINER;
```

EntriesRead: The number of elements in the array.

Buffer: The array of **DFS_INFO_300** structures.

2.3 Directory Service Syntax

This section contains specifications for the DFS configuration container, DFS namespace object, and pKT attribute.

2.3.1 DFS Configuration Container

The DFS configuration container is a well-known container in the domain directory that is used to hold the DFS metadata for a domain-based DFS namespace. The container has the following **distinguished name (DN)**.

```
CN=Dfs-Configuration,CN=System,<domain>
```

where <domain> is the DN of the AD DS domain.

For example, the DFS configuration container for the contoso.com domain would be named:

```
CN=Dfs-Configuration,CN=System,DC=contoso,DC=com
```

The object class of this AD DS object is dfsConfiguration, and its schema is as specified in [MS-ADSC].

2.3.2 LDAP Entries for Domain-Based DFS Namespaces

LDAP entries exist in both domainv1-based DFS namespace and a domainv2-based DFS namespace. This relationship is shown in the following figure.

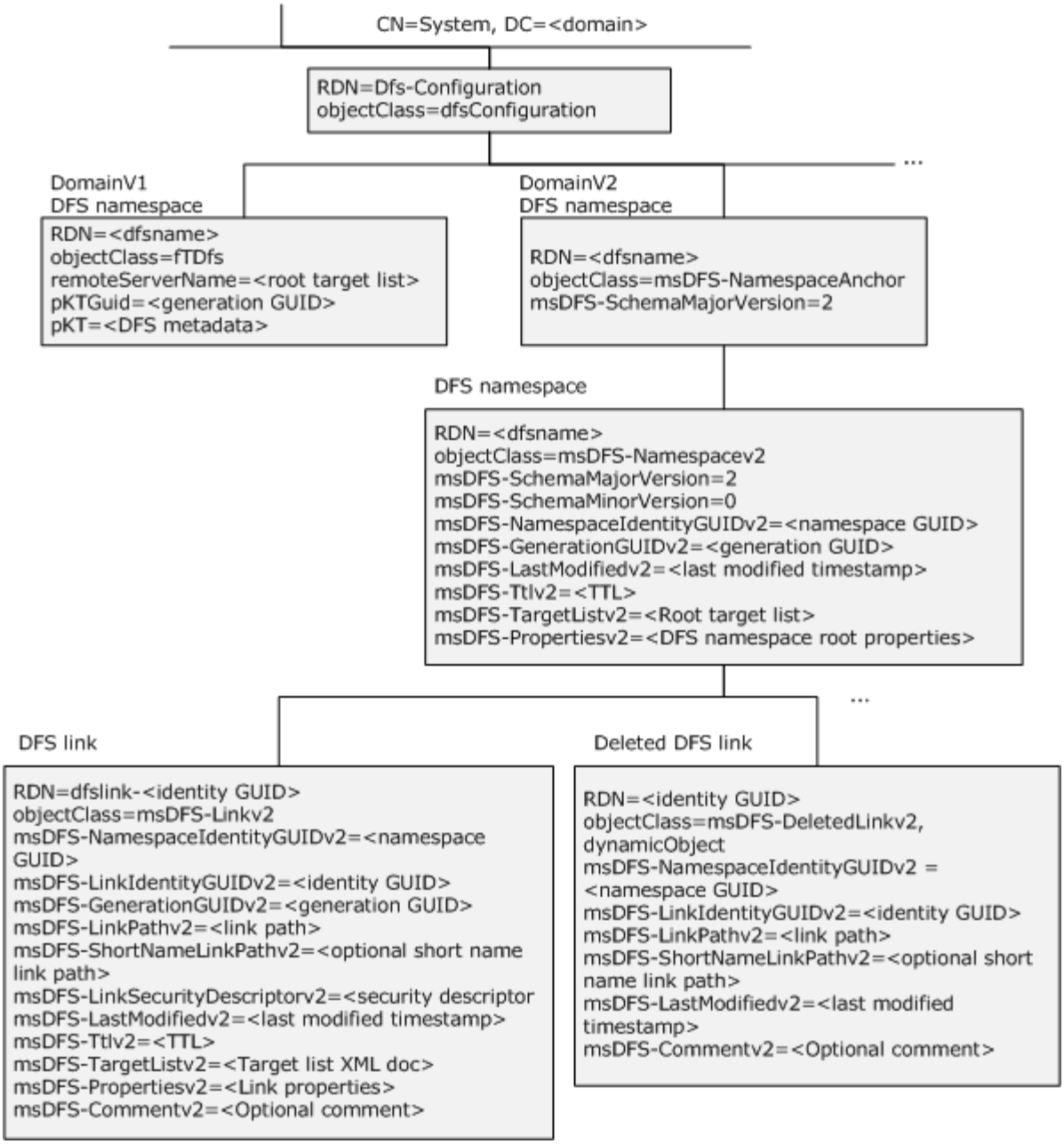


Figure 1: Organization of DFS-related LDAP entries in DFS namespaces

Each domainv1-based DFS namespace has its DFS metadata stored in **AD** as a BLOB in the **pKT** attribute of an LDAP entry.

Each domainv2-based DFS namespace has one DFS namespace anchor LDAP entry, one DFS namespace LDAP entry below it, and one LDAP entry per DFS link in the namespace under the DFS namespace LDAP entry.

Domainv2-based LDAP entries have different objectClasses from domainv1-based ones. This ensures that a domainv2-based DFS namespace is not confused with a domainv1-based DFS namespace by an operating system earlier than Windows Vista.<13>

The following sections specify both domain-based DFS namespace formats.

2.3.3 DFS Namespace AD DS Object for Domainv1-Based DFS Namespace

An AD DS object exists for each domainv1-based DFS namespace in the DFS configuration container. The following is a DN of the AD DS object of a domain-based DFS namespace.

```
CN=<DFSNamespaceName>,CN=Dfs-Configuration,CN=System,<domain>
```

where <DFSNamespaceName> is the domain-based DFS namespace and <domain> is the DN of the AD DS domain.

The following are attributes that apply to this AD DS object.

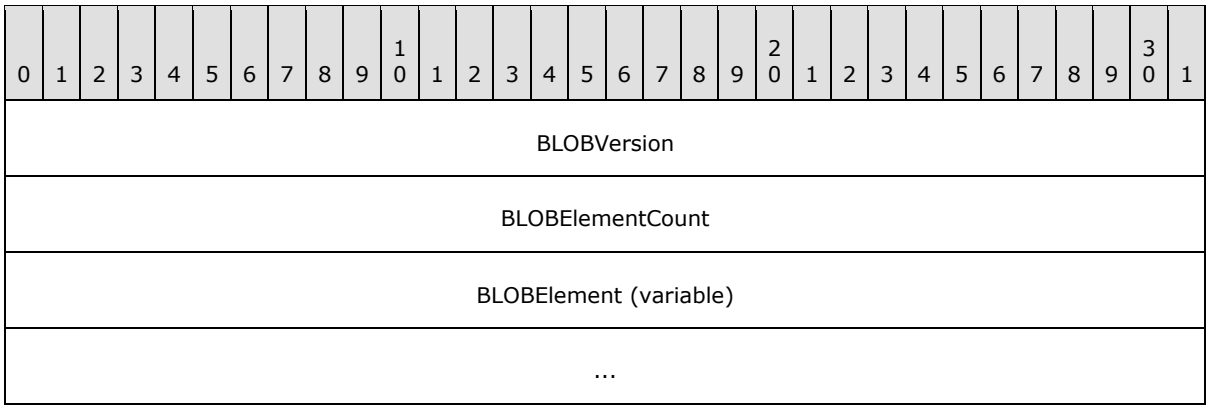
Attribute	Description
name	The **DFS namespace name**.
remoteServerName	A multivalued attribute that contains the DFS root targets for the DFS namespace with the value "*" as the last attribute.
pKTGuid	A GUID used as a generation number to detect changes to the DFS metadata. This MUST be updated whenever the **pKT** attribute is changed.
pKT	The BLOB containing the DFS metadata.

The objectClass of this AD DS object is fTDfs, and its schema is specified in [MS-ADSC]. The schema of the attributes name, remoteServerName, pKTGuid, and pKT are specified in [MS-ADA3].

2.3.3.1 pKT Attribute Contents (Metadata for Domainv1-Based Namespace)

The **pKT** attribute contains the DFS metadata of the domain-based DFS namespace that the AD DS object represents.

The DFS metadata has the following format.



BLOBVersion: The DFS metadata format version stored as an unsigned, 32-bit, **little-endian** integer. This MUST always be set to 0.

BLOBElementCount: The number of <BLOBElement> elements that immediately follow this field in the DFS metadata stored as an unsigned, 32-bit, little-endian integer.

BLOBElement: A variable number of DFSNamespaceElementBLOB structures, which immediately follow the BLOBElementCount. Each DFSNamespaceElementBLOB contains descriptive information about a DFS site, root, or link. The format and size of each DFSNamespaceElementBLOB depend on the information contained in it.

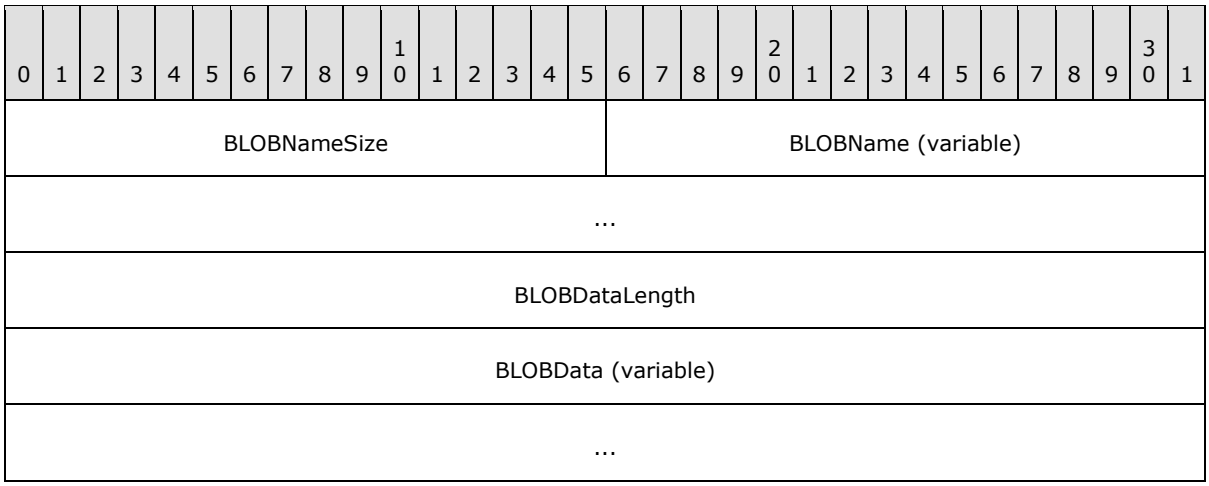
2.3.3.1.1 DFSNamespaceElementBLOB

A DFSNamespaceElementBLOB contains information about a DFS root or a DFS link, or for mapping a server to its **AD DS site** name.

The DFS metadata of a valid DFS namespace MUST consist of one DFSNamespaceRootBLOB for the DFS root and one DFSNamespaceLinkBLOB for each DFS link in the DFS namespace. There MAY be, at most, one SiteInformationBLOB. For more information on SiteInformationBLOB, see section 2.3.3.1.1.4.

No alignment padding requirements exist for any of a BLOB's fields, unless otherwise specified.

Each DFSNamespaceElementBLOB contains the following data elements. The first three fields are standard for all DFSNamespaceElementBLOB structures. Following those fields are additional fields that are specific to the type of DFSNamespaceElementBLOB, in the format of a DFSNamespaceRootBLOB, a DFSNamespaceLinkBLOB, or a SiteInformationBLOB.



BLOBNameSize: The size of the **BLOBName**, in bytes, stored as an unsigned, 16-bit, little-endian integer.

BLOBName: The name of the DFSNamespaceElementBLOB, stored as a string of Unicode characters.

Value	Meaning
SiteInformationBLOB "\siteroot"	A string of Unicode characters that forms the literal "\siteroot".<14>
DFSNamespaceRootBLOB "\domainroot"	A string of Unicode characters that forms the literal "\domainroot".
DFSNamespaceLinkBLOB "\domainroot\<GUIDString>"	A string of Unicode characters that forms the literal "\domainroot\<GUIDString>", where <GUIDString> represents the string form of a GUID, as specified in [RFC4122], section 3. The GUID found in the link's ID BLOB MUST be used to create this.

BLOBDataLength: The length of the BLOB in the **BLOBData** field, stored as an unsigned, 32-bit, little-endian integer. The value of this field MUST be used to determine the start of the next DFSNamespaceElementBLOB.

BLOBData: Data specific to the type of BLOB described, in the form of a DFSNamespaceRootBLOB, DFSNamespaceLinkBLOB, or SiteInformationBLOB.

The following sections specify the format of the DFSNamespaceRootBLOB, DFSNamespaceLinkBLOB, and SiteInformationBLOB.

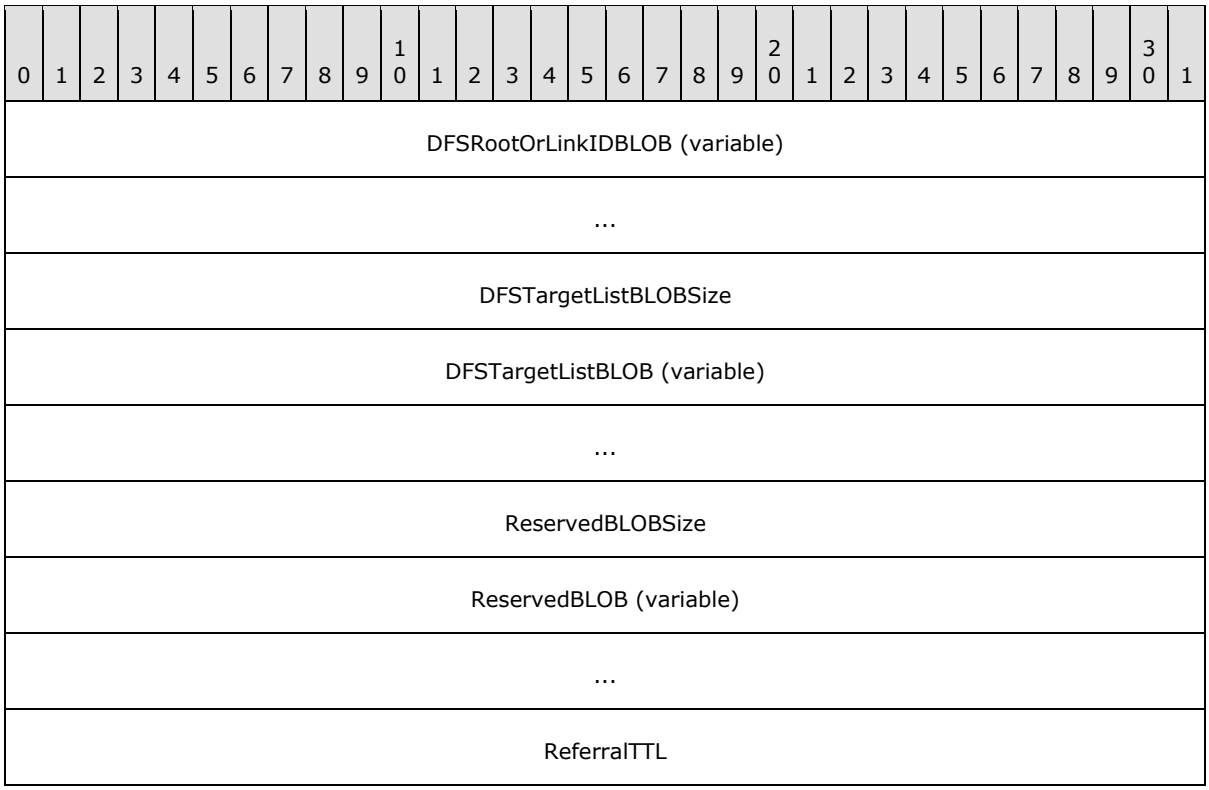
2.3.3.1.1.1 DFSNamespaceRootBLOB or DFSNamespaceLinkBLOB

At most, only one DFSNamespaceRootBLOB can contain information about the DFS namespace root. One DFSNamespaceLinkBLOB exists for each DFS link in the namespace.

Each DFSNamespaceRootBLOB or DFSNamespaceLinkBLOB MUST have the following:

- One BLOB containing the name and other information about the DFS namespace root or DFS link. This is the DFSRootOrLinkIDBLOB, as specified in section 2.3.3.1.1.2.

- One BLOB containing the DFS targets of the DFS root or DFS link. This is the DFSTargetListBLOB, as specified in section 2.3.3.1.1.3.
- One reserved BLOB.



DFSRootOrLinkIDBLOB: A BLOB that contains identification and status information for this DFS root or DFS link.

DFSTargetListBLOBSize: The size, in bytes, of the BLOB in the **TargetListBLOB** field that immediately follows this field, and stored as an unsigned, 32-bit, little-endian integer.

DFSTargetListBLOB: A BLOB that contains the list of targets for the DFS root or DFS link.

ReservedBLOBSize: The size, in bytes, of the BLOB in the **ReservedBLOB** field that immediately follows this field. The **ReservedBLOBSize** is stored as an unsigned, 32-bit, little-endian integer and MUST be at least 4.

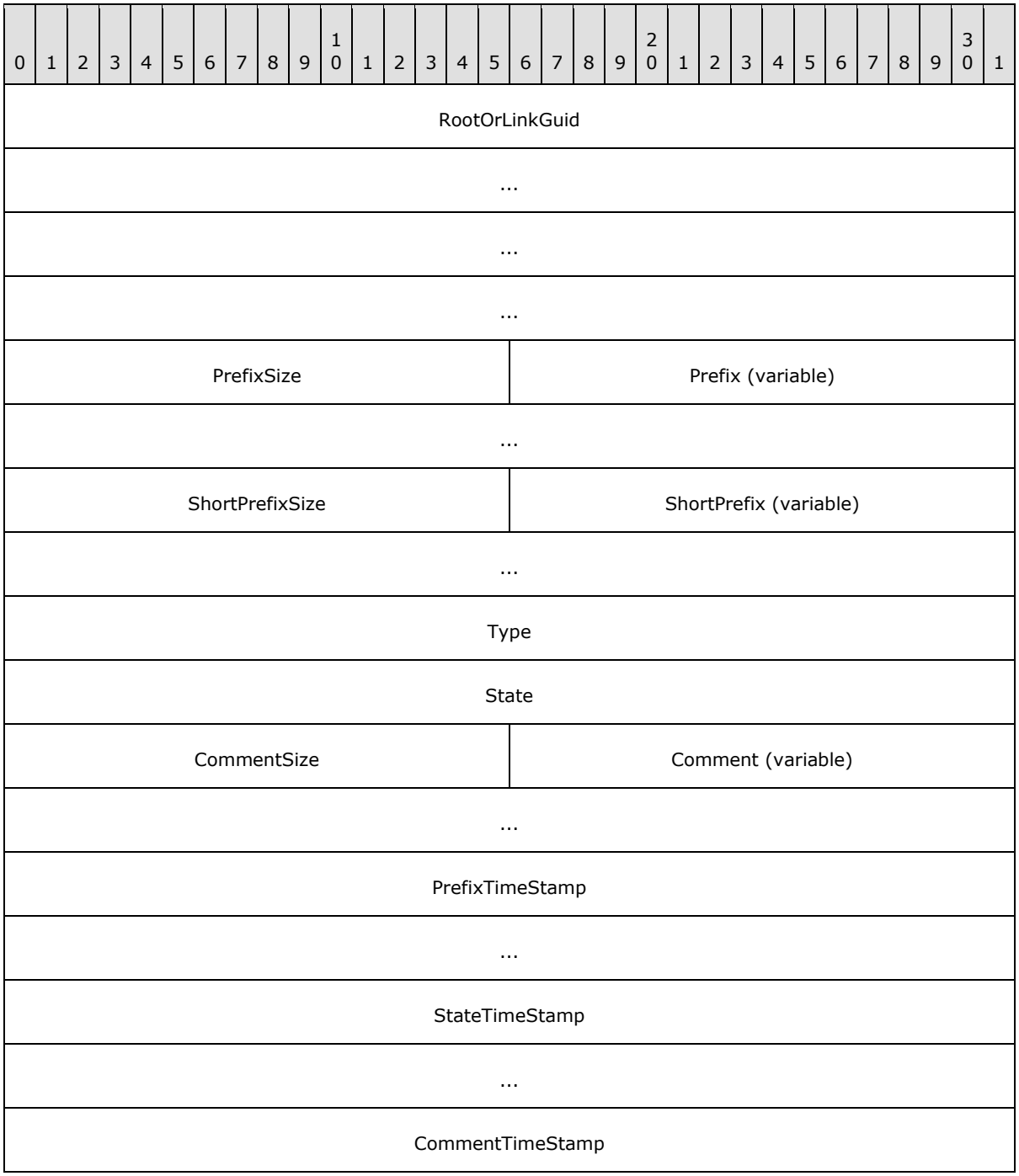
ReservedBLOB: When creating a new DFSNamespaceRootBLOB or DFSNamespaceLinkBLOB, this **ReservedBLOB** MUST be zero-filled. When updating an existing DFSNamespaceRootBLOB or DFSNamespaceLinkBLOB, the contents of this **ReservedBLOB** SHOULD be preserved.<15>

ReferralTTL: The referral time-out value, in seconds, for the DFS root or DFS link. The **ReferralTTL** is stored as an unsigned, 32-bit, little-endian integer.

2.3.3.1.1.2 DFSRootOrLinkIDBLOB

This BLOB contains name and other information about the DFS namespace root or the DFS link.

- If the PKT_ENTRY_TYPE_REFERRAL_SVC (0x00000080) bit is set in the **Type** field, then this BLOB describes the DFS root and is, hence, part of the DFSNamespaceRootBLOB.
- If the PKT_ENTRY_TYPE_REFERRAL_SVC (0x00000080) bit is not set in the **Type** field, then this BLOB describes a link and is, hence, part of the DFSNamespaceLinkBLOB.



...
Version

RootOrLinkGuid: A GUID that identifies the DFS root or the DFS link. It is used to generate the name "\<domainroot>\<GUIDString>" in the **BLOBName** field of the DFSNamespaceLinkBLOB, where <GUIDString> represents the string form of the GUID, as specified in [RFC4122], section 3.

PrefixSize: The size, in bytes, of the **Prefix** field, stored as an unsigned, 16-bit, little-endian integer.

Prefix: The name of the DFS namespace root or the DFS link. The **Prefix** is stored as a string of Unicode characters and MUST be a UNC path string with one leading backslash, instead of the usual two, without a null termination.

ShortPrefixSize: The size, in bytes, of the **ShortPrefix** field, stored as an unsigned, 16-bit, little-endian integer.

ShortPrefix: the name of the DFS namespace root or the DFS link, stored as a string of Unicode characters. This MUST be a UNC path string with one leading backslash, instead of the usual two, without a null termination. The string MAY be the same as that in the **Prefix** field or its **8.3 name**.<16>

Type: A bit field, stored as 32-bits in little-endian order, that describes this BLOB.

Value	Meaning
PKT_ENTRY_TYPE_DFS 0x00000001	This MUST be set to 1.
PKT_ENTRY_TYPE_OUTSIDE_MY_DOM 0x00000010	This MUST be set only when at least one DFS link target points to another DFS namespace. This MUST NOT be set for a DFS root.
PKT_ENTRY_TYPE_INSITE_ONLY 0x00000020	When set, instructs the DFS server to enable the DFS in-site referral mode.
PKT_ENTRY_TYPE_COST_BASED_SITE_SELECTION 0x00000040	Enables DFS referral site costing. This SHOULD be supported.<17>
PKT_ENTRY_TYPE_REFERRAL_SVC 0x00000080	This identifies the DFS namespace root.
PKT_ENTRY_TYPE_ROOT_SCALABILITY 0x00000200	This enables DFS root scalability mode. This SHOULD be supported.<18>
PKT_ENTRY_TYPE_TARGET_FAILBACK 0x00008000	This enables DFS client target failback for targets of this root or link. This SHOULD be supported.<19>

Undefined bit positions MUST be set to 0 on writes and ignored on reads.

State: The status of the DFS root or DFS link stored as an unsigned, 32-bit, little-endian integer. The DFS_VOLUME_STATES bitmask (0x0000000F) MUST be used to access the following DFS root or DFS link state from this field.

Value	Meaning
DFS_VOLUME_STATE_OK 0x00000001	The DFS root or DFS link state is okay.
DFS_VOLUME_STATE_INCONSISTENT 0x00000002	The DFS root or DFS link state is inconsistent. This field MUST NOT be set by the RPC methods and is meant to reflect the current state of the DFS namespace, as determined by the server.
DFS_VOLUME_STATE_OFFLINE 0x00000003	The DFS root or DFS link is offline and not available for use.
DFS_VOLUME_STATE_ONLINE 0x00000004	The DFS root or DFS link is online and available for use.

Undefined bit positions of this field MUST be set to 0 on writes and ignored on reads.

CommentSize: The size, in bytes, of the **Comment** field and stored as an unsigned, 16-bit, little-endian integer.

Comment: A string of Unicode characters whose size in bytes is specified by the **CommentSize** field. The **Comment** field is associated with the namespace root or link and is for informational purposes. The comment is meant for human consumption and does not affect server functionality.

PrefixTimeStamp: The time of the last **Prefix** field modification, stored as **FILETIME**. This SHOULD be set to the last modification time of this BLOB.

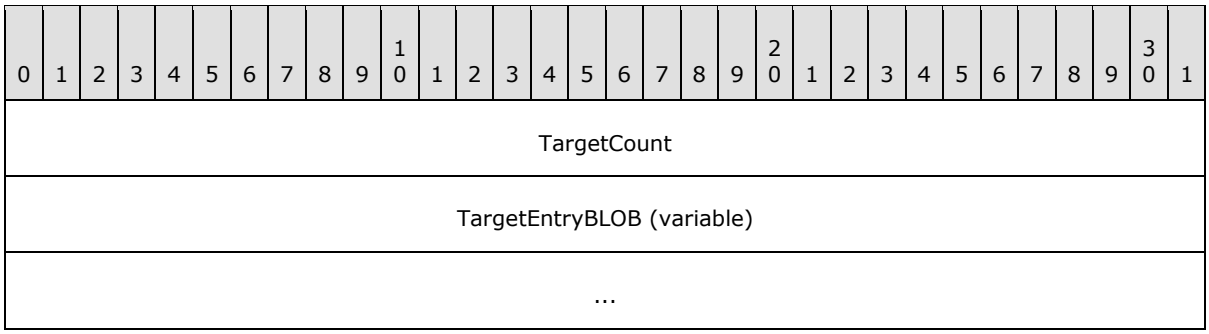
StateTimeStamp: The time of the last **State** field modification, stored as FILETIME. This SHOULD be set to the last modification time of this BLOB.

CommentTimeStamp: The time of the last **Comment** field modification, stored as FILETIME. This SHOULD be set to the last modification time of this BLOB.

Version: The version number of DFSRootOrLinkIDBLOB, stored as an unsigned, 32-bit, little-endian integer. When creating a new DFSRootOrLinkIDBLOB, this MUST be set to 0x00000003. When updating an existing DFSRootOrLinkIDBLOB, the existing value MUST be preserved.

2.3.3.1.1.3 DFSTargetListBLOB

The DFSTargetListBLOB contains information about all of the targets of the DFS root or the DFS link.

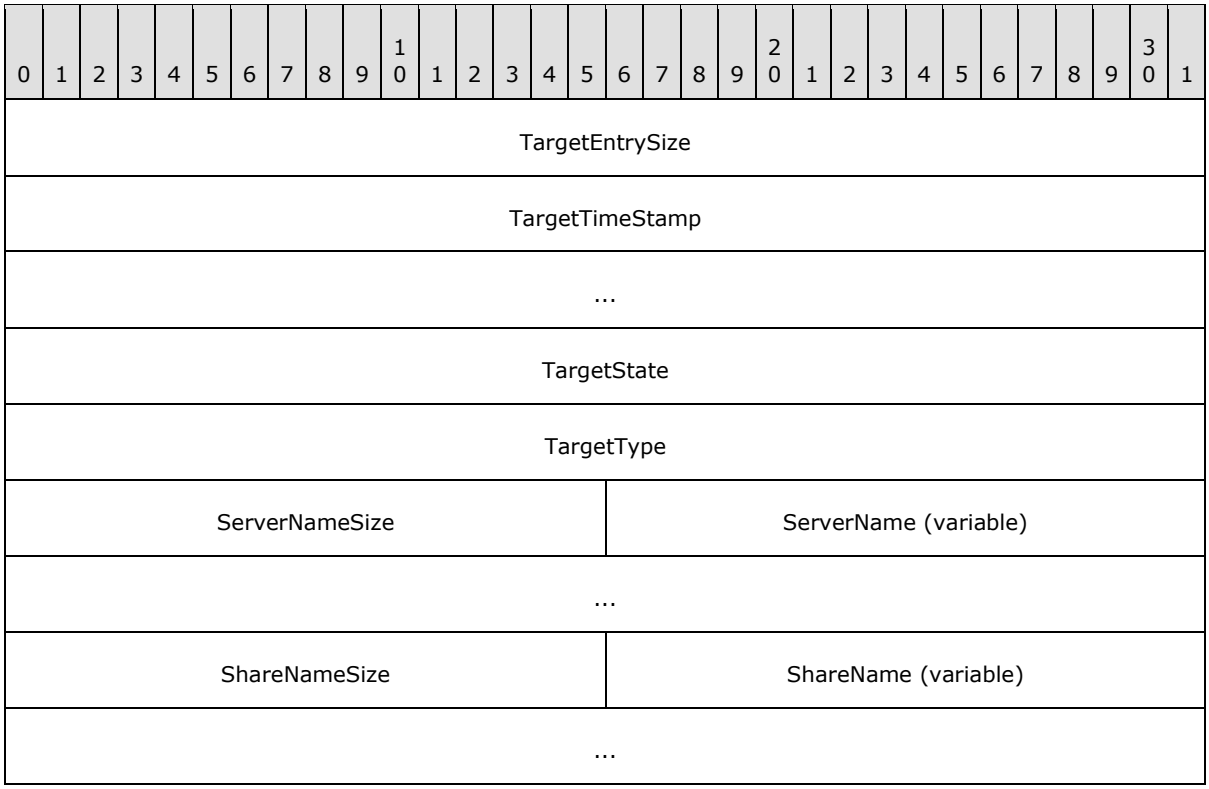


TargetCount: The number of TargetEntryBLOB fields contained in this BLOB, stored as an unsigned, 32-bit, little-endian integer.

TargetEntryBLOB: A BLOB that contains metadata for a DFS target.

2.3.3.1.1.3.1 TargetEntryBLOB

The TargetEntryBLOB holds metadata for the DFS target of a root or a link.



TargetEntrySize: The size, in bytes, of this target entry, starting from the **TargetTimeStamp** field, and stored as an unsigned, 32-bit, little-endian integer.

TargetTimeStamp: The last modification time of this target entry, stored as FILETIME.

If the server does not use this field to store a time stamp, the server MAY use this field for maintaining the target priority information of a DFS target. **TargetTimeStamp** should be

interpreted as a FILETIME if any of the high 56 bits are nonzero. If the high 56 bits are zero, the 64 bits of the **TargetTimeStamp** MUST have the following format:

PriorityRank: The priority rank of a target, ranging in value from 0x00 to 0x1F, where 0x00 is the highest rank. Priority ranks apply only within a priority class, not across priority classes.<20>

PriorityClass: The priority class of a target. This MUST be one of the following values.<21>

Value	Meaning
DFS_TARGET_PRIORITY_CLASS_SITE_COST_NORMAL 0x0	See DfsSiteCostNormalPriorityClass in section 2.2.2.8.
DFS_TARGET_PRIORITY_CLASS_GLOBAL_HIGH 0x1	See DfsSGlobalHighPriorityClass in section 2.2.2.8.
DFS_TARGET_PRIORITY_CLASS_SITE_COST_HIGH 0x2	See DfsSiteCostHighPriorityClass in section 2.2.2.8
DFS_TARGET_PRIORITY_CLASS_SITE_COST_LOW 0x3	See DfsSiteCostLowPriorityClass in section 2.2.2.8
DFS_TARGET_PRIORITY_CLASS_GLOBAL_LOW 0x4	see DfsGlobalLowPriorityClass in section 2.2.2.8

High56Bits: This MUST be 0 to interpret low 8 bits. Otherwise, all 64 bits MUST be interpreted as a FILETIME.

TargetState: The state of this target, stored as an unsigned, 32-bit, little-endian integer. The mask 0x0000000F is used to extract a bit field that contains one of the following valid state values.

Value	Meaning
DFS_STORAGE_STATE_OFFLINE 0x00000001	This target is offline and unavailable for use.
DFS_STORAGE_STATE_ONLINE 0x00000002	This target is online and available for use.
DFS_STORAGE_STATE_ACTIVE 0x00000004	This target is active.

TargetType: The type of target, stored as an unsigned, 32-bit, little-endian integer bit field. This MUST be set to 0x00000002.<22>

ServerNameSize: The size, in bytes, of the **ServerName** field, stored as an unsigned 16-bit, little-endian integer.

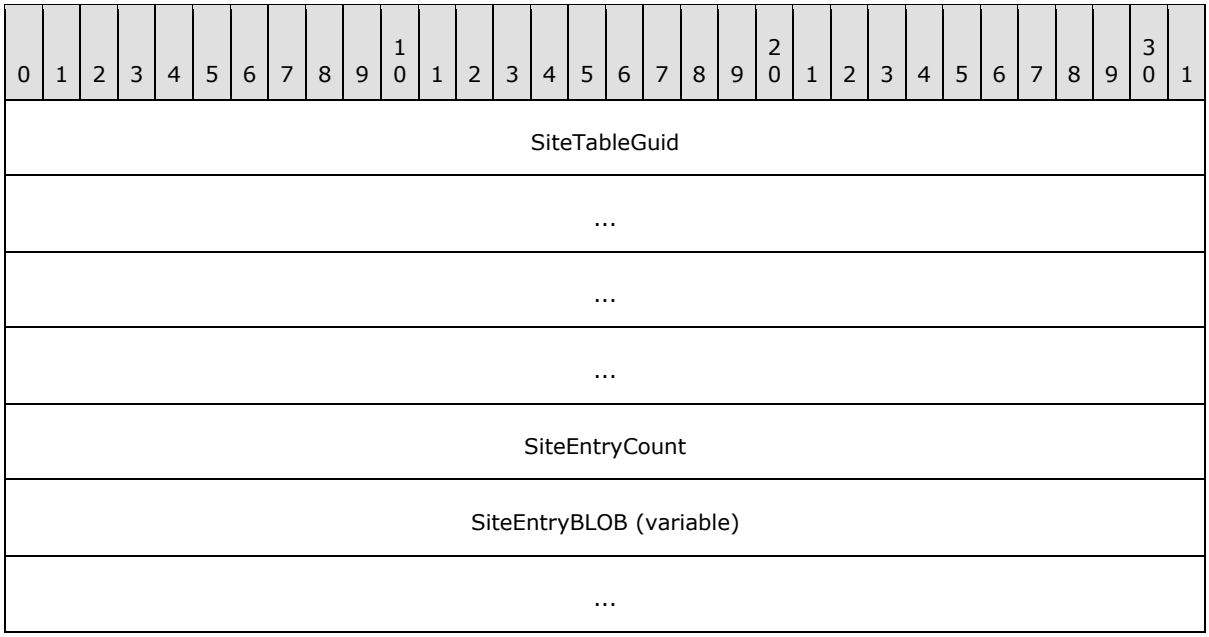
ServerName: An array of Unicode characters that contains the DFS target server host name. The size of the array is given in the **ServerNameSize** field.

ShareNameSize: The size, in bytes, of the **ShareName**, stored as an unsigned, 16-bit, little-endian integer.

ShareName: An array of Unicode characters that contains the DFS target share name.

2.3.3.1.1.4 SiteInformationBLOB

The SiteInformationBLOB contains the mapping from a DFS target host name to its AD DS site name. At most, only one BLOB of this type can exist. This BLOB contains zero or more SiteEntryBLOBs.



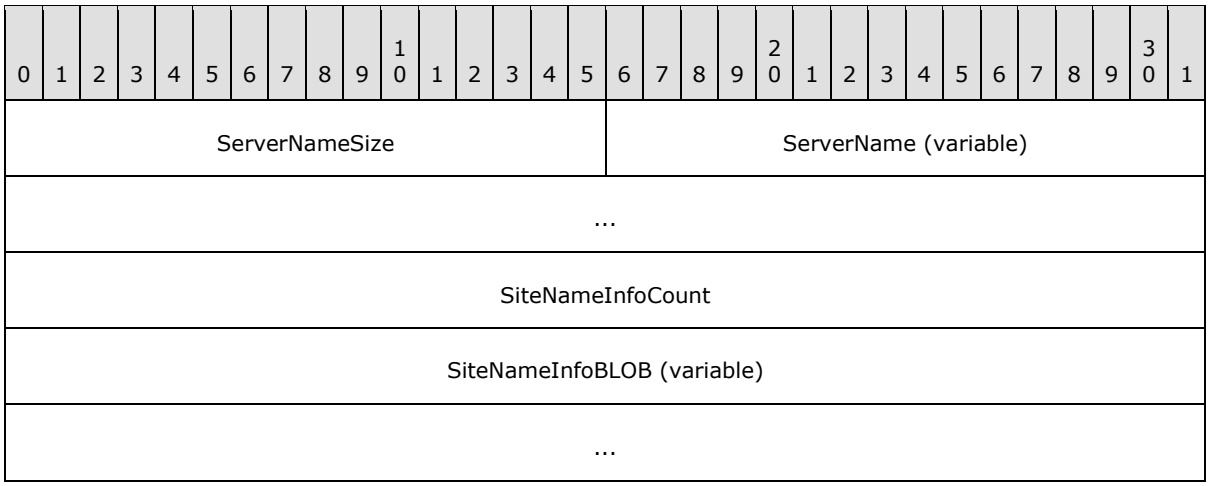
SiteTableGuid: The GUID that uniquely identifies the SiteInformationBLOB.

SiteEntryCount: The number of SiteEntryBLOBs in the SiteEntryBLOB field, stored as an unsigned, 32-bit, little-endian integer. This MAY be zero.<23><24>

SiteEntryBLOB: One or more BLOBs. Each BLOB contains the AD DS site of a root target or link target server in the DFS namespace.

2.3.3.1.1.4.1 SiteEntryBLOB

This BLOB contains a host name whose AD DS site information is specified by the SiteNameInfoBLOB (for more information, see section 2.3.3.1.1.4.1.1).



ServerNameSize: The size, in bytes, of the **ServerName** field, stored as an unsigned, 16-bit, little-endian integer.

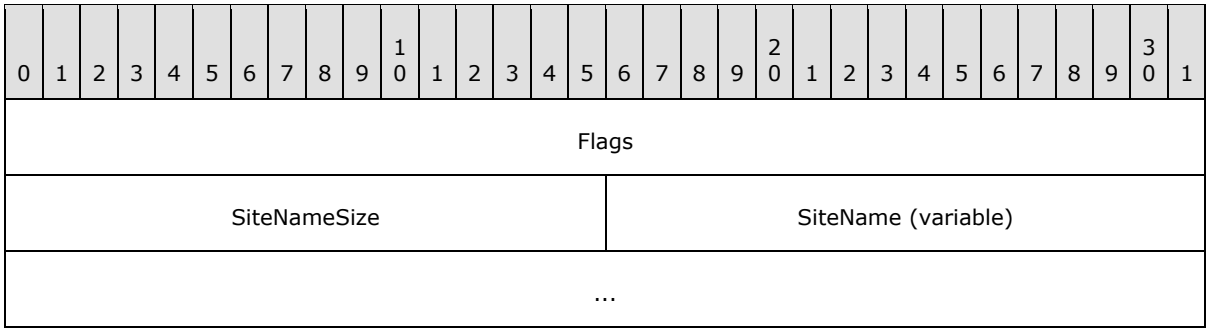
ServerName: String of Unicode characters. The DFS target host name.

SiteNameInfoCount: The number of SiteNameInfoBLOBs in the SiteNameInfoBLOB field, stored as an unsigned, 32-bit, little-endian integer.

SiteNameInfoBLOB: The BLOB containing the AD DS site name of the server in the SiteEntryBLOB.

2.3.3.1.1.4.1.1 SiteNameInfoBLOB

The SiteNameInfoBLOB contains the name of an AD DS site to which a server belongs.



Flags: This MUST be set to 0 on write. MUST be ignored on read.

SiteNameSize: The size, in bytes, of the **SiteName** field, stored as an unsigned, 16-bit, little-endian integer.

SiteName: String of Unicode characters. The AD DS site name of the server. The case of the site name, as provided by AD DS, MUST be preserved when storing in this field.

2.3.4 AD DS Schema for Domainv2-Based DFS Namespace

Each domainv2-based DFS namespace has one DFS namespace anchor LDAP entry, one DFS namespace LDAP entry below it, and one LDAP entry per DFS link in the namespace under the DFS

namespace LDAP entry. The following sections specify the mandatory and optional attributes of the object classes.

2.3.4.1 LDAP Entry for Domainv2-Based DFS Namespace Anchor

Each domainv2-based DFS namespace under the DFS configuration container has a DFS namespace anchor LDAP entry.

This AD DS Object has the following attribute.

Attribute	Description
msDFS-SchemaMajorVersion	An integer value containing the major version number of the supported DFS metadata format.

The Object class of the LDAP entry corresponding to the domainv2-based DFS namespace anchor is ms-DFS-Namespace-Anchor, and its schema is specified in [MS-ADSC]. The schema of the msDFS-SchemaMajorVersion attribute is specified in [MS-ADA2]. Future revisions of the DFS namespace will retain this LDAP entry to provide the DFS metadata version information of the DFS namespace.

2.3.4.2 LDAP Entry for Domainv2-Based DFS Namespace

A DFS namespace LDAP entry exists for each domainv2-based DFS namespace under the DFS namespace anchor LDAP entry.

This AD DS object has the following attributes. The schemas for these attributes are specified in [MS-ADA2].

Attribute	Description
msDFS-SchemaMajorVersion	An integer value that contains the major version number of the DFS metadata format supported.
msDFS-SchemaMinorVersion	An integer value that contains the minor version number of the DFS metadata format supported. The **rangeLower** attribute of the attribute schema's LDAP entry contains 0, and the **rangeUpper** attribute of the attribute schema's LDAP entry contains the highest minor version number supported.
msDFS-NamespaceIdentityGUIDv2	This is the time-stable identifier for a DFS namespace. It is a binary value set at DFS namespace creation time whose size is specified by the **rangeLower** and **rangeUpper** attributes.
msDFS-GenerationGUIDv2	A binary value whose size is specified by the **rangeLower** and **rangeUpper** attributes. This time-stable identifier is overwritten anytime the LDAP entry corresponding to the DFS namespace or the DFS link is modified.<25>
msDFS-LastModifiedv2	A time string format defined by ASN.1 standards, as specified in [X680]. The **Coordinated Universal Time (UTC)** in the form YYYYMMDDHHMMSS.0Z"0Z" indicates no time differential. This attribute is updated each time a DFS root or a DFS link entry is updated.
msDFS-Ttlv2	A 32-bit signed integer that is interpreted as an unsigned referral Time to Live (TTL), in seconds.

Attribute	Description
msDFS-TargetListv2	This attribute is where the domainv2 implementation stores the DFS target information. The information is stored as an XML document that contains a list of targets for the root as well as attributes associated with each target. Examples of target-specific attributes include target priority and referral enable/disable state. The maximum size is 2 MB. For the XML schema of the XML document, see Appendix C.
msDFS-Propertiesv2	This is a multivalued attribute that contains attributes corresponding to the DFS root or the DFS link (not individual targets). A case-insensitive Unicode string, the possible values are "root scalability", "target failback", "cost-based site selection", "insite", "referral svc", "outside my dom", and "type dfs".

The following attributes are mandatory: msDFS-SchemaMajorVersion, msDFS-SchemaMinorVersion, msDFS-NamespaceIdentityGUIDv2, msDFS-GenerationGUIDv2, msDFS-LastModifiedv2, msDFS-Ttlv2, msDFS-TargetListv2, and msDFS-Propertiesv2.

The msDFS-Commentv2 attribute is optional.

The object class of the LDAP entry corresponding to the domainv2-based DFS namespace is msDFS-Namespacev2, and its schema is specified in [MS-ADSC].

2.3.4.3 LDAP Entry for Domainv2-Based DFS Link

One LDAP entry exists for each DFS link in the namespace under the DFS namespace LDAP entry.

This AD DS Object has the following attributes. The schemas for these attributes are specified in [MS-ADA2].

Attribute	Description
msDFS-NamespaceIdentityGUIDv2	This is the time-stable identifier for a DFS namespace. It is a binary value set at DFS namespace creation time whose size is specified by the **rangeLower** and **rangeUpper** attributes.
msDFS-LinkIdentityGUIDv2	This is the time-stable identifier for a DFS link. It is a binary value set at DFS link creation time whose size is specified by the **rangeLower** and **rangeUpper** attributes. This value is retained in the dynamic object created when the link is deleted.
msDFS-GenerationGUIDv2	A binary value whose size is specified by the **rangeLower** and **rangeUpper** attributes. This time-stable identifier is overwritten anytime the LDAP entry corresponding to the DFS namespace or the DFS link is modified.<26>
msDFS-LinkPathv2	A case-insensitive Unicode string that is the DFS root-relative path to the DFS link reparse point. To simplify LDAP searches, path separators are forward slashes (/) instead of backward slashes (\).
msDFS-ShortNameLinkPathv2	A case-insensitive Unicode string that is the DFS namespace root-relative path to the DFS link reparse point in short name form. To simplify LDAP searches, path separators are forward slashes (/) instead of backward slashes (\).<27>
msDFS-LinkSecurityDescriptorv2	A self-relative security descriptor associated with a DFS link. This attribute is used for Microsoft Access-based directory enumeration support.

Attribute	Description
msDFS-LastModifiedv2	A time string format defined by ASN.1 standards. The UTC time in the form YYYYMMDDHHMMSS.0Z"0Z" indicates no time differential. This attribute is updated each time a DFS root or DFS link entry is updated.
msDFS-Ttlv2	A 32-bit signed integer that is interpreted as an unsigned referral TTL, in seconds.
msDFS-TargetListv2	This attribute is where the domainv2 implementation stores the DFS target information. The information is stored as an XML document that contains a list of targets for the root as well as attributes associated with each target. Examples of target-specific attributes include target priority and referral enable/disable state. The maximum size is 2 MB. For the XML schema of the XML document, see Appendix C.
msDFS-Propertiesv2	This is a multivalued attribute that contains attributes corresponding to the DFS root or the DFS link (not individual targets). A case-insensitive Unicode string, for example, "root scalability", "target failback", and "insite".
msDFS-Commentv2	An optional attribute that contains a comment associated with the DFS namespace root or link. A case-insensitive Unicode string, the possible values are "root scalability", "target failback", "cost-based site selection", "insite", "referral svc", "outside my dom", and "type dfs".

The following attributes are mandatory: msDFS-NamespaceIdentityGUIDv2, msDFS-LinkIdentityGUIDv2, msDFS-GenerationGUIDv2, msDFS-LinkPathv2, msDFS-LastModifiedv2, msDFS-Ttlv2, msDFS-TargetListv2, and msDFS-Propertiesv2.

The following attributes are optional: msDFS-ShortNameLinkPathv2, msDFS-LinkSecurityDescriptorv2, and msDFS-Commentv2.

The Object class of the LDAP entry corresponding to a DFS link in a domainV2-based DFS namespace is msDFS-Linkv2, and its schema is specified in [MS-ADSC].

2.3.4.4 LDAP Entry for Domainv2-Based Deleted Link

Only one LDAP entry corresponds to a deleted link in an domainv2-based DFS namespace. This is an AD dynamic object.

This AD DS object has the following attributes. The schemas for these attributes are specified in [MS-ADA2].

Attribute	Description
msDFS-NamespaceIdentityGUIDv2	This is the time-stable identifier for a DFS namespace. It is a binary value set at DFS namespace creation time whose size is specified by the **rangeLower** and **rangeUpper** attributes.
msDFS-LinkIdentityGUIDv2	This is the time-stable identifier for a DFS link. It is a binary value set at DFS link creation time whose size is specified by the **rangeLower** and **rangeUpper** attributes. This value is retained in the dynamic object created when the link is deleted.
msDFS-LastModifiedv2	A time string format defined by ASN.1 standards. The UTC time in the form YYYYMMDDHHMMSS.0Z"0Z" indicates no time differential. This attribute is updated each time a DFS root or DFS link entry is

Attribute	Description
	updated.
msDFS-LinkPathv2	A case-insensitive Unicode string that is the DFS root-relative path to the DFS link reparse point. To simplify LDAP searches, path separators are forward slashes (/) instead of backward slashes (\).
msDFS-Commentv2	An optional attribute that contains a comment associated with the DFS namespace root or link. A case-insensitive Unicode string.
msDFS-ShortNameLinkPathv2	A case-insensitive Unicode string that is the DFS namespace root-relative path to the DFS link reparse point in short name form. To simplify LDAP searches, path separators are forward slashes (/) instead of backward slashes (\).<28>

The following attributes are mandatory: msDFS-NamespaceIdentityGUIDv2, msDFS-LinkIdentityGUIDv2, msDFS-LastModifiedv2, and msDFS-LinkPathv2.

The following attributes are optional: msDFS-Commentv2 and msDFS-ShortNameLinkPathv2.

The object class of the LDAP entry corresponding to a DFS link in a domainv2-based DFS namespace is msDFS-DeletedLinkv2, and its schema is as specified in [MS-ADSC].

3 Protocol Details

3.1 Client Details

3.1.1 Abstract Data Model

No abstract data model is required.

3.1.2 Timers

No protocol timers are required beyond those used internally by the RPC method to implement resiliency to network outages, as specified in [MS-RPCE].

3.1.3 Initialization

The client creates an RPC binding handle to the server RPC method **endpoint** when an RPC method is called. For more information on binding handles, see [C706]. The client MAY create a separate binding handle for each method invocation, or it MAY reuse a binding handle for multiple invocations. The client MUST create an authenticated RPC binding handle.<29>

3.1.4 Higher-Layer Triggered Events

A client's invocation of each method is the result of local application activity. The local application at the client specifies values for all input parameters. No other higher-layer triggered events are processed.

3.1.5 Message Processing Events and Sequencing Rules

The client MUST pass any error received from the invocation of an RPC method to the application that issued the RPC call.

The client MUST NOT send RPC call opnums 7, 8, or 9 to the server. (For more information on message processing and sequencing rules, see section 3.2.5.)

3.1.5.1 Basic Methods

3.1.5.1.1 NetrDfsAdd

If a NetrDfsAdd call to the DFS root target fails with ERROR_NOT_SUPPORTED (0x00000032), the client MUST determine the PDC host name of the DFS root target AD DS domain. The client MUST then issue the NetrDfsAdd2 method instead, passing the PDC in the *DCName* parameter.<30>

3.1.5.1.2 NetrDfsRemove

If a NetrDfsRemove call fails with ERROR_NOT_SUPPORTED (0x00000032), the client MUST determine the PDC of the DFS root target server to which the RPC method was issued. The client MUST then invoke the NetrDfsRemove2 method and specify the PDC.<31>

3.1.5.1.3 NetrDfsSetInfo

If a NetrDfsSetInfo call fails with ERROR_NOT_SUPPORTED (0x00000032), the client MUST determine the PDC of the DFS root target server to which the RPC method was issued. The client MUST then invoke the NetrDfsSetInfo2 method and specify the PDC.<32><33>

3.1.5.1.4 NetrDfsEnum and NetrDfsEnumEx

The client MAY use either the NetrDfsEnum or the NetrDfsEnumEx method to enumerate roots and links. The client MAY use the value that **NetrDfsManagerGetVersion** returns to determine the enumeration method to use.<34><35>

Due to the possibility of concurrent updates to the DFS namespace, the client SHOULD NOT assume completeness or uniqueness of the results returned when resuming an enumeration (for more information on NetrDfsEnum, see section 3.2.5.1.7).<36>

3.1.5.2 Extended Methods

3.1.5.2.1 NetrDfsAdd2

The client MUST determine the PDC of a DFS root target server of the DFS namespace specified by the *DfsEntryPath* parameter and invoke the **NetrDfsAdd2** method specifying the PDC. If successful, the client MUST issue the **NetrDfsSetDcAddress** method to each server returned in the *ppRootList* parameter. The client must use the standard initialization process, as specified in section 3.1.3.<37>

3.1.5.2.2 NetrDfsRemove2

The client MUST determine the PDC of a DFS root target server of the DFS namespace specified by the *DfsEntryPath* parameter and invoke the NetrDfsRemove2 method specifying the PDC. If successful, the client MUST issue the **NetrDfsSetDcAddress** method to each server returned in the *ppRootList* parameter.<38>

3.1.5.2.3 NetrDfsSetInfo2

The client MUST determine the PDC of a DFS root target server of the DFS namespace specified by the *DfsEntryPath* parameter and invoke the NetrDfsSetInfo2 method specifying the PDC. If successful, the client MUST issue the **NetrDfsSetDcAddress (Opnum 17)** method to each server that is returned in the *ppRootList* parameter.<39><40>

3.1.5.3 Root Target Methods

3.1.5.3.1 NetrDfsAddFtRoot

The NetrDfsAddFtRoot method is supported only for a domainv1-based DFS namespace scenario.

The client MUST perform the following steps before invoking the NetrDfsAddFtRoot method:

1. Determine the PDC of the AD DS domain of the DFS root target server that the *ServerName* parameter specifies.
2. If an AD DS object for the domain-based DFS namespace does not already exist, create a new AD DS object for the domain-based DFS namespace, as specified in section 2.3.3.
3. Update the **access control list (ACL)** on the AD DS object of the DFS namespace to permit read/write access by the DFS root target server.
4. Call the **NetrDfsFlushFtTable** method on the PDC, specifying the DFS namespace name.
5. If all prior steps succeeded without error, then the client MUST issue the **NetrDfsSetDcAddress (Opnum 17)** method to each server that the *ppRootList* parameter returns.<41><42>

3.1.5.3.2 NetrDfsRemoveFtRoot

The client MUST determine the PDC of the AD DS domain of the DFS root target server that the *ServerName* parameter specifies. If the *ApiFlags* parameter is not DFS_FORCE_REMOVE, the client MUST issue the RPC method to the DFS root target server that the *ServerName* parameter specifies; otherwise, the client MUST issue the RPC method to the PDC.

This method is supported only for a domainv1-based DFS namespace scenario. If a client attempts to use it on a domainv2-based DFS namespace or target, the server MUST fail with a return value of ERROR_NOT_SUPPORTED.

If all prior steps succeeded without error, then the client MUST perform the following steps:

1. Invoke the **NetrDfsSetDcAddress** method to each server returned in the *ppRootList* parameter.<43>
2. Update the ACL on the AD DS object (as specified in section 2.3.3) of the DFS namespace to remove read/write access by the DFS root target server.
3. Remove the AD DS object itself if the **remoteServerName** attribute of the DFS namespace AD DS object (as specified in section 2.3.3) has exactly one value in it.
4. Call the **NetrDfsFlushFtTable** method on the PDC, specifying the DFS namespace name.

3.1.6 Timer Events

No protocol timer events are required on the client beyond those required in the underlying RPC call transport.

3.1.7 Other Local Events

No additional local events are used on the client beyond those maintained in the underlying **RPC transport**.

3.2 Server Details

3.2.1 Abstract Data Model

This section describes a conceptual model of a possible data organization that an implementation could maintain in order to participate in this protocol. This organization is provided to facilitate the explanation of how the protocol behaves. This specification does not mandate that implementations adhere to this model, as long as their external behavior is consistent with that described in this document. The following data items are implemented on the server side and are specific to this protocol:

- **PDCRoleHolder:** For servers of domain-based DFS namespaces, this is the PDC corresponding to the server's AD DS domain. DFS metadata updates SHOULD be issued to this DC. DFS metadata MAY be loaded from any DC in the server's domain.
- **DFSMetadataCache:** DFS metadata of DFS namespaces for which the server is a root target MAY<44> be cached as an optimization.

Note The preceding conceptual data can be implemented using a variety of techniques. There are no limitations on data implementation.

3.2.2 Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages, as specified in [MS-RPCE].

3.2.3 Initialization

The server MUST listen on the well-known endpoint defined for this RPC interface, as specified in section 2.1.

Information about DFS namespaces that the server hosts MAY be obtained from a configuration information store such as the registry. If the server is joined to a domain and is hosting at least one DFS namespace, it MAY determine the PDC for the domain and initialize PDCRoleHolder. As a performance optimization, it MAY preload the DFSMetadataCache with the DFS metadata of the DFS namespaces for which it is acting as a root target.

3.2.4 Higher-Layer Triggered Events

If DFS root scalability mode is disabled when a DFS root target server attempts to update the DFS metadata of a domain-based DFS namespace, then the server SHOULD notify other DFS root targets of the change in DFS metadata by issuing the NetrDfsSetInfo or NetrDfsSetInfo2 method with a *Level* parameter value of 101, and with the **DFS_INFO_101 State** field set to VOLUME_STATE_RESYNCHRONIZE.

If, however, a full synchronization is required because of a domainv2-based DFS namespace, the server SHOULD issue a NetrDfsSetInfo or NetrDfsSetInfo2 method with a *Level* parameter value of 101, and with the **DFS_INFO_101 State** field set to VOLUME_STATE_FORCE_RESYNC. For more information, see sections 3.2.5.1.5 and 3.2.5.2.4.<45>

When any RPC method is received for a domain-based DFS namespace that is hosted by the server, the server MUST first check whether the DFS namespace is up-to-date with any changes that may have happened on it. If DFS root scalability mode is disabled for the domain-based DFS namespace, this check MUST be done against the PDC; otherwise, this check MUST be done against a DC closest to the server in terms of AD DS site cost. When any changes are detected, the server MUST first bring the namespace it hosts up-to-date and then process the RPC method received. The actual details of this operation are implementation-specific.

3.2.5 Message Processing Events and Sequencing Rules

To avoid updates to a domainv1-based DFS namespace that are based on state data, the DFS metadata to use for an update operation MUST be obtained by querying the PDCRoleHolder, even if DFS root scalability mode is enabled.. If the **pKTGuid** value matches a cached value, the server MAY work on a cached copy of the **pKTGuid**. The update operation MUST be committed by issuing LDAP writes for both the updated **pKT** attribute and a newly generated **pKTGuid** attribute. Using the same LDAP update operation for both attributes ensures atomicity of the update.

For non-update operations in domain-based DFS namespaces, DFS servers MAY retrieve DFS metadata from any DC within the domain. For update operations in domain-based DFS namespaces, DFS servers MUST retrieve and store DFS metadata on the PDC.

Unless noted otherwise, DFS servers MUST process host names as case-insensitive string literals. The DFS server MUST NOT, for example, consider a DNS-conformant host name (as specified in [RFC1034]) and an IP address as equivalent, even if the host name resolves via DNS to the IP address.

This protocol uses Win32 error codes. These values are taken from the Windows error number space, as specified in [MS-ERREF]. Vendors SHOULD reuse those values with their indicated meanings. Choosing any other value runs the risk of future collisions.<46>

The remainder of this section describes the methods used in the DFS: Namespace Management Protocol. The following table lists opnum values associated with the methods described in this document, as well as the section where each is described.

Methods in RPC Opnum Order

Method	Description
NetrDfsManagerGetVersion	A basic method that returns the version number of the DFS server. Opnum: 0
NetrDfsAdd	A basic method that creates a new DFS link or that adds a new target to an existing link of a DFS namespace. Opnum: 1
NetrDfsRemove	A basic method that removes a link or a link target from a DFS namespace. Opnum: 2
NetrDfsSetInfo	A basic method that sets or modifies information relevant to a specific DFS root, DFS root target, DFS link, or DFS link target. Opnum: 3
NetrDfsGetInfo	A basic method that returns information about a DFS root, a DFS link, or a DFS namespace. Opnum: 4
NetrDfsEnum	A basic method that enumerates the DFS roots hosted on a server or the DFS links of a namespace on the server. Opnum: 5
NetrDfsMove	A basic method that renames or moves one or more DFS links. Opnum: 6
Opnum7NotUsedOnWire	Reserved for local use. Opnum: 7
Opnum8NotUsedOnWire	Reserved for local use. Opnum: 8
Opnum9NotUsedOnWire	Reserved for local use. Opnum: 9
NetrDfsAddFtRoot	A root target method that creates a new domain-based DFS namespace or that adds a root target to an existing namespace. Opnum: 10
NetrDfsRemoveFtRoot	A root target method that removes a root target from a

Method	Description
	domain-based DFS namespace or that removes a domain-based DFS namespace. Opnum: 11
NetrDfsAddStdRoot	A stand-alone namespace method that creates a new stand-alone DFS namespace. Opnum: 12
NetrDfsRemoveStdRoot	A stand-alone namespace method that deletes a stand-alone DFS namespace. Opnum: 13
NetrDfsManagerInitialize	A basic method that instructs the DFS server to discard its current state and to reinitialize itself from its stored configuration settings. Opnum: 14
NetrDfsAddStdRootForced	A stand-alone namespace method that creates a new stand-alone DFS namespace without verifying the existence of the DFS root target share. Opnum: 15
NetrDfsGetDcAddress	A domain-based namespace method that returns the host name of the DC for the client to use during the following processes: creating a domain-based DFS namespace, adding a root target to a domain-based DFS namespace, removing a root target from a domain-based DFS namespace, or removing a domain-based DFS namespace. Opnum: 16
NetrDfsSetDcAddress	A domain-based namespace method that instructs a DFS server to use a specific DC for DFS metadata access in a domain-based DFS namespace. Opnum: 17
NetrDfsFlushFtTable	A root target method that instructs the DFS server on a DC to purge a domain-based DFS entry from its referral cache. Opnum: 18
NetrDfsAdd2	An extended method that creates a new DFS link or that adds a new target to an existing link of a DFS namespace. Opnum: 19
NetrDfsRemove2	An extended method that removes a link or a link target. Opnum: 20
NetrDfsEnumEx	An extended method that enumerates DFS roots hosted on a machine or DFS links of a namespace. Opnum: 21
NetrDfsSetInfo2	An extended method that sets or modifies the information that is associated with a DFS root, a DFS root target, a DFS link, or a DFS link target.

Method	Description
	Opnum: 22
NetrDfsAddRootTarget	A basic method that creates a stand-alone DFS namespace, a domainv1-based DFS namespace, or a domainv2-based DFS namespace.<47> Opnum: 23
NetrDfsRemoveRootTarget	A basic method that deletes a stand-alone DFS namespace, a domainv1-based DFS namespace, or a domainv2-based DFS namespace.<48> Opnum: 24
NetrDfsGetSupportedNamespaceVersion	A basic method that determines the supported DFS metadata version number. This method is useful in determining an appropriate version number to pass to the NetrDfsAddRootTarget() method.<49> Opnum: 25

In the preceding table, the term "Reserved for local use" means that the client MUST NOT send the opnum, and the server behavior is undefined<50> because it does not affect interoperability.

3.2.5.1 Basic Methods

3.2.5.1.1 NetrDfsManagerInitialize (Opnum 14)

The **NetrDfsManagerInitialize** method instructs the DFS server to discard its current state and reinitialize itself from its stored configuration settings.

The **NetrDfsManagerInitialize** method has the following **Microsoft Interface Definition Language (MIDL)** syntax.

```
NET_API_STATUS NetrDfsManagerInitialize(
  [in, string] WCHAR* ServerName,
  [in] DWORD Flags
);
```

ServerName: The pointer to a null-terminated Unicode host name string of the DFS root target server or DC where the DFS service is to be reinitialized.

Flags: This parameter MUST be zero.

Return Values: The method returns 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

Support for this method is optional. If supported, the DFS server MUST discard its current state and reinitialize itself from its stored configuration settings.<51><52>

3.2.5.1.2 NetrDfsManagerGetVersion (Opnum 0)

The **NetrDfsManagerGetVersion** method returns the version number of the DFS server in use on the server.

The **NetrDfsManagerGetVersion** method has the following MIDL syntax.

```
DWORD NetrDfsManagerGetVersion();
```

This method has no parameters.

Return Values: This method MUST return one of the following values.

Return value	Description
0x00000001	The server MUST support stand-alone DFS namespaces and opnums from 0 through 5, inclusive. The server MAY support domain-based DFS namespaces and other opnums.
0x00000002	In addition to the preceding, the server MUST support domain-based DFS namespaces and opnums 10 through 22, inclusive. The server MAY support hosting more than one DFS namespace on the same server.
0x00000004	In addition to the preceding, the server MUST support hosting more than one DFS namespace on the same server and Level parameter value 200 of the NetrDfsEnumEx method. It SHOULD support opnum 6.
0x00000006	In addition to the above, the server MUST support domainv2-based DFS namespace and opnums 23 through 25, inclusive.

The clients MAY use the version information to determine the RPC methods that the DFS server supports.<53><54><55><56><57>

3.2.5.1.3 NetrDfsAdd (Opnum 1)

The **NetrDfsAdd (Opnum 1)** method creates a new DFS link or adds a new target to an existing link of a DFS namespace.

The **NetrDfsAdd (Opnum 1)** method has the following MIDL syntax.

```
NET_API_STATUS NetrDfsAdd(
  [in, string] WCHAR* DfsEntryPath,
  [in, string] WCHAR* ServerName,
  [in, unique, string] WCHAR* ShareName,
  [in, unique, string] WCHAR* Comment,
  [in] DWORD Flags
);
```

DfsEntryPath: The pointer to a DFS link path that contains the name of an existing link when additional link targets are being added or the name of a new link is being created.

ServerName: The pointer to a null-terminated Unicode string that specifies the DFS link target host name.

ShareName: The pointer to a null-terminated Unicode DFS link target share name string. This may also be a share name with a path relative to the share, for example, "share1\mydir1\mydir2". When specified this way, each pathname component MUST be a directory.

Comment: The pointer to a null-terminated Unicode string that contains a comment associated with this root or link. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality. The comment MUST be ignored when adding a target to an existing link.

Flags: The following table lists the flag that indicates the operation to perform.

Value	Meaning
0x00000000	This creates a new link or adds a new target to an existing link.
DFS_ADD_VOLUME 0x00000001	This creates a new link in the DFS namespace if one does not already exist or fails if a link already exists.
DFS_RESTORE_VOLUME 0x00000002	This adds a target without verifying its existence.

Return Values: The method MUST return 0 on success or a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The server MUST verify the existence of the DFS namespace that the *DfsEntryPath* parameter specifies. If the namespace does not exist, the server MUST return an implementation-defined failure value.

If the link to be added already exists, a new link target is added. If the new link target already exists, the server MUST return an implementation-defined failure value.

The *Comment* parameter MUST be ignored when adding a target to an existing link.

The server SHOULD support the creation of a new link without requiring the *Flags* parameter to be *DFS_ADD_VOLUME*.<58>

The server MUST update the following fields in the DFS metadata for a domainv1-based DFS namespace.

Operation	DFS metadata changes required
Adding a new link	New DFSNamespaceLinkBLOB and BLOBElementCount.
Adding a new target to an existing link	TargetCount in existing DFSTargetListBLOB, new TargetEntryBLOB, TargetListBLOBSize, and BlobDataSize of DFSNamespaceLinkBLOB.

The server MUST update the following fields in the DFS metadata for a domainv2-based DFS namespace.

Operation	DFS metadata changes required
Adding a new link	The following mandatory attributes MUST be updated: msDFS-NamespaceIdentityGUIDv2, msDFS-LinkIdentityGUIDv2, msDFS-GenerationGUIDv2, msDFS-LinkPathv2, msDFS-LastModifiedv2, msDFS-TargetListv2, msDFS-Propertiesv2, and msDFS-Ttlv2. The following optional attribute MAY be updated: msDFS-Commentv2.<59>
Adding a new target to an existing link	The **Update targetCount**, **totalStringLengthInBytes**, **priority** and **state** attributes in msDFS-TargetListv2. Update msDFS-LastModifiedv2.

The server MUST synchronously update the DFS metadata of a domain-based DFS namespace. If DFS root scalability mode is not enabled for the domain-based DFS namespace, the server MUST notify other DFS root targets of the change in DFS metadata by issuing a **NetrDfsSetInfo (Opnum 3)** method with the Level parameter 101, and with the State field of **DFS_INFO_101** set to VOLUME_STATE_RESYNCHRONIZE.<60><61>

For a domainv2-based DFS namespace, if the DFS root target server is switching to a new PDC, or if DFS root scalability mode is enabled but synchronization is done with a different DC other than the PDC, the server MUST notify other DFS root targets of the change in DFS metadata. This is done by issuing a **NetrDfsSetInfo** method with the *Level* parameter 101, and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_FORCE_SYNC to ensure that a full synchronization is done, instead of an incremental synchronization that picks up only changed data.

3.2.5.1.4 NetrDfsRemove (Opnum 2)

The **NetrDfsRemove** method removes a link or a link target from a DFS namespace. A link can be removed regardless of the number of targets associated with it.

The **NetrDfsRemove** method has the following MIDL syntax.

```
NET_API_STATUS NetrDfsRemove(
  [in, string] WCHAR* DfsEntryPath,
  [in, unique, string] WCHAR* ServerName,
  [in, unique, string] WCHAR* ShareName
);
```

DfsEntryPath: The pointer to the DFS link path that contains the name of an existing link.

ServerName: The pointer to a null-terminated Unicode DFS link target host name string. Clients MUST set *ServerName* to a null pointer in requests to remove the link and all its link targets.

ShareName: The pointer to a null-terminated Unicode DFS link target share name string. This may also be a share name with a path relative to the share, for example, "share1\mydir1\mydir2". Clients MUST set *ShareName* to a null pointer in requests to remove the link and all its link targets.

Return Values: The method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The server MUST verify the existence of the DFS namespace that the *DfsEntryPath* parameter specifies. If the namespace does not exist, the server MUST return a failure.

If the *ServerName* and *ShareName* parameters are both NULL, the server MUST remove the link and all its link targets. If either *ServerName* or *ShareName* is not NULL, the server MUST remove the specified link target. If the specific target is the last target of the link, the server MUST remove the link as well.

The server MUST update the following fields in the DFS metadata for a domainv1-based DFS namespace.

Operation	DFS metadata changes required
Remove link	Remove DFSNamespaceLinkBLOB; update BLOBElementCount.
Remove link target	Update TargetCount in existing DFSTargetListBLOB, remove TargetEntryBLOB. Update DFSTargetListBLOBSize, update BLOBDataSize of DFSNamespaceLinkBLOB.

The server MUST update the following fields in the DFS metadata for a domainv2-based DFS namespace.

Operation	DFS metadata Changes Required
Remove link	Remove the DFS link object.
Remove link target	Update **TargetCount** and **totalStringLengthInBytes** attributes in msDFS-TargetListv2; update msDFS-LastModifiedv2.

A remove link operation in a domainv2-based DFS namespace first creates a dynamic object for the entry to be deleted. To create a dynamic object, the server MUST do the following:

- Set the object class to msDFS-DeletedLinkv2; a normal DFS link LDAP entry's object class is msDFS-Linkv2.
- Set the deleted DFS link's identity GUID.
- Set an updated msDFS-LastModifiedv2 time-stamp attribute.

If the dynamic object is created successfully, the original link LDAP entry is then deleted. If the delete is successful, the dynamic object is left intact; otherwise, the dynamic object is itself deleted. The advantage of using a dynamic object is that AD performs garbage collection.

The server MUST synchronously update the DFS metadata of a domain-based DFS namespace. If DFS root scalability mode is not enabled for the domain-based DFS namespace, the server MUST notify other DFS root targets of the change in DFS metadata by issuing a **NetrDfsSetInfo** method with the *Level* parameter 101, and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_RESYNCHRONIZE.<62><63>

For a domainv2-based DFS namespace, if the DFS root target server is switching to a new PDC, or if DFS root scalability mode is enabled, but synchronization is done with a different DC other than the PDC, the server MUST notify other DFS root targets of the change in DFS metadata. This is done by

issuing a **NetrDfsSetInfo** method with the *Level* parameter 101, and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_FORCE_SYNC to ensure that a full synchronization is done, instead of an incremental synchronization that picks up only changed data.

3.2.5.1.5 NetrDfsSetInfo (Opnum 3)

The **NetrDfsSetInfo** method sets or modifies information relevant to a specific DFS root, DFS root target, DFS link, or DFS link target.

The **NetrDfsSetInfo** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsSetInfo(
  [in, string] WCHAR* DfsEntryPath,
  [in, unique, string] WCHAR* ServerName,
  [in, unique, string] WCHAR* ShareName,
  [in] DWORD Level,
  [in, switch_is(Level)] DFS_INFO_STRUCT* DfsInfo
);
```

DfsEntryPath: The pointer to a DFS root or a DFS link path.

ServerName: The pointer to a null-terminated Unicode DFS root target or DFS link target host name string. Clients MUST set this to a null pointer when the DFS root or DFS link is used and not the DFS root target or DFS link target.

ShareName: The pointer to a null-terminated Unicode string DFS root target or DFS link target host name. Clients MUST set this to a null pointer when the DFS root or DFS link is used and not the DFS root target or DFS link target.

Level: Specifies the information level of the data and, in turn, determines the action the method performs. Clients MUST set this to one of the following values.

Value	Meaning
Level_100 0x00000064	Sets the comment associated with the root or link specified in the *DfsInfo* parameter. In this case, the *DfsInfo* parameter MUST point to a **DFS_INFO_100** structure. The *ServerName* and *ShareName* parameters MUST be null.
Level_101 0x00000065	Sets the state associated with the root, link, root target, or link target specified in *DfsInfo*. In this case, *DfsInfo* MUST point to a **DFS_INFO_101** structure.<64>
Level_102 0x00000066	Sets the time-out value associated with the root or link specified in *DfsInfo*. In this case, *DfsInfo* MUST point to a **DFS_INFO_102** structure. The *ServerName* and *ShareName* parameters MUST be null.
Level_103 0x00000067	Sets the property flags for the root or link specified in *DfsInfo*. In this case, *DfsInfo* MUST point to a **DFS_INFO_103** structure. The *ServerName* and *ShareName* parameters MUST be null.
Level_104 0x00000068	Sets the target priority rank and class for the root target or link target specified in *DfsInfo*. In this case, *DfsInfo* MUST point to a **DFS_INFO_104** structure.
Level_105 0x00000069	Sets the comment, state, and time-out information, along with property flags, for the namespace root or link specified in *DfsInfo*. Does not apply to the root target or link target. In this case, *DfsInfo* MUST point to a **DFS_INFO_105** structure. The *ServerName* and *ShareName* parameters MUST be null.

Value	Meaning
Level_106 0x0000006A	Sets the target state and priority for the DFS root target or DFS link target specified in *DfsInfo*.<65> This does not apply to the DFS namespace root or link. *DfsInfo* MUST point to a **DFS_INFO_106** structure.
Level_107 0x0000006B	Sets the comment, state, time-out, and security descriptor information, along with property flags, for the namespace root or link specified in *DfsInfo*. This does not apply to a DFS root target or DFS link target. In this case, *DfsInfo* MUST point to a **DFS_INFO_107** structure. The *ServerName* and *ShareName* parameters MUST be null. The security descriptor MUST NOT have owner, group, or **system access control lists (SACLs)** in it.
Level_150 0x00000096	Sets the security descriptor associated with a DFS link. Only stand-alone DFS namespaces and domainv2-based DFS namespaces are supported. The *ServerName* and *ShareName* parameters must both be NULL. When using NetrDfsSetInfo(), the security descriptor MUST NOT have owner, group, or SACLs in it.

Any other value is invalid. If the *Level* parameter is not equal to one of the valid values, the server MUST fail the call.<66>

DfsInfo: The pointer to a **DFS_INFO_STRUCT** union that contains the specified data. The value of the *Level* parameter selects the case of the union.

Return Values: The method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The server MUST verify the existence of the DFS namespace that the *DfsEntryPath* parameter specifies. If the namespace does not exist, the server MUST return a failure.

The server MUST fail any attempt to set the state of a DFS root, a DFS link, a DFS root target or a DFS link target to a value that is not specified for the *Level* parameter. The server MUST fail any attempt to set the property flags on a DFS link that are defined only for a DFS root.

When the *Level* parameter is 101 and the **State** field in the **DFS_INFO_101** structure is DFS_VOLUME_STATE_RESYNCHRONIZE, the server MUST reload the contents of the DFSMetadataCache for the DFS namespace that the *ShareName* parameter specifies.. It MUST then update its local DFS namespace information by comparing this information against the DFS metadata. The details of this update operation are implementation-dependent.

Level 107 can be used for a stand-alone, domainv1-based, or domainv2-based DFS namespace root, or for a domainv1-based DFS link where the *pSecurityDescriptor* parameter MUST be NULL. It MUST fail with ERROR_NOT_SUPPORTED if the *pSecurityDescriptor* is not NULL. It can also be used for a domainv2-based DFS link that does not require *pDescriptor* to be NULL. The *ServerName* and *ShareName* parameters MUST both be NULL.

The server MUST update the following fields in the DFS metadata for a domainv1-based DFS namespace, depending on the value of the *Level* parameter.

Value	DFS metadata changes required
100 (0x00000064)	Update **CommentSize** and **Comment** in DFSRootOrLinkIDBLOB and BLOBDataSize of DFSNamespaceLinkBLOB.
101 (0x00000065)	If a root or link, update the **State** field of DFSRootOrLinkIDBLOB. If a root target or link target, update the **TargetState** field of TargetEntryBLOB.
102 (0x00000066)	Update the **ReferralTTL** field of DFSNamespaceRootBLOB or DFSNamespaceLinkBLOB.
103 (0x00000067)	Update the **Type** field of DFSRootOrLinkIDBLOB.
104 (0x00000068)	Update the **PriorityRank** and **PriorityClass** fields of TargetEntryBLOB.
105 (0x00000069)	Update **CommentSize** and **Comment** in DFSRootOrLinkIDBLOB, the **State** field of DFSRootOrLinkIDBLOB, the **ReferralTTL** field of DFSNamespaceRootBLOB or DFSNamespaceLinkBLOB, and the **Type** field of DFSRootOrLinkIDBLOB.
106 (0x0000006A)	Update the **PriorityRank**, **PriorityClass**, and **TargetState** fields of TargetEntryBLOB.
107 (0x0000006B)	Update Comment, State of Link, Timeout, PropertyFlagMask, and PropertyFlags.

The server MUST update the following fields in the DFS metadata for a domainv2-based DFS namespace, depending on the value of the *Level* parameter.

Value	DFS metadata changes required
100 (0x00000064)	Update msDFS-Commentv2.
101 (0x00000065)	**State** field of msDFS-TargetListv2.
102 (0x00000066)	Update msDFS-Ttlv2.
103 (0x00000067)	Update msDFS-Propertiesv2.
104 (0x00000068)	Update the **priorityClass** and **priorityRank** attributes in msDFS-TargetListv2.
105 (0x00000069)	Update msDFS-Commentv2, msDFS-Ttlv2, State, and msDFS-Propertiesv2.
106 (0x0000006A)	Update priorityClass, priorityRank, and State attributes of msDFS-TargetListv2.
107 (0x0000006B)	Update msDFS-Commentv2, msDFS-Ttlv2, msDFS-Propertiesv2, and msDFS-LinkSecurityDescriptorv2.
150 (0x00000096)	Update msDFS-LinkSecurityDescriptorv2.

The server MUST synchronously update the DFS metadata of a domain-based DFS namespace. If DFS root scalability mode is not enabled for the domain-based DFS namespace, the server MUST notify other DFS root targets of the change in DFS metadata by issuing a **NetrDfsSetInfo** method with the *Level* parameter 101, and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_RESYNCHRONIZE.<67>

For a domainv2-based DFS namespace, if the DFS root target server is switching to a new PDC, or if DFS root scalability mode is enabled, but synchronization is done with a different DC other than the PDC, the server MUST notify other DFS root targets of the change in DFS metadata. This is done by issuing a **NetrDfsSetInfo** method with the *Level* parameter 101, and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_FORCE_SYNC to ensure that a full synchronization is done, instead of an incremental synchronization that picks up only changed data.

3.2.5.1.6 NetrDfsGetInfo (Opnum 4)

The **NetrDfsGetInfo** method returns information about a DFS root or a DFS link of the specified DFS namespace.

The **NetrDfsGetInfo** method has the following MIDL syntax.

```
NET_API_STATUS NetrDfsGetInfo(
  [in, string] WCHAR* DfsEntryPath,
  [in, unique, string] WCHAR* ServerName,
  [in, unique, string] WCHAR* ShareName,
  [in] DWORD Level,
  [out, switch_is(Level)] DFS_INFO_STRUCT* DfsInfo
);
```

DfsEntryPath: The pointer to a DFS root or a DFS link path.

ServerName: This parameter MUST be a null pointer.

ShareName: This parameter MUST be a null pointer.

Level: This parameter specifies the information level of the data and, in turn, determines the action the method performs. Clients MUST set this to one of the following values.

Value	Meaning
Level_1 0x00000001	Returns the name of the DFS root or the DFS link.
Level_2 0x00000002	Returns the name, comment, state, and number of targets for the DFS root or the DFS link.
Level_3 0x00000003	Returns the name, comment, state, number of targets, and information about each target for the DFS root or the DFS link.
Level_4 0x00000004	Returns the name, comment, state, time-out, GUID, number of targets, and information about each target for the DFS root or the DFS link.
Level_5 0x00000005	Returns the name, comment, state, time-out, GUID, property flags, metadata size, and number of targets for the DFS root or the DFS link.
Level_6 0x00000006	Returns the name, comment, state, GUID, time-out, property flags, metadata size, number of targets, and a list of targets for the DFS root or the DFS link.

Value	Meaning
Level_7 0x00000007	Returns the version number GUID of the DFS metadata.
Level_8 0x00000008	Returns the name, comment, state, time-out, GUID, property flags, metadata size, number of targets, and security descriptor associated with the DFS link. Only stand-alone DFS namespaces and domainv2-based DFS namespaces are supported.
Level_9 0x00000009	Returns the name, comment, state, GUID, time-out, property flags, metadata size, number of targets, list of targets, and security descriptor for the DFS link. Only stand-alone DFS namespaces and domainv2-based DFS namespaces are supported.
Level_50 0x00000032	Returns the DFS metadata version and capability information of an existing DFS namespace. This level is valid only for the DFS namespace root, not for DFS links. The *ServerName* and *ShareName* parameters must be NULL. This level is supported only in Windows Vista and Windows Server 2008.
Level_100 0x00000064	Returns the comment associated with the root or DFS link specified in the *DfsEntryPath* parameter.
Level_150 0x00000096	Returns the security descriptor associated with a DFS link. Only stand-alone DFS namespaces and domainv2-based DFS namespaces are supported.

Any other value is invalid. If the *Level* parameter is not equal to one of the valid values, the server MUST fail the call.<68><69>

DfsInfo: The pointer to a **DFS_INFO_STRUCT** union to receive the returned information. The case of the union is selected by the value of the *Level* parameter.

Return Values: The method returns 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The server MUST verify the existence of the DFS namespace that the *DfsEntryPath* parameter specifies. If the namespace does not exist, the server MUST return a failure.

The server MUST obtain the required information from the DFS metadata of the DFS namespace.

The server's response depends on the value of the *Level* parameter.

The server MUST use the following fields in the DFS metadata for a domainv1-based DFS namespace to return the required information, depending on the value of the *Level* parameter.

Value	DFS metadata Field
1	The **PrefixSize** and **Prefix** fields of DFSRootOrLinkIDBLOB.

Value	DFS metadata Field
(0x00000001)	
2 (0x00000002)	In addition to those for Level 1: the **CommentSize**, **Comment**, and **State** fields of DFSRootOrLinkIDBLOB and the **TargetCount** field of DFSTargetListBLOB.
3 (0x00000003)	In addition to those for Level 2: TargetEntryBLOB.
4 (0x00000004)	In addition to those for Level 3: the **ReferralTTL**field of DFSNamespaceRootBLOB or DFSNamespaceLinkBLOB; the **RootOrLinkGuid** field of DFSRootOrLinkIDBLOB.
5 (0x00000005)	All of the fields relevant to 4, excluding TargetEntryBLOB, the **RootOrLinkGuid** of the DFSRootOrLinkIDBLOB, the **Type** field of DFSRootOrLinkIDBLOB, and the **Size** of the value stored in the **pKT** attribute of the DFS namespace's AD DS object.
6 (0x00000006)	In addition to those for Level 4: TargetEntryBLOB.
7 (0x00000007)	The Value of **pKTGuid** attribute of the DFS namespace's AD DS object.
50 (0x00000032)	No metadata attribute stores this data. The value returned MUST, however, be one of the values defined in the table in section 2.2.3.10.
100 (0x00000064)	**CommentSize** and **Comment** in DFSRootOrLinkIDBLOB, and BLOBDataSize of DFSNamespaceLinkBLOB.

The server MUST use the following fields in the DFS metadata for a domainv2-based DFS namespace to return the required information, depending on the value of the *Level* parameter.

Value	DFS metadata Field
1 (0x00000001)	The **msDFS-LinkPathv2** attribute.
2 (0x00000002)	In addition to those for Level 1: msDFS-Commentv2, and the **State** and **TargetCount** fields of msDFS-TargetListv2.
3 (0x00000003)	In addition to those for Level 2: **msDFS-TargetListv2**.
4 (0x00000004)	In addition to those for Level 3: msDFS-Ttlv2, msDFS-NamespaceIdentityGUIDv2 for DFS root, and msDFS-LinkIdentityGUIDv2 for DFS link.
5 (0x00000005)	In addition to those for Level 4: msDFS-TargetListv2 and msDFS-Propertiesv2.
6 (0x00000006)	In addition to those for Level 4: msDFS-TargetListv2 and msDFS-Propertiesv2.
7 (0x00000007)	The value of the **msDFS-GenerationGUIDv2** attribute of the DFS namespace's AD DS object.

Value	DFS metadata Field
8 (0x00000008)	In addition to those for Level 5: msDFS-LinkSecurityDescriptorv2.
9 (0x00000009)	In addition to those for Level 6: msDFS-LinkSecurityDescriptorv2.
50 (0x00000032)	The **msDFS-SchemaMajorVersion** and **msDFS-SchemaMinorVersion** attributes.
100 (0x00000064)	The **msDFS-Commentv2** attribute.
150 (0x00000096)	The **msDFS-LinkSecurityDescriptorv2** attribute.

3.2.5.1.7 NetrDfsEnum (Opnum 5)

The **NetrDfsEnum** method enumerates the DFS roots hosted on a server or the DFS links of a namespace hosted by a server. Depending on the information level, the targets of these roots and links are also displayed.

The **NetrDfsEnum** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsEnum(
  [in] DWORD Level,
  [in] DWORD PrefMaxLen,
  [in, out, unique] DFS_INFO_ENUM_STRUCT* DfsEnum,
  [in, out, unique] DWORD* ResumeHandle
);
```

Level: This parameter specifies the information level of the data and, in turn, determines the action that the method performs. Clients MUST set this to one of the following values.

Value	Meaning
Level_1 0x00000001	Gets the name of the DFS root and all links beneath it. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_1** structures.
Level_2 0x00000002	Gets the name, comment, state, and number of targets for the DFS root and all links under the root. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_2** structures.
Level_3 0x00000003	Gets the name, comment, state, number of targets, and information about each target for the DFS root and all links under the root. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_3** structures.
Level_4 0x00000004	Gets the name, comment, state, time-out, GUID, number of targets, and information about each target for the DFS root and all links under the root. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_4** structures.
Level_5 0x00000005	Gets the name, comment, state, time-out, GUID, property flags, metadata size, and number of targets for a DFS root and all links under the root. In this case,

Value	Meaning
	NetrDfsEnum MUST point to an array of **DFS_INFO_5** structures.
Level_6 0x00000006	Gets the name, comment, state, time-out, GUID, property flags, metadata size, number of targets, and target information for a DFS root or DFS links. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_6** structures.
Level_8 0x00000008	Gets the name, comment, state, time-out, GUID, property flags, metadata size, and number of targets for a DFS root and all DFS links under the root. Also returns the security descriptor associated with each of the DFS links. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_8** structures.
Level_9 0x00000009	Gets the name, comment, state, time-out, GUID, property flags, metadata size, and number of targets for a DFS root and all DFS links under the root. Also returns the security descriptor associated with each of the DFS links. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_9** structures.
Level_300 0x0000012C	Enumerates the stand-alone and domain-based DFS roots hosted by the specified server. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_300** structures.

Any other value is invalid. If the *Level* parameter is not equal to one of these values, the server MUST fail the call.<70><71>

PrefMaxLen: This parameter MAY specify restrictions on the total size or number of elements returned. A value of 0xFFFFFFFF means there are no restrictions.<72>

DfsEnum: The pointer to a **DFS_INFO_ENUM_STRUCT** union to receive the returned information. The value of the *Level* parameter determines the case of the union.

ResumeHandle: This parameter MAY be used to continue an enumeration when more data is available than can be returned in a single invocation of this method.<73>

- If this parameter is not a null pointer, and the method returns ERROR_SUCCESS, this parameter receives an implementation-specific nonzero value that can be passed in subsequent calls to this method to continue the enumeration.
- If this parameter is a null pointer or points to a 0 value, the enumeration starts from the beginning.
- If this parameter is not a null pointer and points to a nonzero value returned in *ResumeHandle* by an earlier invocation of this method, the server will attempt to continue a previous enumeration, but MAY produce incomplete or inconsistent results due to the possibility of concurrent updates to the DFS namespace.

Return Values: The method MUST return 0 on success and a nonzero error code on failure. The server MUST return ERROR_NO_MORE_ITEMS (0x00000103) if there is no data to return. For protocol purposes, all other values transmitted in this field are implementation-specific and MUST be treated as equivalent failures .

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.
0x00000103	There is no data to return.

Return value/code	Description
ERROR_NO_MORE_ITEMS	

The response of the server depends on the value of the *Level* parameter. <74><75>

3.2.5.1.8 NetrDfsMove (Opnum 6)

The **NetrDfsMove (Opnum 6)** method renames or moves a DFS link. This method has the following MIDL syntax.

```
NET_API_STATUS NetrDfsMove(
  [in, string] WCHAR* DfsEntryPath,
  [in, string] WCHAR* NewDfsEntryPath,
  [in] unsigned long Flags
);
```

DfsEntryPath: The pointer to a DFS path, this parameter specifies the source path for the move operation. This MUST be a DFS link or the path prefix of any DFS link in the DFS namespace.

NewDfsEntryPath: The pointer to a DFS path, this parameter specifies the destination DFS path for the move operation. This MUST be a path or a DFS link in the same DFS namespace.

Flags: A bit field specifying the action to take when moving the link.

Value	Meaning
DFS_MOVE_FLAG_REPLACE_IF_EXISTS 0x00000001	If the destination path is an existing link, replace it as part of the move operation.

Return Values: The method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. All nonzero values MUST be handled as equivalent failures for protocol purposes.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The server MUST validate whether the source and destination paths are in the same DFS namespace, otherwise the server MUST fail the call.

The server SHOULD perform DFS link move operations atomically. That is, either all of the constituent operations are performed as part of the move and the call succeeds, or no changes are made to the DFS namespace and the call fails.<76>

If either the source or the destination path has wildcards ("*", "?") or relative pathname components (".", ".."), the server MAY fail the call.<77>

When the source and destination are both paths in the DFS namespace and not links themselves, all DFS links in the DFS namespace that have the source path as their prefix MUST be converted to DFS links with the destination path as the prefix. In effect, each DFS link that has the prefix specified by the source path is removed, and new DFS links that have exactly the same targets and target properties are added, but with the prefix specified by the destination path. For example, with a

source path of "\\MyServer\MyDfs\dir1" and a destination path of "\\MyServer\MyDfs\dir2", the DFS link "\\MyServer\MyDfs\dir1\link1" becomes "\\MyServer\MyDfs\dir2\link1", while the DFS link "\\MyServer\MyDfs\link2" is unaffected by the move operation.

If a DFS link already exists at the destination path, unless the *Flags* parameter is DFS_MOVE_FLAG_REPLACE_IF_EXISTS, the server MUST fail the call. This MUST be enforced only if the destination is an existing link, not if the destination is an existing file or directory. In the preceding example, if a DFS link "\\MyServer\MyDfs\dir2\link1" already exists, the move operation will fail unless the DFS_MOVE_FLAG_REPLACE_IF_EXISTS flags parameter is specified. If the DFS_MOVE_FLAG_REPLACE_IF_EXISTS flags parameter is specified, the DFS link at the destination path is removed and replaced by the moved DFS link. If "\\MySever\MyDfs\dir2\link1" is an existing directory and not a link, the operation does not require the DFS_MOVE_FLAG_REPLACE_IF_EXISTS flags parameter to be specified.

DFS servers SHOULD support the case of intermediate or leaf pathname components in the destination path being filed. For example, a server SHOULD support the case of a source path being "\\MyServer\MyDfs\dir1\link1", a destination path being "\\MyServer\MyDfs\comp1\link1", and "\\MyServer\MyDfs\comp1" being a file.<78>

If intermediate directories in the pathname of a source DFS link are empty, they SHOULD be removed, as required, after a move operation. For example, if "\\MyServer\MyDfs\dir1\dir2\link1" is moved to "\\MyServer\MyDfs\link1", the dir1 and dir2 directories are removed if they are empty.<79>

If the move operation results in a source DFS link becoming the prefix of an existing destination DFS link, the move operation MUST be failed. For example, if the source is a DFS link "\\MyServer\MyDfs\dir1\link1", the destination DFS link specified is "\\MyServer\MyDfs\dir2", and if a DFS link "\\MyServer\MyDfs\dir2\link2" already exists, the move operation fails.

For a domainv1-based DFS namespace and stand-alone DFS namespaces, the link's identity is changed. Thus, to another DFS root target of the same domainv1-based DFS namespace, one link is deleted and then another link is created instead of an existing link being moved.

The server MUST update the following fields in the DFS metadata for a domainv1-based DFS namespace.

Operation	DFS metadata changes required
Remove link	Remove DFSNamespaceLinkBLOB and update BLOBElementCount.
Add link	Add DFSNamespaceLinkBLOB and update BLOBElementCount.

In contrast, for a domainv2-based DFS namespace, the link's identity is not changed. Instead, the **msDFS-LinkPathv2**, **msDFS-LastModifiedv2**, and **msDFS-GenerationGUIDv2** attributes of a DFS link's LDAP entry are updated during a move operation. When a destination link is deleted, the required local state changes (on-disk, in-memory) are performed on the DFS root target server performing the move operation as well.

The server MUST update the following fields in the DFS metadata for a domainv2-based DFS namespace.

Operation	DFS metadata changes required
Remove link	Update **msDFS-LinkPathv2**, **msDFS-LastModifiedv2**, and **msDFS-GenerationGUIDv2**.
Add link	Update **msDFS-LinkPathv2**, **msDFS-LastModifiedv2**, and msDFS-GenerationGUIDv2.

The server MUST synchronously update the DFS metadata of a domain-based DFS namespace. If DFS root scalability mode is not enabled for the domain-based DFS namespace, the server MUST notify other DFS root targets of the change in DFS metadata by issuing a **NetrDfsSetInfo (Opnum 3)** method with the *Level* parameter 101, and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_RESYNCHRONIZE.<80>

For a domainv2-based DFS namespace, if the DFS root target server is switching to a new PDC, or if DFS root scalability mode is enabled but synchronization is done with a different DC other than the PDC, the server MUST notify other DFS root targets of the change in DFS metadata. This is done by issuing a **NetrDfsSetInfo (Opnum 3)** method with the Level parameter 101, and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_FORCE_SYNC, to ensure that a full synchronization is done instead of an incremental synchronization that picks up only changed data.

The move operation on a stand-alone DFS namespace or a domainv2-based DFS namespace also correctly applies to any security descriptor that is associated with the DFS link to the new reparse point created after the move operation.

3.2.5.1.9 NetrDfsAddRootTarget (Opnum 23)

The **NetrDfsAddRootTarget** method is used to create a stand-alone DFS namespace, a domainv1-based DFS namespace, or a domainv2-based DFS namespace.<81>

The **NetrDfsAddRootTarget** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsAddRootTarget(
  [in, unique, string] LPWSTR DfsPath,
  [in, unique, string] LPWSTR TargetPath,
  [in] ULONG MajorVersion,
  [in, unique, string] LPWSTR Comment,
  [in] BOOLEAN NewNamespace,
  [in] ULONG Flags
);
```

DfsPath: The pointer to a null-terminated Unicode string. This MUST be \\<domain>\<dfsname> for domain-based DFS or \\<server>\<share> for stand-alone DFS.

TargetPath: The pointer to a null-terminated Unicode string. This MUST be \\<server>\<share>[\<path>] for domain-based DFS or NULL for stand-alone DFS. The latter restriction is required to ensure that a typographic error in the AD DS domain name while attempting to create a domain-based DFS does not result in a stand-alone DFS namespace being created on the DFS root target server, if the first pathname component of the *DfsPath* parameter is used to detect whether a domain-based DFS namespace or stand-alone DFS namespace is being created.

MajorVersion: The DFS metadata version to use to create the DFS namespace. When adding a DFS root target to an existing DFS namespace, *MajorVersion* MUST be either 0 or the major version number of the existing DFS namespace. Otherwise, the call MUST fail.

Comment: The pointer to a null-terminated Unicode string that contains a comment associated with this root or link. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality. The comment MUST be ignored when adding a target to an existing link.

NewNamespace: A Boolean value that, if TRUE, indicates a request to create a new root. If FALSE, this value indicates a request to add a new root target to an existing root.

Flags: This parameter MUST be zero.

Return Values: The method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The following table summarizes the various actions that the **NetrDfsAddRootTarget** method takes based on the parameter values.

pDfsPath parameter	pTargetPath parameter	MajorVersion parameter	Explanation
\\<domain>\<dfsname>	\\<server>\<share>	1	Creates a new domainv1-based DFS namespace or adds a new root target to an existing domainv1-based DFS namespace. If a DFS namespace already exists, the version specified must match the DFS namespace; otherwise, the call fails.
\\<domain>\<dfsname>	\\<server>\<share>	2	Creates a new domainv1-based DFS namespace or adds a new root target to an existing domainv1-based DFS namespace. If a DFS namespace already exists, the version specified must match the DFS namespace; otherwise, the call fails.
\\<domain>\<dfsname>	\\<server>\<share>	0	Adds a new root target to an existing domain-based DFS namespace or a domainv2-based DFS namespace. If a DFS namespace does not already exist, the call fails.
\\<server>\<share>	NULL	1	Creates a new stand-alone DFS namespace.

The following scheme is used to create a new domainv2-based DFS namespace:

- **NetrDfsGetSupportedNamespaceVersion** is called to determine an appropriate version number to pass to the NetrDfsAddRootTarget() method.
- The client-side method creates a DFS metadata format-independent LDAP entry, called the DFS namespace anchor. It contains only the DFS metadata major version number.
- Updates the ACL on the AD DS object of the DFS namespace to permit read/write access by the DFS root target server.
- The client-side method then issues an RPC call to the DFS root target server.

- The DFS server creates a new DFS namespace LDAP entry with the DFS namespace anchor LDAP entry as its parent.
- All DFS links are created with the DFS namespace LDAP entry as the parent. For more information, see section 2.3.2.

This results in two LDAP entries in domainv2 corresponding to the single LDAP entry in domainv1.

3.2.5.1.10 NetrDfsRemoveRootTarget (Opnum 24)

The **NetrDfsRemoveRootTarget (Opnum 24)** method is the unified DFS namespace deletion method. It deletes stand-alone DFS namespaces, domainv1-based DFS namespaces, or domainv2-based DFS namespaces based on parameters specified.<82>

The **NetrDfsRemoveRootTarget (Opnum 24)** method has the following MIDL syntax.

```
NET_API_STATUS NetrDfsRemoveRootTarget(
  [in, unique, string] LPWSTR DfsPath,
  [in, unique, string] LPWSTR TargetPath,
  [in] ULONG Flags
);
```

DfsPath: The pointer to a null-terminated Unicode string. This MUST be \\<domain>\<dfsname> for domain-based DFS or \\<server>\<share> for stand-alone DFS.

TargetPath: The pointer to a null-terminated Unicode string. This MUST be \\<server>\<share>[\<path>] for domain-based DFS or NULL for stand-alone DFS.

Flags: The only supported bit in the *Flags* parameter is DFS_FORCE_REMOVE.

Value	Meaning
DFS_FORCE_REMOVE 0x80000000	Forcibly removes the DFS root target.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

This client-side method is responsible for deleting the DFS namespace anchor LDAP entry corresponding to a domainv2-based DFS namespace. This allows the client to be completely decoupled from the AD DS schema in use.

When a user who is not a domain administrator is granted administrator rights to the Dfs-Configuration container, the user must also be granted explicit rights to permit deletion of the DFS namespace anchor LDAP entry. The class schema for the domainv2-based DFS namespace anchor LDAP entry has a **defaultSecurityDescriptor** attribute granting appropriate rights to the creator-owner.

The following table summarizes the various actions that the **NetrDfsRemoveRootTarget** method takes based on the parameter values.

pDfsPath parameter	pTargetPath parameter	Explanation
\\<domain>\<dfsname>	\\<server>\<share>	Deletes a domain-based DFS root target. If the DFS root target removed is the last one for the DFS namespace, then it removes the DFS namespace itself. This parameter can be used for either a domainv1-based DFS namespace or a domaina2-based DFS namespace.
\\<domain>\<dfsname>	NULL	Deletes a Stand-Alone DFS Namespace.

3.2.5.1.11 NetrDfsGetSupportedNamespaceVersion (Opnum 25)

The **NetrDfsGetSupportedNamespaceVersion (Opnum 25)** method is used to determine the supported DFS metadata version number.<83>

The **NetrDfsGetSupportedNamespaceVersion (Opnum 25)** method has the following MIDL syntax.

```
NET_API_STATUS NetrDfsGetSupportedNamespaceVersion(
  [in] DFS_NAMESPACE_VERSION_ORIGIN Origin,
  [in, unique, string] NETDFS_SERVER_OR_DOMAIN_HANDLE pName,
  [out] PDFS_SUPPORTED_NAMESPACE_VERSION_INFO pVersionInfo
);
```

Origin: This parameter lists the maximum version that the client, server, or AD DS domain can support.

pName: The pointer to a null-terminated Unicode string. This is the host name of the server to which the RPC method is issued.

pVersionInfo: The pointer to a **DFS_SUPPORTED_NAMESPACE_VERSION_INFO** structure to receive the DFS metadata version number determined.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The version number of the DFS metadata that can be used for a new DFS namespace depends on the following:

- For domain-based DFS namespaces, the version supported by the DFS metadata schema in use in the AD DS domain.
- The version supported by the server that is to host the DFS root target.

Thus, the DFS metadata version that can be effectively used for creating a new DFS namespace is the minimum version that the AD DS domain and the server support. This can be determined by calling this method with the *pName* parameter set to the name of the server that is to host the new DFS root target, and the *Origin* parameter set to DFS_NAMESPACE_VERSION_ORIGIN_COMBINED.

This method is useful in determining an appropriate version number to pass to the **NetrDfsAddRootTarget (Opnum 23)** method.

3.2.5.2 Extended Methods

3.2.5.2.1 NetrDfsAdd2 (Opnum 19)

The **NetrDfsAdd2 (Opnum 19)** method creates a new DFS link or adds a new target to an existing link of a DFS namespace.<84>

The **NetrDfsAdd2** method has the following MIDL syntax.

```
NET_API_STATUS NetrDfsAdd2(
  [in, string] WCHAR* DfsEntryPath,
  [in, string] WCHAR* DcName,
  [in, string] WCHAR* ServerName,
  [in, unique, string] WCHAR* ShareName,
  [in, unique, string] WCHAR* Comment,
  [in] DWORD Flags,
  [in, out, unique] DFSM_ROOT_LIST** ppRootList
);
```

DfsEntryPath: The pointer to a DFS link path that contains the name of an existing link when additional link targets are added or the name of a new link is created.

DcName: The pointer to a null-terminated Unicode string. For a domain-based DFS namespace, this is the host name of the DC that the DFS root target uses to get or update DFS metadata for the DFS namespace. This parameter MAY be a null pointer; otherwise, it MUST be the host name of the PDC for the AD domain of the DFS namespace.

ServerName: The pointer to a null-terminated Unicode string that specifies the DFS link target host name.

ShareName: The pointer to a null-terminated Unicode DFS link target share name string. This may also be a share name with a path relative to the share, for example, share1\mydir1\mydir2. When specified this way, each pathname component MUST be a directory.

Comment: The pointer to a null-terminated Unicode string that contains a comment associated with this root or link. This string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality. The comment MUST be ignored when adding a target to an existing link.

Flags: This is the flag that indicates the operation to perform. The following table lists the possible values.

Value	Meaning
0x00000000	Creates a new link or adds a new target to an existing link.

Value	Meaning
DFS_ADD_VOLUME 0x00000001	Creates a new link in the DFS namespace if it does not exist, or fails if it exists.
DFS_RESTORE_VOLUME 0x00000002	Adds a target without verifying its existence.

ppRootList: When the RPC method is successful, the server returns a list of DFS root targets in the domain-based DFS namespace. The client MUST notify each root target in the list of the change in the DFS metadata for the DFS namespace. The list MAY be empty if the server has itself performed the notification.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The server MUST verify the existence of the DFS namespace that the *DfsEntryPath* parameter specifies. If the namespace does not exist, the server MUST return a failure.

If the link to be added already exists, a new link target is added. If the new link target already exists, the call MUST fail.

The server SHOULD support the creation of a new link without requiring the DFS_ADD_VOLUME *Flags* parameter.<85>

The server MUST update the same fields in the DFS metadata for a domain-based DFS namespace as in the NetrDfsAdd method, as specified in section 3.1.5.1.1.

The server MUST synchronously update the DFS metadata of a domain-based DFS namespace.

If DFS root scalability mode is not enabled for the domain-based DFS namespace, the server MUST do one of the following:

- Notify other DFS root targets of the change in DFS metadata by issuing a **NetrDfsSetInfo** method with the *Level* parameter 101 and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_RESYNCHRONIZE.
- Return a list of DFS root targets to the client in the *ppRootList* parameter.

For a domainv2-based DFS namespace, if the DFS root target server is switching to a new PDC, or if DFS root scalability mode is enabled, but synchronization is done with a different DC other than the PDC, the server MUST notify other DFS root targets of the change in DFS metadata. This is done by issuing a **NetrDfsSetInfo** method with the *Level* parameter 101 and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_FORCE_SYNC to ensure that a full synchronization is done instead of an incremental synchronization that picks up only changed data.

3.2.5.2.2 NetrDfsRemove2 (Opnum 20)

The **NetrDfsRemove2 (Opnum 20)** method removes the specified link or link target.<86>

The **NetrDfsRemove2** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsRemove2(
  [in, string] WCHAR* DfsEntryPath,
  [in, string] WCHAR* DcName,
  [in, unique, string] WCHAR* ServerName,
  [in, unique, string] WCHAR* ShareName,
  [in, out, unique] DFSM_ROOT_LIST** ppRootList
);
```

DfsEntryPath: The pointer to a DFS link path that contains the name of the DFS link to remove.

DcName: The pointer to a null-terminated Unicode string. For a domain-based DFS namespace, this string contains the host name of the DC to be used by the DFS root target that is removing the DFS link. This parameter MAY be a null pointer; otherwise it MUST be the PDC for the AD domain of the DFS namespace.

ServerName: The pointer to a null-terminated Unicode DFS link target host name string. This MUST be a null pointer when the link and all of the link targets are to be removed.

ShareName: The pointer to a null-terminated Unicode DFS link target share name string. This MUST be a null pointer when the link and all of the link targets are to be removed.

ppRootList: When the RPC method is successful, the server MUST return a list of DFS root targets in the domain-based DFS namespace that the client MUST notify of the change in the DFS metadata for the DFS namespace. The list MAY be empty if the server has itself performed the notification.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The server MUST verify the existence of the DFS namespace that the *DfsEntryPath* parameter specifies. If the namespace does not exist, the server MUST return a failure.

If the *ServerName* and *ShareName* parameters are both NULL, the server MUST remove the DFS link itself; otherwise the server MUST remove the specified link target. If the removed link target is the last target of the link, the server MUST also remove the DFS link. The server MUST support removing a link and all of its targets without requiring the client to remove one target at a time from the link.

The server MUST update the same fields in the DFS metadata for a domain-based DFS namespace, as specified in the **NetrDfsRemove** method.

The server MUST synchronously update the DFS metadata of a domain-based DFS namespace.

If DFS root scalability mode is not enabled for the domain-based DFS namespace, the server MUST do one of the following:

- Notify other DFS root targets of the change in DFS metadata by issuing a **NetrDfsSetInfo** method with the *Level* parameter 101, and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_RESYNCHRONIZE.
- Return a list of DFS root targets to the client in the *ppRootList* parameter.

For a domainv2-based DFS namespace, if the DFS root target server is switching to a new PDC, or if DFS root scalability mode is enabled but synchronization is done with a different DC other than the PDC, the server MUST notify other DFS root targets of the change in DFS metadata. This is done by issuing a **NetrDfsSetInfo** method with the *Level* parameter 101 and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_FORCE_SYNC to ensure that a full synchronization is done instead of an incremental synchronization that picks up only changed data.

3.2.5.2.3 NetrDfsEnumEx (Opnum 21)

The **NetrDfsEnumEx (Opnum 21)** method enumerates the DFS roots hosted on a server, or DFS links of a namespace hosted by the server.<87><88> Depending on the information level, the targets associated with the roots and links are also displayed.

The **NetrDfsEnumEx** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsEnumEx(
  [in, string] WCHAR* DfsEntryPath,
  [in] DWORD Level,
  [in] DWORD PrefMaxLen,
  [in, out, unique] DFS_INFO_ENUM_STRUCT* DfsEnum,
  [in, out, unique] DWORD* ResumeHandle
);
```

DfsEntryPath: The pointer to an AD DS domain name, a host name, or a DFS path, depending on the *Level* parameter.

- An AD DS domain name MUST be a null-terminated Unicode string in the following form:

  ```
  <ADDSDomainName>
  ```

  where <ADDSDomainName> is the AD DS domain name to use for the enumeration.
- A host name MUST be a null-terminated Unicode string in the following form:

  ```
  \\<ServerName>
  ```

  where <ServerName> is the host name to which the RPC method is to be issued.
- A DFS link path.

Level: This parameter specifies the information level of the data and, in turn, determines the action the method performs. Clients MUST set this to one of the following values:

Value	Meaning
Level 1 0x00000001	Gets the name of the DFS root and all links beneath it. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_1** structures.
Level 2 0x00000002	Gets the name, comment, state, and number of targets for the DFS root and all links under the root. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_2** structures.
Level 3 0x00000003	Gets the name, comment, state, number of targets, and information about each target for the DFS root and all links under the root. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_3** structures.
Level 4 0x00000004	Gets the name, comment, state, time-out, GUID, number of targets, and information about each target for the DFS root and all links under the root. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_4** structures.
Level 5 0x00000005	Gets the name, comment, state, time-out, GUID, property flags, metadata size, and number of targets for a DFS root and all links under the root. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_5** structures.
Level 6 0x00000006	Gets the name, comment, state, time-out, GUID, property flags, metadata size, number of targets, and target information for a root or link. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_6** structures.
Level 8 0x00000008	Gets the name, comment, state, time-out, GUID, property flags, metadata size, and number of targets for a DFS root and all links under the root. Also returns the security descriptor associated with each of the DFS links. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_8** structures.
Level 9 0x00000009	Gets the name, comment, state, time-out, GUID, property flags, metadata size, and number of targets for a DFS root and all links under the root. Also returns the security descriptor associated with each of the DFS links. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_9** structures.
Level 200 0x000000C8	Enumerates all of the domain-based DFS namespace in a domain. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_200** structures.
Level 300 0x0000012C	Enumerates the stand-alone and domain-based DFS roots that the specified server hosts. In this case, **NetrDfsEnum** MUST point to an array of **DFS_INFO_300** structures.

Any other value is invalid.<89> If the *Level* parameter is not equal to one of the valid values, the server MUST fail the call.<90>

PrefMaxLen: This parameter MAY specify restrictions on the total size or number of elements returned. A value of 0xFFFFFFFF means there are no restrictions.<91>

DfsEnum: The pointer to a **DFS_INFO_ENUM_STRUCT** union to receive the returned information. The *Level* parameter value determines the case of the union.

ResumeHandle: This parameter MAY be used to continue an enumeration when more data is available than can be returned in a single invocation of this method.<92>

- If this parameter is not a null pointer, and the method returns ERROR_SUCCESS, this parameter receives an implementation-specific nonzero value that can be passed in subsequent calls to this method to continue the enumeration.

- If this parameter is a null pointer, or it points to a zero value, the enumeration starts from the beginning.
- If this parameter is not a null pointer, and it points to a nonzero value returned in *ResumeHandle* by an earlier invocation of this method, the server will attempt to continue a previous enumeration.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The server MUST return ERROR_NO_MORE_ITEMS (0x00000103) if there is no data to return. For protocol purposes, all other values transmitted in this field are implementation-specific and MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.
0x00000103 ERROR_NO_MORE_ITEMS	There is no data to return.

Unlike the **NetrDfsEnum** method, this method can be used even when the server is hosting more than one DFS root.

The response of the server depends on the value of the *Level* parameter.

The server MUST get the required information from the DFS metadata of the DFS namespace in the same way as the **NetrDfsEnum** method, as specified in section 3.2.5.1.7.

3.2.5.2.4 NetrDfsSetInfo2 (Opnum 22)

The **NetrDfsSetInfo2 (Opnum 22)** method sets or modifies the information associated with a DFS root, a DFS root target, a DFS link, or a DFS link target.<93><94>

The **NetrDfsSetInfo2** method has the following MIDL syntax.

```
NET_API_STATUS NetrDfsSetInfo2(
  [in, string] WCHAR* DfsEntryPath,
  [in, string] WCHAR* DcName,
  [in, unique, string] WCHAR* ServerName,
  [in, unique, string] WCHAR* ShareName,
  [in] DWORD Level,
  [in, switch_is(Level)] DFS_INFO_STRUCT* pDfsInfo,
  [in, out, unique] DFSM_ROOT_LIST** ppRootList
);
```

DfsEntryPath: The pointer to a DFS root path or a DFS link path that contains the name of a DFS root or DFS link name.

DcName: The pointer to a null-terminated Unicode string. For a domain-based DFS namespace, this string contains the host name of the DC that the DFS root target uses to get or update DFS metadata for the DFS namespace. This parameter MAY be a null pointer; otherwise, it MUST be the PDC for the AD domain of the DFS namespace.

ServerName: The pointer to a null-terminated Unicode DFS root target or a DFS link target host name string. This parameter MUST be a null pointer if the operation is intended for a DFS root or a DFS link and not for targets.

ShareName: The pointer to a null-terminated Unicode DFS root target or a DFS link target share name string. This parameter MUST be a null pointer if the operation is intended for a DFS root or a DFS link and not for targets.

Level: This parameter specifies the information level of the data and, in turn, determines the action the method performs. Clients MUST set this to one of the following values.

Value	Meaning
Level_100 0x00000064	Sets the comment associated with the root or link that specified in *DfsInfo*. In this case, *DfsInfo* MUST point to a **DFS_INFO_100** structure.
Level_101 0x00000065	Sets the storage state associated with the root or link that the *DfsInfo* parameter specifies. In this case, *DfsInfo* MUST point to a **DFS_INFO_101** structure.<95>
Level_102 0x00000066	Sets the time-out value associated with the root or link that the *DfsInfo* parameter specifies. In this case, *DfsInfo* MUST point to a **DFS_INFO_102** structure.
Level_103 0x00000067	Sets the property flags for the root or link that the *DfsInfo* parameter specifies. In this case, *DfsInfo* MUST point to a **DFS_INFO_103** structure.
Level_104 0x00000068	Sets the target priority rank and class for the root target or link target that the *DfsInfo* parameter specifies. In this case, *DfsInfo* MUST point to a **DFS_INFO_104** structure.
Level_105 0x00000069	Sets the comment, state, time-out information, and property flags for the namespace root or link that the *DfsInfo* parameter specifies. This does not apply to the root target or link target. In this case, *DfsInfo* MUST point to a **DFS_INFO_105** structure.
Level_106 0x0000006A	Sets the target state and priority for the root target or link target that the *DfsInfo* parameter specifies. This does not apply to the DFS namespace root or link. *DfsInfo* MUST point to a **DFS_INFO_106** structure.<96>
Level_107 0x0000006B	Sets the comment, state, time-out, security descriptor information, and property flags for the namespace root or link that the *DfsInfo* parameter specifies. This does not apply to the root target or link target. In this case, *DfsInfo* MUST point to a **DFS_INFO_107** structure. The *ServerName* and *ShareName* parameters MUST be null. The security descriptor MUST NOT have an owner, group, or SACLs in it.
Level_150 0x00000096	Sets the security descriptor associated with a DFS link. Only stand-alone DFS namespaces and domainv2-based DFS namespaces are supported. The *ServerName* and *ShareName* parameters must both be NULL. When using **NetrDfsSetInfo ()**, the security descriptor MUST NOT have an owner, group, or SACLs in it.

Any other value is invalid.<97> If the *Level* parameter is not equal to one of the valid values, the server MUST fail the call.<98>

pDfsInfo: The pointer to a **DFS_INFO_STRUCT** union that contains the specified data. The *Level* parameter value determines the case of the union.

ppRootList: When the RPC method is successful, the server returns a list of DFS root targets in the domain-based DFS namespace that the client MUST notify of the change in the DFS

metadata for the DFS namespace. The list MAY be empty if the server has itself performed the notification.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The server MUST verify the existence of the DFS namespace that the *DfsEntryPath* parameter specifies. If the namespace does not exist, the server MUST return a failure.

If the *DcName* parameter is not null, it MUST be the PDC for the AD DS domain of the domain-based DFS namespace.

The server MUST fail any attempt to set the state of a DFS root, a DFS link, a DFS root target, or a DFS link target to a value that is not specified. The server MUST fail any attempt to set the property flags on a DFS link that are defined only for a DFS root.

With the *Level* parameter 101 and the **State** field in the **DFS_INFO_101** structure as DFS_VOLUME_STATE_RESYNCHRONIZE, the server MUST reload the contents of the DFSMetadataCache, if maintained, for the domain-based DFS namespace that the *ShareName* parameter specifies. In the case of both domain-based DFS namespaces and stand-alone DFS namespaces, the server MUST check the DFS namespace it hosts locally with the DFS metadata and perform any required modifications.

With the *Level* parameter 101 and the **State** field in the **DFS_INFO_101** structure as DFS_VOLUME_STATE_FORCE_SYNC, the server MUST perform a full synchronization instead of an incremental synchronization to reload the contents of the DFSMetadataCache and to identify added or deleted DFS links. This **State** field is supported on domainv2-based DFS namespaces and stand-alone DFS namespaces.

With the *Level* parameter 107, **NetrDfsSetInfo2 (Opnum 22)** can be used for a stand-alone DFS namespace or a domainv2-based DFS namespace root. It can also be used for a domainv1-based DFS namespace root or a domainv1-based DFS link, where the *pSecurityDescriptor* parameter MUST be NULL. It can also be used for a domainv2-based DFS link without requiring that the *pDescriptor* be NULL.

The server MUST update the same fields in the DFS metadata for a domain-basedv1 DFS namespace as for the **NetrDfsSetInfo (Opnum 3)** method, as specified in section 3.2.5.1.5.

The server MUST synchronously update the DFS metadata of a domain-based DFS namespace.

If DFS root scalability mode is not enabled for the domain-based DFS namespace, the server MUST do one of the following:

- Notify other DFS root targets of the change in DFS metadata by issuing a **NetrDfsSetInfo (Opnum 3)** method with the *Level* parameter 101, and with the **State** field of **DFS_INFO_1** set to VOLUME_STATE_RESYNCHRONIZE.
- Return a list of DFS root targets to the client in the *ppRootList* parameter.

For a domainv2-based DFS namespace, if the DFS root target server is switching to a new PDC, or if DFS root scalability mode is enabled but synchronization is done with a different DC other than the

PDC, the server MUST notify other DFS root targets of the change in DFS metadata. This is done by issuing a **NetrDfsSetInfo** method with the *Level* parameter 101, and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_FORCE_SYNC to ensure that a full synchronization is performed instead of an incremental synchronization that picks up only changed data.

3.2.5.3 Root Target Methods

3.2.5.3.1 NetrDfsAddFtRoot (Opnum 10)

The **NetrDfsAddFtRoot (Opnum 10)** method creates a new domainv1-based DFS namespace or adds a root target to an existing namespace.

The **NetrDfsAddFtRoot** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsAddFtRoot(
  [in, string] WCHAR* ServerName,
  [in, string] WCHAR* DcName,
  [in, string] WCHAR* RootShare,
  [in, string] WCHAR* FtDfsName,
  [in, string] WCHAR* Comment,
  [in, string] WCHAR* ConfigDN,
  [in] BOOLEAN NewFtDfs,
  [in] DWORD ApiFlags,
  [in, out, unique] DFSM_ROOT_LIST** ppRootList
);
```

ServerName: The pointer to a null-terminated Unicode string. This is the host name of the new DFS root target.

DcName: The pointer to a null-terminated Unicode string. For a domainv1-based DFS namespace, this string contains the host name of the DC that the new DFS root target is to use to get or update DFS metadata for the DFS namespace. This parameter MAY be a null pointer; otherwise, it MUST be the PDC for the AD domain of the DFS namespace.

RootShare: The pointer to a null-terminated Unicode string. This is the new DFS root target share name. This may be different from the *FtDfsName* parameter. The share MUST already exist.

FtDfsName: The pointer to a null-terminated Unicode string. This is the name of the new or existing domain-based DFS namespace.

Comment: The pointer to a null-terminated Unicode string that contains a comment associated with the DFS namespace. Used for informational purposes, this string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality. This parameter MAY be null.

ConfigDN: This MUST be a null pointer.

NewFtDfs: A Boolean value that, if TRUE, indicates a request to create a new root. If FALSE, this value indicates a request to add a new root target to an existing root.

ApiFlags: This parameter MUST be 0.

ppRootList: When the RPC method is successful, the server returns a list of DFS root targets in the domain-based DFS namespace that the client MUST notify of the change in the DFS

metadata for the DFS namespace. The list MAY be empty if the server has itself performed the notification.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The share that the *RootShare* parameter specifies MUST already exist on the server.

If the *DcName* parameter is not null, the server assumes that this is the PDC for the AD DS domain in which the DFS namespace is to be created.

If the domain-based DFS namespace already exists, and the *ServerName* and *RootShare* parameters are a root target, the server MUST fail with ERROR_ALREADY_EXISTS.

The server MUST update the following fields in the domainv1-based DFS metadata.

Operation	DFS metadata changes required
Adding a new namespace	Creates new DFS metadata.
Adding a new root target.	Updates the **TargetCount** field of the DFSTargetListBLOB, creates a new TargetEntryBLOB, updates the DFSTargetListBLOBSize, updates the BLOBDataSize of the DFSNamespaceRootBLOB, and adds the DFSRootTarget to the **remoteServerName** attribute in the AD DS object.

The server MUST synchronously update the DFS metadata. If DFS root scalability mode is not enabled, the server MUST do one of the following:

- Notify other DFS root targets of the change in DFS metadata by issuing a **NetrDfsSetInfo (Opnum 3)** method with the *Level* parameter 101 and with the **State** field of **DFS_INFO_101** set to VOLUME_STATE_RESYNCHRONIZE.
- Return a list of DFS root targets to the client in the *ppRootList* parameter.<99>

3.2.5.3.2 NetrDfsRemoveFtRoot (Opnum 11)

The **NetrDfsRemoveFtRoot (Opnum 11)** method removes the specified root target from a domainv1-based DFS namespace.<100> If the target is the last one associated with the DFS namespace, this method also deletes the DFS namespace. The DFS namespace can be removed without first removing all of the links in it.

If a client tries to use this method on a domainv2-based DFS namespace target, the server MUST fail with the return value of ERROR_NOT_SUPPORTED.

The **NetrDfsRemoveFtRoot** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsRemoveFtRoot(
  [in, string] WCHAR* ServerName,
  [in, string] WCHAR* DcName,
```

```
  [in, string] WCHAR* RootShare,
  [in, string] WCHAR* FtDfsName,
  [in] DWORD ApiFlags,
  [in, out, unique] DFSM_ROOT_LIST** ppRootList
);
```

ServerName: The pointer to a null-terminated Unicode string. This is the host name DFS root target to be removed.

DcName: The pointer to a null-terminated Unicode string. For a domainv1-based DFS namespace, this string contains the host name of the DC to be used by the DFS root targets being removed for getting or updating DFS metadata for the DFS namespace. This parameter MAY be a null pointer; otherwise, it MUST be the PDC for the AD domain of the DFS namespace.

RootShare: The pointer to a null-terminated Unicode DFS root target share name string. The share is not removed automatically when the method is successful; it MUST be removed explicitly as needed.

FtDfsName: The pointer to a null-terminated Unicode string that contains the DFS namespace in which the operation is to be performed. It MAY be different from the *RootShare*.

ApiFlags: This parameter MUST be 0 when issued to a **member server**.

ppRootList: When the RPC method is successful, the server returns a list of DFS root targets in the domain-based DFS namespace that the client MUST notify of the change in the DFS metadata for the DFS namespace. The list MAY be empty if the server has itself performed the notification.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

The server MUST support deleting a DFS namespace without first requiring removal of all the DFS links in it.

If the *DcName* parameter is not null, it MUST be the PDC for the AD domain of the DFS namespace.

The server MUST verify the existence of the DFS namespace that the *DfsEntryPath* parameter specifies. If the namespace does not exist, the server MUST return a failure.

The server MUST remove itself as a root target of the domain-based DFS namespace specified by the *ServerName* and *RootShare* parameters by updating the **remoteServerName** attribute of the namespace's AD DS object (as specified in section 2.3.3) in the root target. If the last DFS root target is being removed, the server SHOULD NOT remove the AD DS object of the namespace; this MUST be done by the client invoking the method.<101>

This RPC method does not automatically remove the share backing the DFS namespace.

The server MUST update the following fields in the domainv1-based DFS metadata.

Operation	DFS metadata changes required
Remove a namespace.	Removes the AD DS object of DFS namespace.
Remove a root target.	Updates the **TargetCount** in the existing DFSTargetListBLOB, removes the TargetEntryBLOB, updates the DFSTargetListBLOBSize, updates the BLOBDataSize of the DFSNamespaceRootBLOB, and removes the root target from the **remoteServerName** attribute in the AD DS object.

The server MUST synchronously update the DFS metadata of the namespace. If DFS root scalability mode is not enabled, the server MUST do one of the following:

- Notify other DFS root targets of the change in DFS metadata by issuing a **NetrDfsSetInfo (Opnum 3)** method with the *Level* parameter 101 and the **State** field of **DFS_INFO_101** set to VOLUME_STATE_RESYNCHRONIZE.
- Return a list of DFS root targets to the client in the *ppRootList* parameter.<102>

3.2.5.3.3 NetrDfsFlushFtTable (Opnum 18)

The **NetrDfsFlushFtTable (Opnum 18)** method instructs the DFS server on a DC to purge the specified domainv1-based DFS entry from any DFS root referral cache it may have.

The **NetrDfsFlushFtTable** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsFlushFtTable(
  [in, string] WCHAR* DcName,
  [in, string] WCHAR* wszFtDfsName
);
```

DcName: The pointer to a null-terminated Unicode string that contains the host name of the DC to which the RPC method is issued.

wszFtDfsName: The pointer to a null-terminated Unicode string that contains the name of the domain-based DFS namespace.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

3.2.5.4 Stand-Alone Namespace Methods

3.2.5.4.1 NetrDfsAddStdRoot (Opnum 12)

The **NetrDfsAddStdRoot (Opnum 12)** method creates a new stand-alone DFS namespace.<103> <104>

The **NetrDfsAddStdRoot** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsAddStdRoot(
  [in, string] WCHAR* ServerName,
  [in, string] WCHAR* RootShare,
  [in, string] WCHAR* Comment,
  [in] DWORD ApiFlags
);
```

ServerName: The pointer to a null-terminated Unicode string. This is the host name of the new DFS root target. This parameter MAY be a null pointer.

RootShare: The pointer to a null-terminated Unicode string. This is the new DFS root target share name as well as the DFS namespace name. The share MUST already exist.

Comment: The pointer to a null-terminated Unicode string that contains a comment associated with the DFS namespace. Used for informational purposes, this string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality. This parameter MAY be a null pointer.

ApiFlags: This parameter MUST be 0.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.
0x00000050 ERROR_FILE_EXISTS	The DFS namespace that the *ServerName* and *RootShare* parameters specify already exists.
0x00000906 NERR_NetNameNotFound	The share that the *RootShare* parameter specifies does not already exist.

On receiving this method, the server MUST do the following:

- If the DFS namespace that the *ServerName* and *RootShare* parameters specify already exists, the RPC method fails with ERROR_FILE_EXISTS (0x00000050).
- If the share that the *RootShare* parameter specifies does not already exist, the RPC method fails with NERR_NetNameNotFound (0x00000906).
- Create the stand-alone DFS namespace and initialize the DFS metadata in an implementation-defined manner.

3.2.5.4.2 NetrDfsRemoveStdRoot (Opnum 13)

The **NetrDfsRemoveStdRoot (Opnum 13)** method deletes the specified stand-alone DFS namespace.<105> The DFS namespace can be removed without first removing all of the links in it.

The **NetrDfsRemoveStdRoot** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsRemoveStdRoot(
  [in, string] WCHAR* ServerName,
```

```
  [in, string] WCHAR* RootShare,
  [in] DWORD ApiFlags
);
```

ServerName: The pointer to a null-terminated Unicode string. This is the host name of the DFS root target to be removed.

RootShare: The pointer to a null-terminated Unicode DFS root target share name string. This is also the DFS namespace name. The share is not removed automatically when the method is successful; it MUST be removed explicitly, as needed.

ApiFlags: This parameter MUST be 0.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.
0x00000490 ERROR_NOT_FOUND	The DFS namespace that the *ServerName* and *RootShare* parameters specify does not already exist.

The server MUST support the removal of a DFS namespace without requiring that all of the DFS links be removed first. This RPC method does not automatically remove the share backing the DFS namespace.

On receiving this method, the server MUST do the following:

- If the DFS namespace that the *ServerName* and *RootShare* parameters specify does not already exist, the RPC method fails with ERROR_NOT_FOUND (0x00000490).
- Remove the DFS namespace in an implementation-defined manner.

3.2.5.4.3 NetrDfsAddStdRootForced (Opnum 15)

The **NetrDfsAddStdRootForced (Opnum 15)** method creates a new stand-alone DFS namespace without checking for the availability and accessibility of the specified share.<106><107><108>

The **NetrDfsAddStdRootForced** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsAddStdRootForced(
  [in, string] WCHAR* ServerName,
  [in, string] WCHAR* RootShare,
  [in, string] WCHAR* Comment,
  [in, string] WCHAR* Share
);
```

ServerName: The pointer to a null-terminated Unicode string. This is the host name of the new DFS root target.

RootShare: The pointer to a null-terminated Unicode DFS root target share name string. This is also the DFS namespace name. This method does not create the share; it MUST be created separately.

Comment: The pointer to a null-terminated Unicode string that contains a comment associated with the DFS namespace. Used for informational purposes, this string has no protocol-specified restrictions on length or content. The comment is meant for human consumption and does not affect server functionality. This parameter MAY be a null pointer.

Share: The pointer to a null-terminated Unicode string that contains the local file system path corresponding to the share on the server receiving the RPC method, in the form:

```
<X>:\<path>
```

where <*X*> is a drive letter (a single character from A to Z) and <*path*> is a file system path whose leaf component is a directory.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x000000 ERROR_SUCCESS	Successful completion.

The support for this method is optional. If supported, the server MUST support the ability to create a DFS namespace even when the share that the *RootShare* parameter specifies is not available or accessible.

On receiving this method, the server MUST do the following:

- If the DFS namespace that the *ServerName* and *RootShare* parameters specify already exists, the RPC method fails with ERROR_FILE_EXISTS (0x00000050).
- Create the stand-alone DFS namespace and initialize the DFS metadata in an implementation-defined manner.

3.2.5.5 Domain-Based Namespace Methods

3.2.5.5.1 NetrDfsGetDcAddress (Opnum 16)

The **NetrDfsGetDcAddress (Opnum 16)** method returns the DC host name used by the DFS server to which the RPC method is issued.<109> The client MUST use this DC to create a domain-based DFS namespace, add a root target to a domain-based DFS namespace, remove a root target from a domain-based DFS namespace, or remove a domain-based DFS namespace.

The **NetrDfsGetDcAddress** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsGetDcAddress(
  [in, string] WCHAR* ServerName,
  [in, out, string] WCHAR** DcName,
```

```
  [in, out] BOOLEAN* IsRoot,
  [in, out] unsigned long* Timeout
);
```

ServerName: The pointer to a null-terminated Unicode string. This is the host name of the server to which the RPC method is issued.

DcName: When the **NetrDfsGetDcAddress** method is successful, the DFS server returns a pointer to a null-terminated Unicode DC host name string.<110>

IsRoot: The pointer to a Boolean, set to TRUE on return if the server hosts any DFS root target and FALSE otherwise.<111>

Timeout: The pointer to an unsigned long. Set to the time period, in seconds, that the server will use the DC that is returned for its DFS metadata accesses.<112>

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

A server MUST implement this method if it supports domain-based DFS namespaces.

In the *DcName* parameter, the server SHOULD return the host name of the DC it is using to access DFS metadata for any domain-based DFS namespace that the server is hosting. If the server is not currently using a DC, it MUST determine a DC and return its name.

The server SHOULD also return a time-out value, in seconds, that is the length of time that the server will be using the DC in the *Timeout* parameter, assuming that another RPC method does not change it.

The server uses the *IsRoot* parameter to specify whether whether supports the ability to host more than one DFS namespace, and also to indicate if it is currently hosting a DFS namespace. If the server supports the ability to host more than one DFS namespace, it MUST return FALSE in the *IsRoot* parameter, regardless of whether it is actually hosting a DFS namespace. If the server does not support the ability to host more than one DFS namespace, then if it is actually hosting a DFS namespace, it SHOULD return TRUE in the *IsRoot* parameter; otherwise, it should return FALSE.

3.2.5.5.2 NetrDfsSetDcAddress (Opnum 17)

The **NetrDfsSetDcAddress (Opnum 17)** method instructs the server receiving the RPC method to use the specified DC for DFS metadata accesses for domain-based DFS namespaces.<113>

The **NetrDfsSetDcAddress** method uses the following MIDL syntax.

```
NET_API_STATUS NetrDfsSetDcAddress(
  [in, string] WCHAR* ServerName,
  [in, string] WCHAR* DcName,
  [in] DWORD Timeout,
  [in] DWORD Flags
```

```
);
```

ServerName: The pointer to a null-terminated Unicode string. This is the host name of the server to which the RPC method is issued.

DcName: the pointer to a null-terminated Unicode DC host name string.

Timeout: The time period, in seconds, that the server uses the specified DC when storing and retrieving domain-based DFS metadata. This is valid only when the NET_DFS_SETDC_TIMEOUT bit of the *Flags* parameter is set.

Flags: the bit field specifying additional operations to perform. Possible values are as follows:

Value	Meaning
NET_DFS_SETDC_FLAGS 0x00000000	Indicates that no additional operation is requested.
NET_DFS_SETDC_TIMEOUT 0x00000001	Sets the time-out value based on the *Timeout* parameter.
NET_DFS_SETDC_INIT_PKT 0x00000002	Instructs the called server to reload its DFS metadata from the specified DC.

Return Values: This method MUST return 0 on success and a nonzero error code on failure. The values transmitted in this field are implementation-specific. For protocol purposes, all nonzero values MUST be treated as equivalent failures.

Return value/code	Description
0x00000000 ERROR_SUCCESS	Successful completion.

Servers MAY choose not to implement this method. If the server implements this function, the server MUST update the DC it uses for accessing DFS metadata for the domain-based DFS namespace it is hosting with the specified DC. If no time-out is specified in the *Timeout* parameter (NET_DFS_SETDC_TIMEOUT is not set in the *Flags* parameter), the server MUST use its default time-out. The DC the server should use at the end of this time-out is implementation-defined.

When NET_DFS_SETDC_INIT_PKT is set in the *Flags* parameter, the server SHOULD initiate a background synchronization of the domain-based DFS namespace it is hosting with either the DC specified by this method or the default DC the server is using. This MUST be treated as functionally equivalent to receiving a **NetrDfsSetInfo (Opnum 3)** method with the *Level* parameter value 101 and the **State** field of **DFS_INFO_101** set to VOLUME_STATE_RESYNCHRONIZE.<114>

3.2.6 Timer Events

No protocol timer events are required on the client beyond the timers required in the underlying RPC transport.

3.2.7 Other Local Events

No additional local events are used on the client beyond the events maintained in the underlying RPC transport.

3.3 Domain Controller Details

A DC hosting a DFS root target MUST conform to the specification in section 3.2.

In addition, the DFS server on a DC MUST do the following:

- Receive and respond to DFS root and DFS link referral requests for any domain-based DFS namespace in the domain. The DC need not be a DFS root target for the domain-based DFS namespace identified in the referral request.
- Receive and respond to the RPC methods, as specified in section 3.3.5.

3.3.1 Abstract Data Model

A DFS server on a DC MAY maintain the following data item:

- **ReferralCache:** A referral cache that is used when distributing referrals for domain-based DFS namespaces. This is for use by the DFS Referral Protocol, as specified in [MS-DFSC].

3.3.2 Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages, as specified in [MS-RPCE].

3.3.3 Initialization

No initialization is required beyond that used internally by RPC, as specified in [MS-RPCE].

3.3.4 Higher-Layer Triggered Events

A DFS server receives and acts upon DFS referral requests or DFS RPC method invocations. No other higher-layer triggered events are used.

3.3.5 Message Processing Events and Sequencing Rules

A DC MAY itself be a DFS root target. In such cases, it MUST process RPC methods, as specified in section 3.2. In addition, some methods SHOULD be specially supported, as specified in the following topics.

3.3.5.1 Extended Methods

3.3.5.1.1 NetrDfsEnumEx

A DFS server SHOULD support the *Level* parameter 200. The DFS server MUST validate the *DfsEntryPath* parameter against the name of the AD DS domain to which the DC is joined, and fail with ERROR_INVALID_NAME (0x0000007B) if it does not match. The DFS server then returns the list of domain-based DFS namespaces in the domain. This MUST be returned by performing an LDAP search for AD DS objects on the domain controller under the container. The container has the following DN.

```
CN=Dfs-Configuration,CN=System,<domain>
```

where <*domain*> is the DN of the AD DS domain. For details and more information, see section 2.3.1.<115>

3.3.5.2 Root Target Methods

3.3.5.2.1 NetrDfsRemoveFtRoot

A DFS server on a DC MUST support the following value for the *ApiFlags* parameter. This value is used to forcibly remove a domain-based DFS root target from a DFS namespace. This value is also used to delete a domain-based DFS namespace when the root target servers of the namespace are no longer available (for example, decommissioned).

Value	Meaning
DFS_FORCE_REMOVE 0x80000000	Forcibly removes the DFS root target.

The DFS server MUST remove the AD DS object of the domainv1-based DFS namespace (as specified in section 2.3.3) if the last DFS root target is removed. LDAP update operations MUST be issued to the PDCRoleHolder.<116>

3.3.5.2.2 NetrDfsFlushFtTable

A DFS server SHOULD purge the ReferralCache of all entries for the DFS namespace specified by the *wszFtDfsName* parameter of the NetrDfsFlushFtTable method.<117>

3.3.6 Timer Events

No protocol timer events are required on the client beyond the timers required in the underlying RPC transport.

3.3.7 Other Local Events

No additional local events are used on the client beyond the events maintained in the underlying RPC transport.

4 Protocol Examples

The following examples for domainv1 are in a Windows Server 2003 AD DS domain; the client used is Windows XP SP2. The member servers and domain controllers used in the example are all Windows Server 2003 SP1.

For domainv2, the client is Windows Vista. The member servers and domain controllers in this scenario are all Windows Server 2008.

4.1 Creating a New Domainv1-Based DFS Namespace

The following example describes the steps used to create a new domainv1-based DFS namespace.

1. A client determines whether a new domainv1-based DFS namespace is being created, or a new DFS root target is being added to an existing domain-based DFS namespace, by issuing an LDAP search for the AD DS object corresponding to the domainv1-based DFS namespace. The following illustration shows the LDAP search parameters that do this.
2. Because the domainv1-based DFS namespace does not already exist, the DC fails the LDAP search with LDAP_NO_SUCH_OBJECT.
3. The client creates an AD DS object for the new domainv1-based DFS namespace.
4. AD DS object creation is successful.
5. The client updates the ACL on the AD DS object to permit the new DFS root target CFS-41X-2C02 to update the AD DS object.
6. The ACL change on the AD DS object is successful.
7. The client issues a NetrDfsAddFtRoot method to the DFS root target server.
8. The DFS root target creates the DFS metadata required for the new domainv1-based DFS namespace and updates the DFS metadata in the AD object corresponding to the DFS namespace. This is shown as an LDAP modify operation to the PDC for the AD DS domain. The following illustration shows the LDAP modify parameters that do this.
9. The DFS metadata write is successful.
10. The DFS root target completes the **NetrDfsAddFtRoot** method to the client.

The following illustration shows the sequence of events.

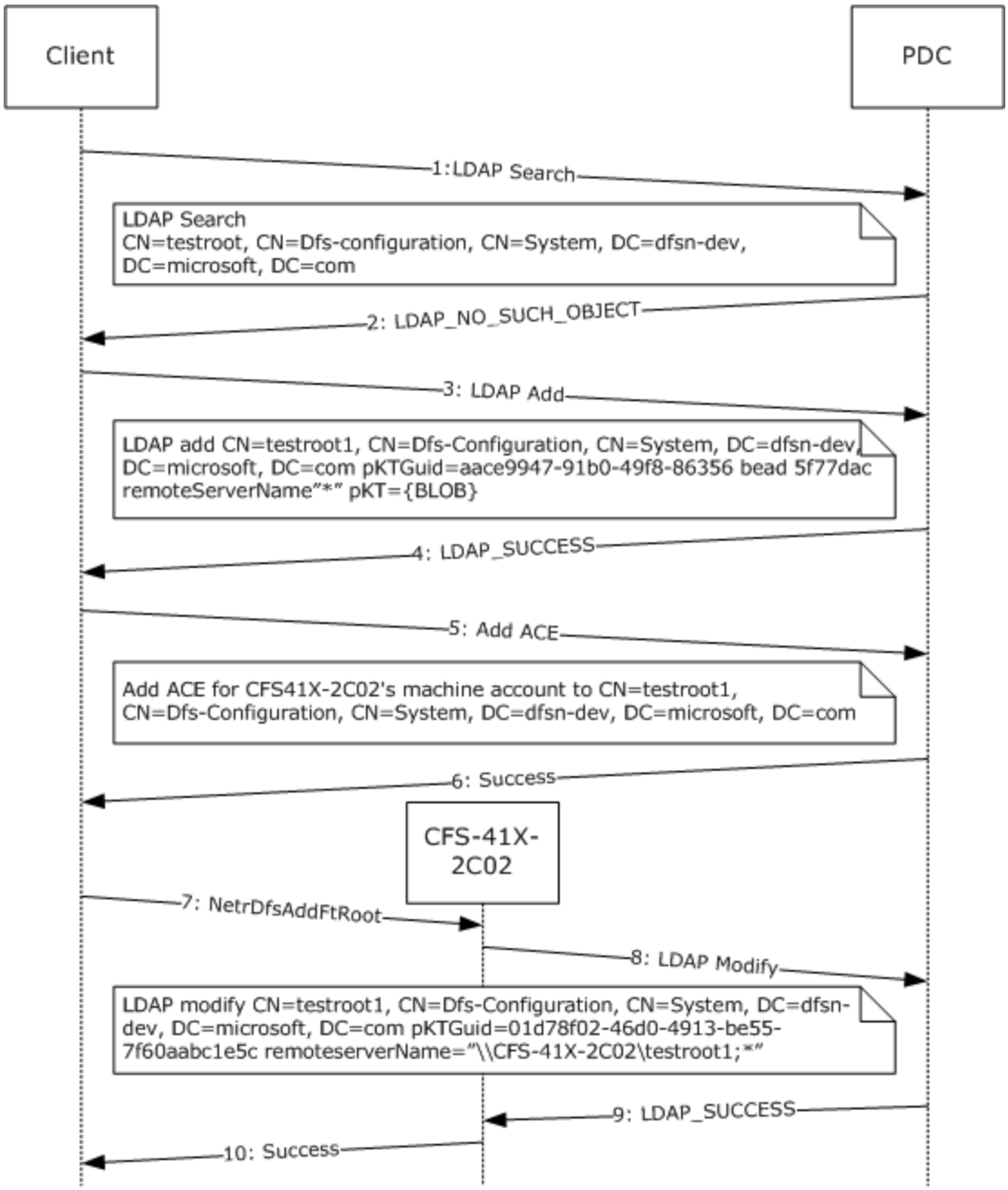


Figure 2: Creating a new domainv1-based DFS namespace

4.2 Adding a Root Target to an Existing Domainv1-Based DFS Namespace

The following example describes the steps used to create a new root target in an existing domainv1-based DFS namespace.

1. The client determines whether a new domainv1-based DFS namespace is being created, or a new DFS root target is being added to an existing domainv1-based DFS namespace, by issuing an

LDAP search for the AD DS object corresponding to the domainv1-based DFS namespace to the PDC.

2. Because the domainv1-based DFS namespace already exists, the LDAP search is successful.
3. The client updates the ACL on the AD DS object to permit the new DFS root target CFS-41X-2C03 to update the AD DS object.
4. The ACL change on the AD DS object is successful.
5. The client issues a NetrDfsAddFtRoot method to the new DFS root target server CFS-41X-2C03.
6. The DFS root target server CFS-41X-2C03 creates the DFS metadata required for the new domainv1-based DFS namespace and writes it to the PDC for the AD DS domain. The following illustration shows the LDAP modify parameters that do this.
7. The DFS metadata write is successful.
8. The DFS root target CFS-41X-2C03 completes the **NetrDfsAddFtRoot** method to the client.

The following illustration shows the sequence of events.

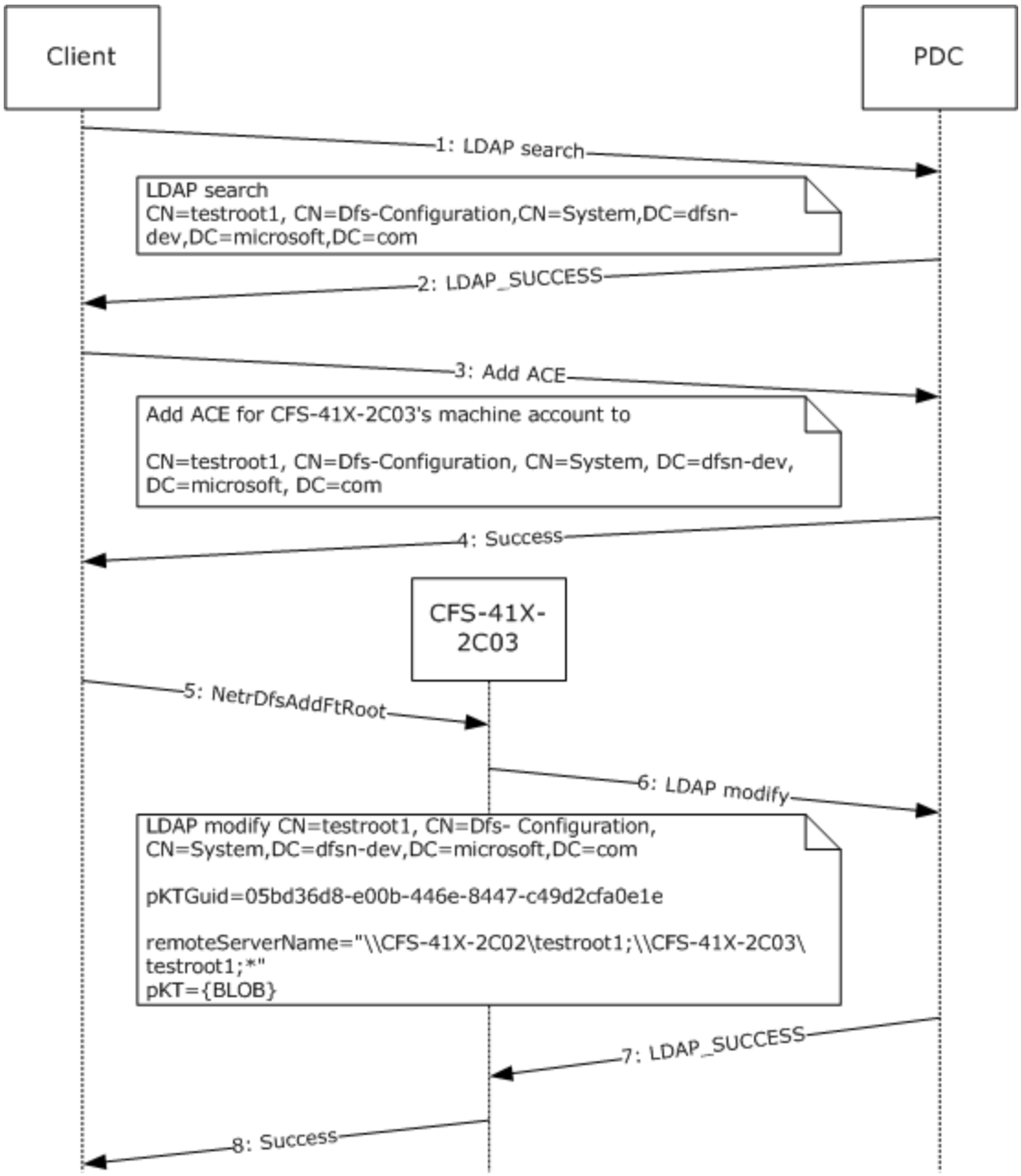


Figure 3: Adding a root target to an existing domainv1-based DFS namespace

4.3 Adding a New Link to a Domain-Based DFS Namespace

The following example describes the steps used to add a new DFS link to an existing domainv1-based DFS namespace with two root targets. The illustration in this example also shows how the DFS root target uses **NetrDfsSetInfo**, *Level* parameter 101, and VOLUME_STATE_RESYNCHRONIZE to update the DFS metadata of the domain-based DFS namespace with the new DFS link information, and then notifies the other root targets.

1. A client issues a NetrDfsAdd RPC method to the DFS root target CFS-41X-2C02 for the domainv1-based DFS namespace.

2. The DFS root target CFS-41X-2C02 issues an LDAP search operation to retrieve the **pKTGuid** attribute in the AD DS object for the domainv1-based DFS namespace to the PDC for the domain. The following illustration shows the DN of the AD DS object and the attribute searched.

3. The LDAP search is successful, and the value of the **PktGuid** attribute is returned.

4. The DFS root target CFS-41X-2C02 determines that the DFS metadata in its cache is up-to-date and determines whether the new link target is already in another DFS namespace. This is done by issuing the **NetrShareGetInfo** method, as specified in [MS-SRVS], specifying a *Level* parameter 1005 to the DFS link target CFS-44X-2B08 to check the link target share's properties. For more information on the **NetrShareGetInfo** method, see [MS-SRVS].

5. The **NetrShareGetInfo** RPC method returns an indication that the DFS link target share is not a DFS namespace. This information is used to determine the value of the PKT_ENTRY_TYPE_OUTSIDE_MY_DOM bit of the **Type** field of the DFSRootOrLinkIDBLOB (for more information, see section 2.3.3.1.1.2) for the DFS link. For this example, the bit is set to 0.

6. DFS link target CFS-41X-2C02 issues an LDAP modify operation to the PDC with a new **pKTGuid** value and the updated DFS metadata containing the new DFS link information.

7. The LDAP modify operation is successful.

8. The **NetrDfsAdd** method invoked by the client completes successfully.

9. The DFS root target, which updated the DFS metadata, issues the **NetrDfsSetInfo** method with the *Level* parameter 101 and the **State** field of **DFS_INFO_1** set to VOLUME_STATE_RESYNCHRONIZE to all of the other root targets. CFS-41X-2C02, in this example, is notifying CFS-41X-2C03.

10. On receiving the **NetrDfsSetInfo** method, *Level* parameter 101, and VOLUME_STATE_RESYNCHRONIZE, CFS-41X-2C03 issues an LDAP search to the PDC to verify whether the DFS metadata in its cache is up-to-date.

11. The LDAP search operation is successful and contains the **pKTGuid** attribute's value.

12. CFS-41X-2C03 determines that the cached DFS metadata it has needs to be refreshed. It then issues an LDAP search operation to retrieve the value of the **pKT** attribute, which contains the actual DFS metadata.

13. The LDAP search is successful and contains the DFS metadata in the reply.

14. In this example, CFS-41X-2C03 performs the required changes to its local state by adding the new DFS link. The **NetrDfsSetInfo** method that CFS-41X-2C02 issued is then completed.

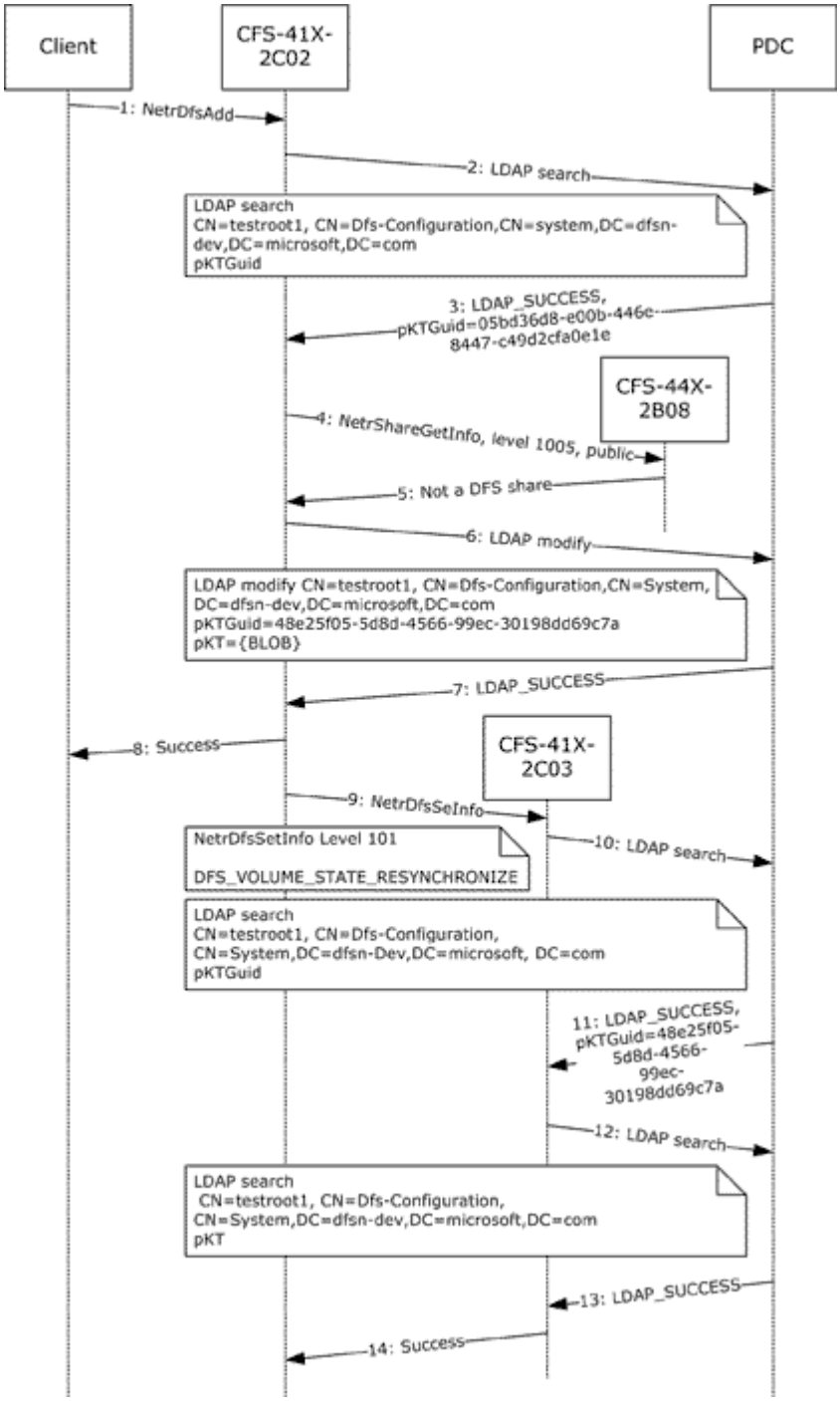


Figure 4: Adding a new link to a domainv1-based DFS namespace

4.4 Creating a New Domainv2-Based DFS Namespace

The following example describes the steps used to create a new domainv2-based DFS namespace.

1. A client determines whether a new domainv2-based DFS namespace is being created, or a new DFS root target is being added to an existing domainv2-based DFS namespace, by issuing an LDAP search for the AD DS object corresponding to the domainv2-based DFS namespace. The following illustration shows the LDAP search parameters that do this.

2. Because the domainv2-based DFS namespace does not already exist, the DC fails the LDAP search with LDAP_NO_SUCH_OBJECT.

3. The client creates an AD DS object for the new domainv2-based DFS namespace.

4. AD DS object creation is successful.

5. The client updates the ACL on the AD DS object to permit the new DFS root target CFS-41X-2C02 to update the AD DS object.

6. The ACL change on the AD DS object is successful.

7. The client issues a **NetrDfsAddRootTarget** method to the DFS root target server.

8. The DFS root target server creates a new DFS namespace LDAP entry with the DFS namespace anchor LDAP entry as its parent. The server also creates the DFS metadata required for the new domainv2-based DFS namespace and updates the DFS metadata in the AD object corresponding to the DFS namespace. This appears as an LDAP modify operation to the PDC for the AD DS domain. The following illustration shows the LDAP modify parameters that do this.

9. The DFS metadata write is successful.

10. The DFS root target completes the **NetrDfsAddRootTarget** method to the client.

The following illustration shows this sequence of events.

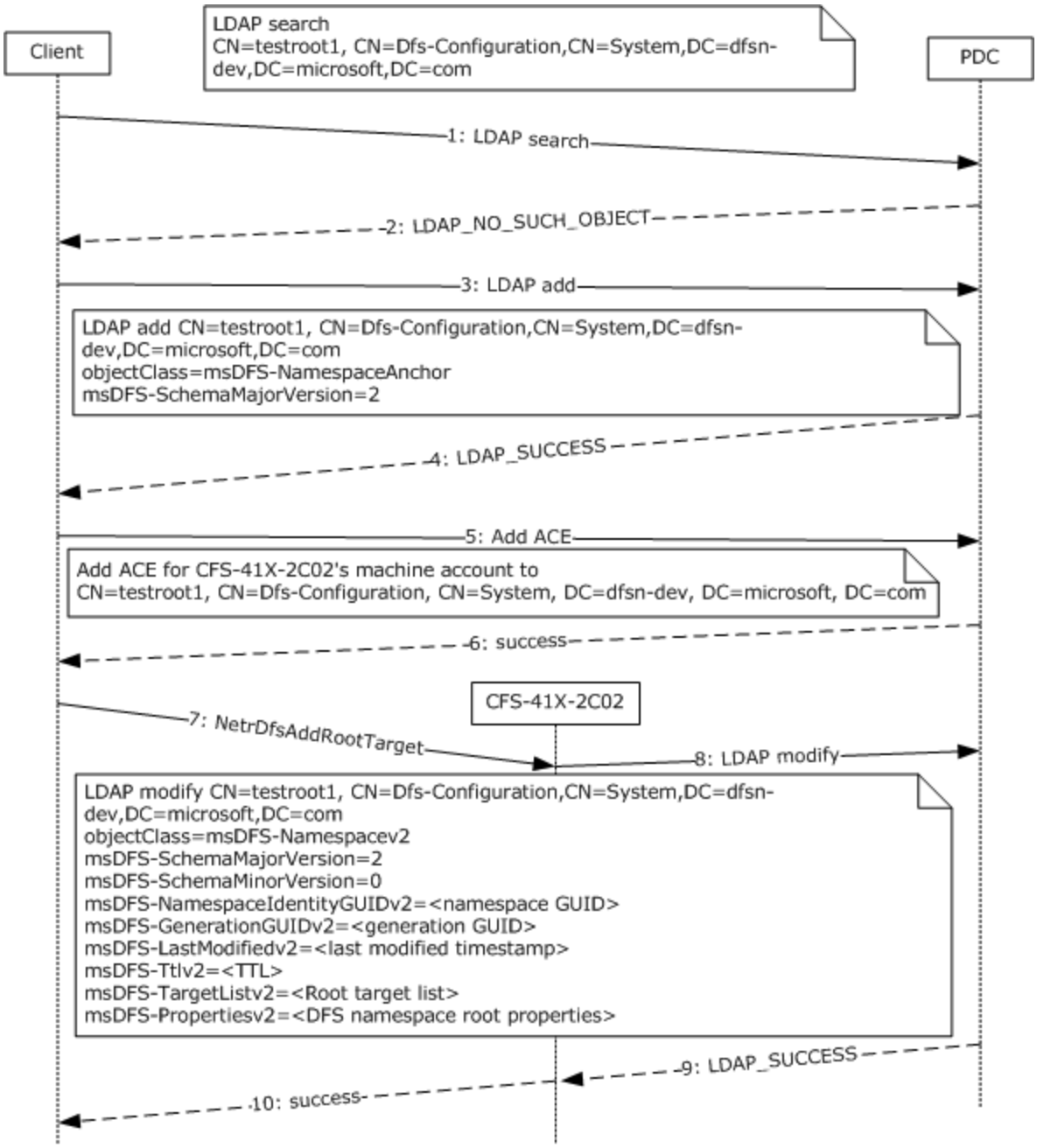


Figure 5: Creating a new domainv2-based DFS namespace

4.5 Adding a Root Target to an Existing Domainv2-Based DFS Namespace

The following example describes the steps used to create a new root target in an existing domainv2-based DFS namespace.

1. The client determines whether a new domainv2-based DFS namespace is being created, or a new DFS root target is being added to an existing domainv2-based DFS namespace, by issuing an

LDAP search for the AD DS object corresponding to the domainv2-based DFS namespace to the PDC.

2. Because the domainv2-based DFS namespace already exists, the LDAP search is successful.
3. The client updates the ACL on the AD DS object to permit the new DFS root target CFS-41X-2C03 to update the AD DS object.
4. The ACL change on the AD DS object is successful.
5. The client issues a **NetrDfsAddRootTarget** method to the new DFS root target server CFS-41X-2C03.
6. The DFS root target server CFS-41X-2C03 creates the DFS metadata required for the new domainv2-based DFS namespace and writes it to the PDC for the AD DS domain. The following illustration shows the LDAP modify parameters that do this.
7. The DFS metadata write is successful.
8. The DFS root target CFS-41X-2C03 completes the **NetrDfsAddRootTarget** method to the client.

The following illustration shows the sequence of events.

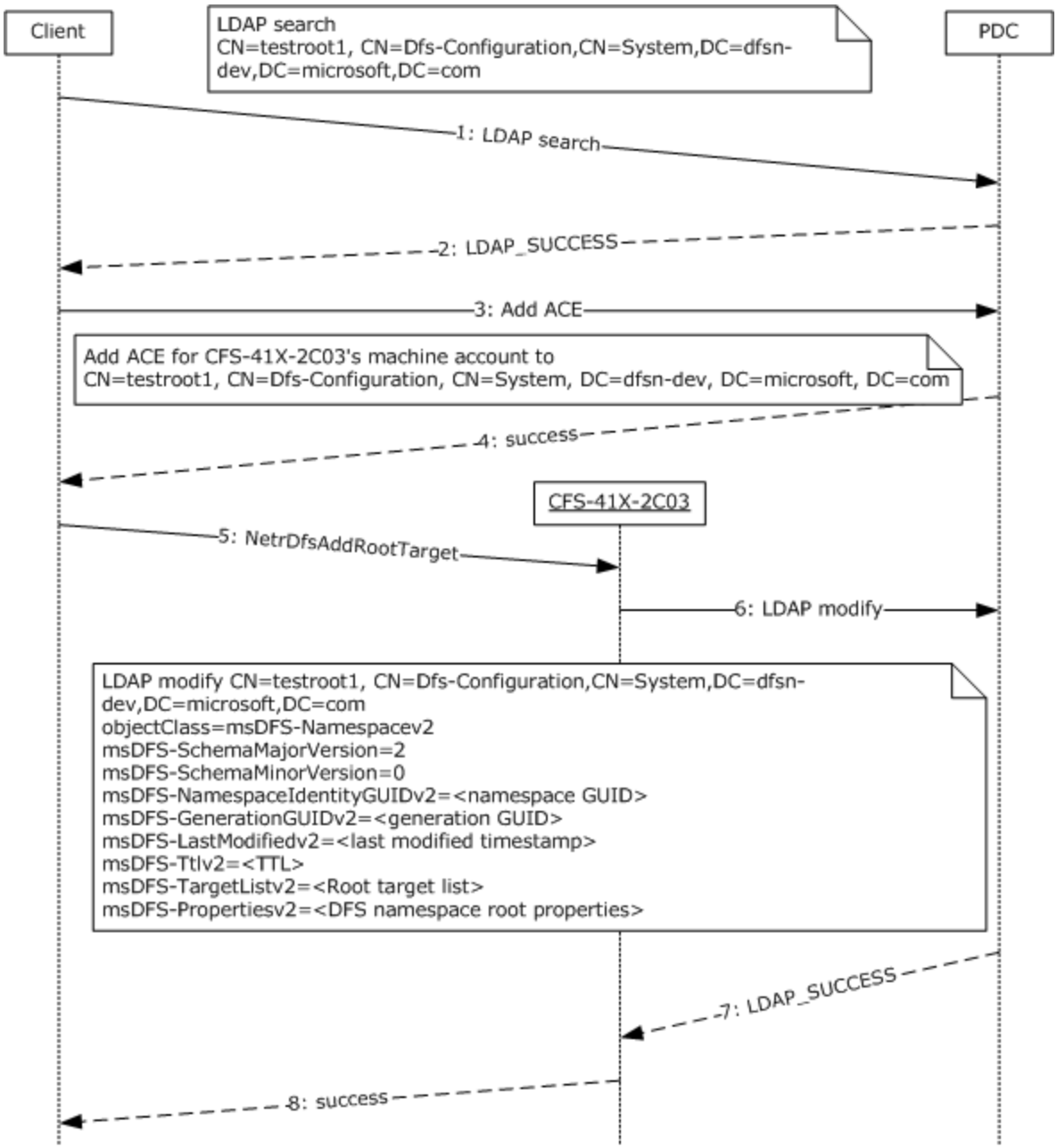


Figure 6: Adding a root target to an existing domainv2-based DFS namespace

4.6 Adding a New Link to a Domainv2-Based DFS Namespace

The following example describes the steps used to add a new DFS link to an existing domainv2-based DFS namespace with two root targets. The illustration in this example also shows how the DFS root target uses **NetrDfsSetInfo**, *Level* parameter 101, and VOLUME_STATE_ FORCE_SYNC, to update the DFS metadata of the domain-based DFS namespace with the new DFS link information, and then notifies the other root targets.

1. A client issues a **NetrDfsAdd** RPC method to the DFS root target CFS-41X-2C02 for the domainv2-based DFS namespace.

2. The DFS root target CFS-41X-2C02 performs either a full synchronization or an incremental synchronization. A full synchronization is performed if the DFS root target server is switching to a new PDC or if the DC that the synchronization operation is currently using is different from that in the <uSNChanged, DC invocation ID> tuple that was saved at the start of a previous full synchronization for the DFS namespace. An incremental synchronization is performed if the DFS root target server is already syncing with the PDC.

3. An incremental synchronization is done by issuing an LDAP search operation for the DFS namespace LDAP entry subtree to determine whether in any of the object classes of msDFS-Namespacev2, msDFS-Linkv2, or msDFS-DeletedLinkv2 the uSNChanged is greater than the saveduSNChanged value, where saveduSNChanged is the uSNChanged value from the tuple <uSNChanged, DC invocation ID> that was saved previously.

4. The DFS root target CFS-41X-2C02 determines that the DFS metadata in its cache is up-to-date and whether the new link target is already in another DFS namespace. This is done by issuing the **NetrShareGetInfo** method, as specified in [MS-SRVS], specifying a *Level* parameter 1005 to the DFS link target CFS-44X-2B08 to check the link target share's properties. For more information, see [MS-SRVS].

5. The **NetrShareGetInfo** RPC method returns an indication that the DFS link target share is not a DFS namespace. This information is used to determine the value of msDFS-Propertiesv2 for the DFS link.

6. DFS link target CFS-41X-2C02 issues an LDAP modify operation to the PDC with a new identity GUID (msDFS-LinkIdentityGUID) value and the updated DFS metadata that contains the new DFS link information. The link identity GUID is set at DFS link creation time and does not change for the lifetime of the LDAP entry. It is used to locate the in-memory data structure that corresponds to the DFS link in the DFS metadata cache.

7. The LDAP modify operation is successful.

8. The **NetrDfsAdd** method invoked by the client completes successfully.

9. To perform a full synchronization to ensure that this is propagated to all other root targets, the DFS root target, which updated the DFS metadata, issues the **NetrDfsSetInfo** method, with the *Level* parameter 101 and the **State** field of **DFS_INFO_101** set to VOLUME_STATE_FORCE_SYNC, to all of the other root targets. This is used to identify added or deleted DFS links. In this example, CFS-41X-2C02 is notifying CFS-41X-2C03.

10. On receiving the **NetrDfsSetInfo** method with *Level* parameter 101 and VOLUME_STATE_FORCE_SYNC, CFS-41X-2C03 issues an LDAP search to the PDC to verify whether the DFS metadata in its cache is up-to-date. A different uSNChanged value from the <uSNChanged, DC invocation ID> tuple saved at the start of a previous full sync would indicate what has changed, and it would subsequently perform an incremental sync to propagate any DFS metadata change.

11. CFS-41X-2C03 determines that the cached DFS metadata it has needs to be refreshed. It then issues an LDAP search operation to retrieve the attributes associated with the msDFS-Linkv2 class, which contains the actual DFS metadata.

12. The LDAP search is successful and contains the DFS metadata in the reply.

13. In this example, CFS-41X-2C03 performs the required changes to its local state by adding the new DFS link. The **NetrDfsSetInfo** method that CFS-41X-2C02 issued is then completed.

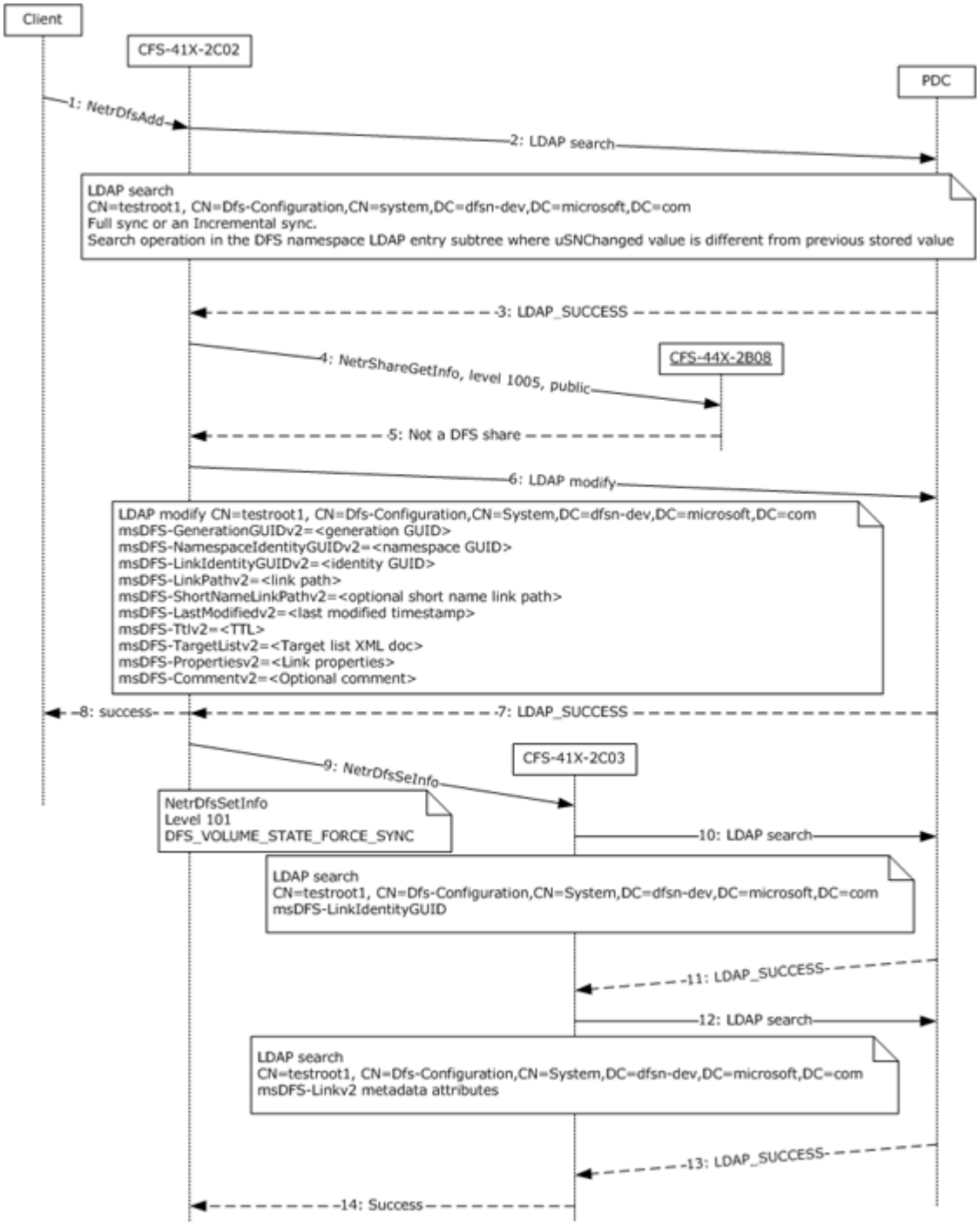


Figure 7: Adding a new link to a domainv2-based DFS namespace

4.7 Enumerating DFS Links in a Domain-Based DFS Namespace

The following example describes the sequence of an interactive administration application that enumerates all domain-based DFS namespaces in a domain and all DFS links of a domain-based DFS namespace.

1. To enumerate all of the domain-based DFS namespaces in the dfsn-dev AD DS domain, the administration application issues the **NetrDfsEnumEx** RPC method with the *DfsEntryPath* parameter \\dfsn-dev and *Level* parameter 200. This method is issued to the DC.

2. The DC returns two domain-based DFS namespaces in the AD DS domain: \\dfsn-dev\testroot1 and \\dfsn-dev\testroot2.

3. The user decides to view information about the domain-based DFS namespace \\dfsn-dev\testroot1. Before the administering application can issue the RPC method to obtain information about that DFS namespace, it must determine which DFS root target it will issue the RPC method to. This is done by issuing a DFS root referral request to the DC, as specified in [MSDFS].

4. The DC responds to the DFS root referral request with the two DFS root targets: \\cfs-41x-2c02\testroot1 and \\cfs-41x-2c03\testroot1.

5. The **NetrDfsEnumEx** RPC method to obtain information about the DFS namespace \\dfsn-dev\testroot1 is then issued to the root target cfs-41x-2c02.

6. The root target returns the DFS root and one DFS link in the domain-based DFS namespace.

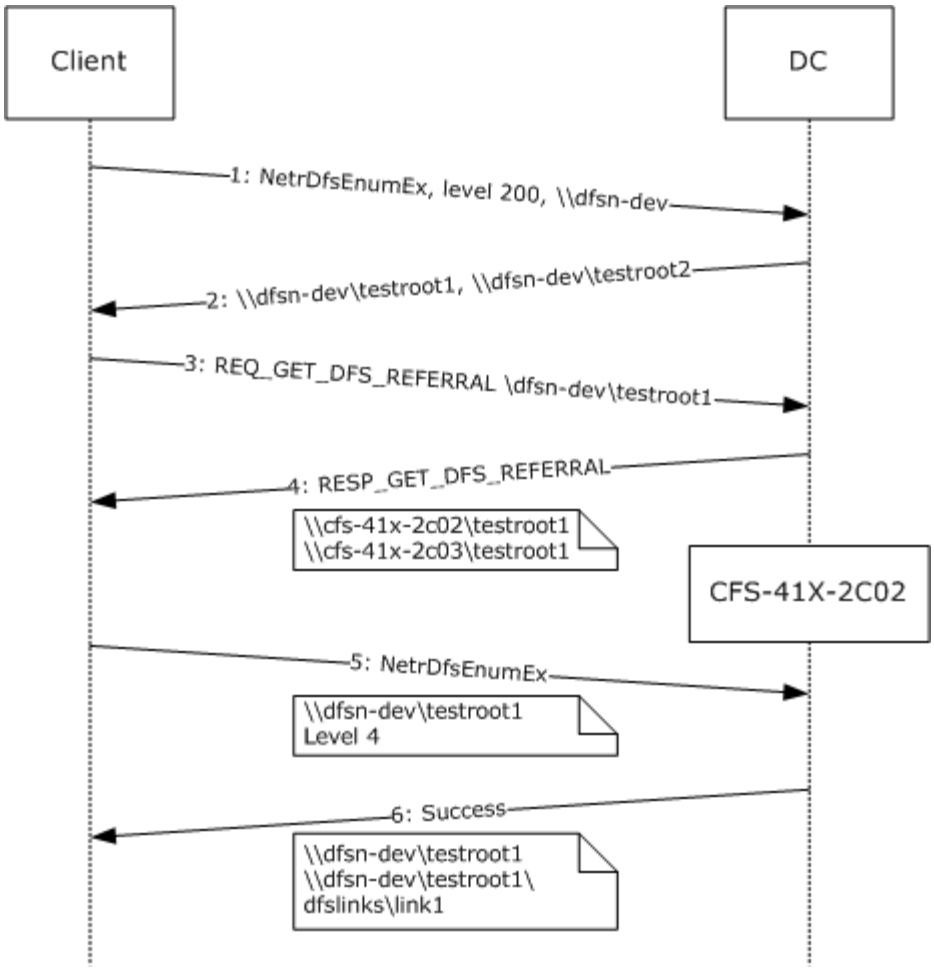


Figure 8: Enumerating DFS links in a domainv1-based DFS namespace

4.8 DFS Metadata of a Domainv1-Based DFS Namespace

This example uses the following domainv1-based DFS namespaces:

- DFS root: \\dfsn-dev\testroot1
- DFS root targets: \\cfs-41x-2c02\testroot1 and \\cfs-41x-2c03\testroot1
- DFS link: \\dfsn-dev\testroot1\dfslinks\link1
- DFS link target \\cfs-44x-2b08\public

The following illustration shows the hexadecimal dump of the DFS metadata for this domainv1-based DFS namespace. The offsets in the hexadecimal dump are used to explain the dump.

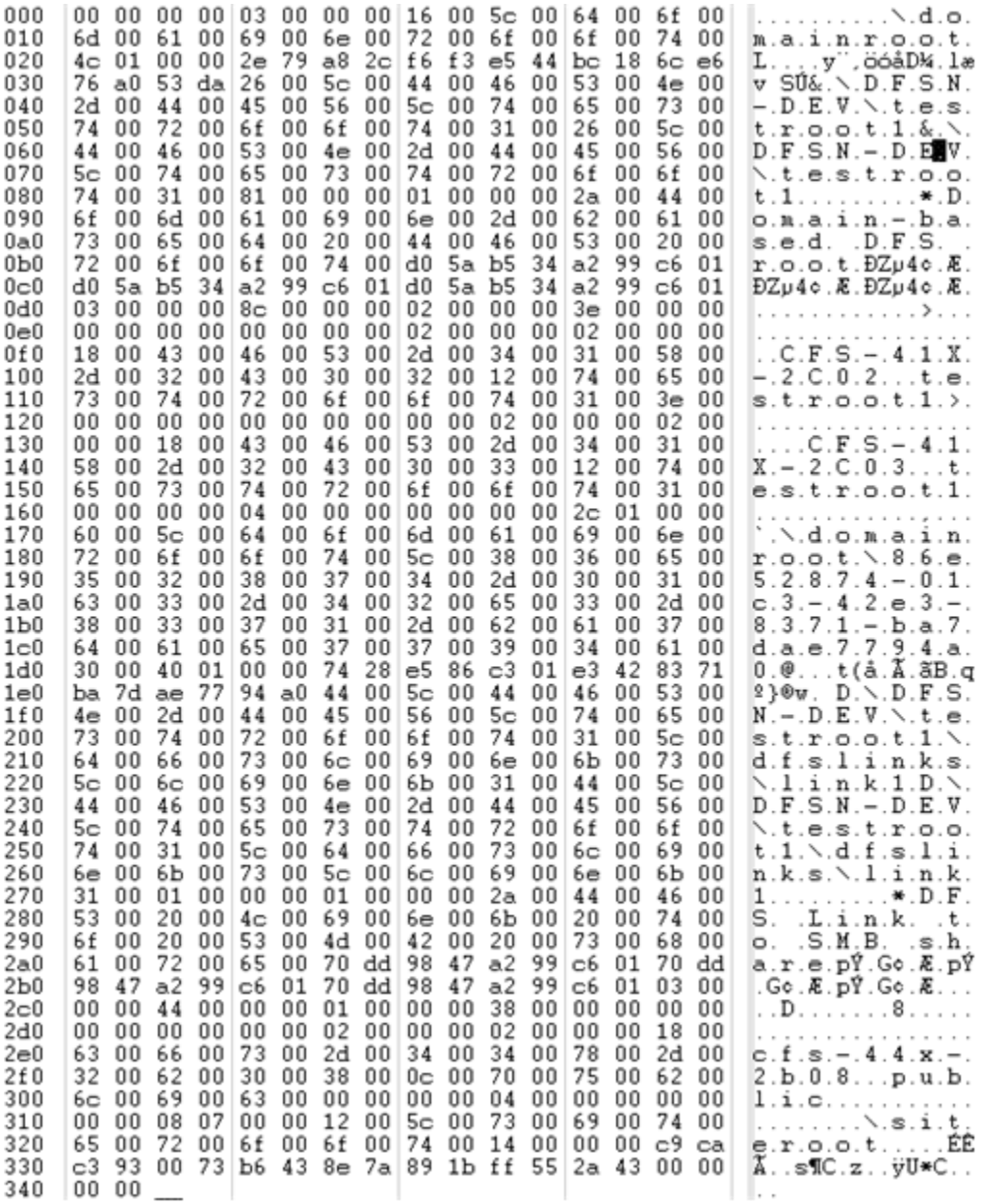


Figure 9: Hexadecimal dump of the DFS metadata for a domainv1-based DFS namespace

The following table lists elements of the hexadecimal dump.

Offset	Raw hex values	DFS metadata field
0x000	00 00 00 00	BLOBVersion: 0
0x004	03 00 00 00	BLOBElementCount: 3
0x008	-	DFSNamespaceElementBLOB #1

Offset	Raw hex values	DFS metadata field
0x008	16 00	BLOBNameSize: 0x0016 (22)
0x00A	5C 00 64 00 6F 00 6D 00 61 00 69 00 6E 00 72 00 6F 00 6F 00 74 00	BLOBName: \domainroot
0x020	4C 01 00 00	BLOBDataSize: 0x0000014c (332)
0x024	-	DFSNamespaceRootBLOB
0x024	-	DFSRootOrLinkIDBLOB
0x024	2E 79 A8 2C F6 F3 E5 44 BC 18 6C E6 76 A0 53 DA	RootOrLinkGuid: 2ca8792e-f3f6-44e5-bc18-6ce676a053da
0x034	26 00	PrefixSize: 0x0026 (38)
0x036	5C 00 44 00 46 00 53 00 4E 00 2D 00 44 00 45 00 56 00 5C 00 74 00 65 00 73 00 74 00 72 00 6F 00 6F 00 74 00 31 00	Prefix: \DFSN-DEV\testroot1
0x05C	26 00	ShortPrefixSize: 0x0026 (38)
0x05E	5C 00 44 00 46 00 53 00 4E 00 2D 00 44 00 45 00 56 00 5C 00 74 00 65 00 73 00 74 00 72 00 6F 00 6F 00 74 00 31 00	Short prefix: \DFSN-DEV\testroot1
0x084	81 00 00 00	Type: 0x00000081
0x088	01 00 00 00	State: 0x00000001
0x08C	2A 00	CommentSize: 0x002A (42)
0x08E	44 00 6F 00 6D 00 61 00 69 00 6E 00 2D 00 62 00 61 00 73 00 65 00 64 00 20 00 44 00 46 00 53 00 20 00 72 00 6F 00 6F 00 74 00	Comment: Domain-based DFS root
0x0B8	D0 5A B5 34 A2 99 C6 01	PrefixTimeStamp: June 26, 2006 21:28:57
0x0C0	D0 5A B5 34 A2 99 C6 01	StateTimeStamp: June 26, 2006 21:28:57
0x0C8	D0 5A B5 34 A2 99 C6 01	CommentTimeStamp: June 26, 2006 21:28:57

Offset	Raw hex values	DFS metadata field
0x0D0	03 00 00 00	Version: 0x0000003
0x0D4	8C 00 00 00	DFSTargetListBLOBSize: 0x0000008C (140)
0x0D8	-	DFSTargetListBLOB
0x0D8	02 00 00 00	TargetCount: 0x00000002
0x0DC	-	TargetEntryBLOB
0x0DC	3E 00 00 00	TargetEntrySize: 0x0000003E (62)
0x0E0	00 00 00 00 00 00 00 00	TargetTimeStamp: 0
0x0E8	02 00 00 00	TargetState: 0x00000002
0x0EC	02 00 00 00	TargetType: 0x00000002
0x0F0	18 00	ServerNameSize: 0x0018 (24)
0x0F2	43 00 46 00 53 00 2D 00 34 00 31 00 58 00 2D 00 32 00 43 00 30 00 32 00	ServerName: CFS-41X-2C02
0x10A	12 00	ShareNameSize: 0x0012 (18)
0x10C	74 00 65 00 73 00 74 00 72 00 6F 00 6F 00 74 00 31 00	ShareName: testroot1
0x11E	-	TargetEntryBLOB
0x11E	3E 00 00 00	TargetEntrySize: 0x0000003E (62)
0x122	00 00 00 00 00 00 00 00	TargetTimeStamp: 0
0x12A	02 00 00 00	TargetState: 0x00000002
0x12E	02 00 00 00	TargetType: 0x00000002
0x132	18 00	ServerNameSize: 0x0018 (24)
0x134	43 00 46 00 53 00 2D 00 34 00 31 00 58 00 2D 00 32 00 43 00 30 00 33 00	ServerName: CFS-41X-2C03
0x14C	12 00	ShareNameSize: 0x0012 (18)
0x14E	74 00 65 00 73 00 74 00 72 00 6F 00 6F 00 74 00 31 00	ShareName: testroot1
0x160	00 00 00 00	Padding
0x164	04 00 00 00	ReservedBLOBSize: 0x00000004

Offset	Raw hex values	DFS metadata field
0x168	00 00 00 00	ReservedBLOB: 0x00000000
0x16C	2C 01 00 00	TTL: 0x0000012C (300 Seconds)
0x170	60 00	BLOBNameSize: 0x0060 (96)
0x172	5C 00 64 00 6F 00 6D 00 61 00 69 00 6E 00 72 00 6F 00 6F 00 74 00 5C 00 38 00 36 00 65 00 35 00 32 00 38 00 37 00 34 00 2D 00 30 00 31 00 63 00 33 00 2D 00 34 00 32 00 65 00 33 00 2D 00 38 00 33 00 37 00 31 00 2D 00 62 00 61 00 37 00 64 00 61 00 65 00 37 00 37 00 39 00 34 00 61 00 30 00	BLOBName: \domainroot\86e52874-01c3-42e3-8371-ba7dae7794a0
0x1D2	40 01 00 00	BLOBDataSize: 0x00000140 (320)
0x1D6	-	DFSNamespaceLinkBLOB
0x1D6	-	DFSRootOrLinkIDBLOB
0x1D6	74 28 E5 86 C3 01 E3 42 83 71 BA 7D AE 77 94 A0	RootOrLinkGuid: 86e52874-1c3-42e3-8371-ba7dae7794a0
0x1E6	44 00	PrefixSize: 0x0044 (68)
0x1E8	5C 00 44 00 46 00 53 00 4E 00 2D 00 44 00 45 00 56 00 5C 00 74 00 65 00 73 00 74 00 72 00 6F 00 6F 00 74 00 31 00 5C 00 64 00 66 00 73 00 6C 00 69 00 6E 00 6B 00 73 00 5C 00 6C 00 69 00 6E 00 6B 00 31 00	Prefix: \DFSN-DEV\testroot1\dfslinks\link1
0x22C	44 00	ShortPrefixSize: 0x0044 (68)
0x22E	5C 00 44 00 46 00 53 00 4E 00 2D 00 44 00 45 00 56 00 5C 00 74 00 65 00 73 00 74 00 72 00 6F 00 6F 00 74 00 31 00 5C 00 64 00 66 00 73 00 6C 00 69 00 6E 00 6B 00 73 00	Short prefix: \DFSN-DEV\testroot1\dfslinks\link1

Offset	Raw hex values	DFS metadata field
	5C 00 6C 00 69 00 6E 00 6B 00 31 00	
0x272	01 00 00 00	Type: 0x00000001
0x276	01 00 00 00	State: 0x00000001
0x27A	2A 00	CommentSize: 0x002A (42)
0x27C	44 00 46 00 53 00 20 00 4C 00 69 00 6E 00 6B 00 20 00 74 00 6F 00 20 00 53 00 4D 00 42 00 20 00 73 00 68 00 61 00 72 00 65 00	Comment: DFS link to SMB share
0x2A6	70 DD 98 47 A2 99 C6 01	PrefixTimeStamp: June 26, 2006 21:29:29
0x2AE	70 DD 98 47 A2 99 C6 01	StateTimeStamp: June 26, 2006 21:29:29
0x2B6	70 DD 98 47 A2 99 C6 01	CommentTimeStamp: June 26, 2006 21:29:29
0x2BE	03 00 00 00	Version: 3
0x2C2	44 00 00 00	DFSTargetListBLOBSize: 0x00000044 (68)
0x2C6	-	DFSTargetListBLOB
0x2C6	01 00 00 00	TargetCount: 0x00000001
0x2CA	-	TargetEntryBLOB
0x2CA	38 00 00 00	TargetEntrySize: 0x00000038 (56)
0x2CE	00 00 00 00 00 00 00 00	TargetTimeStamp: 0
0x2D6	02 00 00 00	TargetState: 0x00000002
0x2DA	02 00 00 00	TargetType: 0x00000002
0x2DE	18 00	ServerNameSize: 0x0018 (24)
0x2E0	63 00 66 00 73 00 2D 00 34 00 34 00 78 00 2D 00 32 00 62 00 30 00 38 00	ServerName: cfs-44x-2b08
0x2F8	0C 00	ShareNameSize: 0x000C (12)
0x2FA	70 00 75 00 62 00 6C 00 69 00 63 00	ShareName: public
0x306	00 00 00 00	Padding

Offset	Raw hex values	DFS metadata field
0x30A	04 00 00 00	ReservedBLOBSize: 0x00000004
0x30E	00 00 00 00	ReservedBLOB
0x312	08 07 00 00	TTL: 0x00000708 (1800 Seconds)
0x316	12 00	BLOBNameSize: 0x0012 (18)
0x318	5C 00 73 00 69 00 74 00 65 00 72 00 6F 00 6F 00 74 00	BLOBName: \siteroot
0x32A	14 00 00 00	BLOBDataSize: 0x00000014 (20)
0x32E	-	SiteInformationBLOB
0x32E	C9 CA C3 93 00 73 B6 43 8E 7A 89 1B FF 55 2A 43	SiteTableGuid: 93c3cac9-7300-43b6-8e7a-891bff552a43
0x33E	00 00 00 00	SiteEntryCount: 0x00000000

5 Security

The following sections specify security considerations for implementers of the DFS: Namespace Management Protocol.

5.1 Security Considerations for Implementers

The DFS: Namespace Management Protocol allows any user to establish a connection to the RPC server. The protocol uses the underlying RPC Protocol to retrieve the identity of the caller that made the method call, as specified in [MS-RPCE], section 3.3.3.5.3.. Clients SHOULD create an authenticated RPC connection. Servers SHOULD use this identity to perform method-specific access checks.<118>

5.2 Index of Security Parameters

Security Parameter	Section
Authentication Protocol	2.1

6 Appendix A: Full IDL

The DFS: Namespace Management Protocol contains one interface, whose IDL definition is listed in this section.

```
import "ms-dtyp.idl";

/* ----- structures and methods described [MS-DFSNM], section 2 and 3 ----- */

[
    uuid(4fc742e0-4a10-11cf-8273-00aa004ae673),
    version(3.0),
    ms_union,
    pointer default(unique)
]

interface netdfs {

typedef DWORD    NET_API_STATUS;
typedef WCHAR *  NETDFS SERVER OR DOMAIN HANDLE;

typedef enum _DFS_TARGET_PRIORITY_CLASS {
    DfsInvalidPriorityClass        = -1,
    DfsSiteCostNormalPriorityClass = 0,
    DfsGlobalHighPriorityClass     = 1,
    DfsSiteCostHighPriorityClass   = 2,
    DfsSiteCostLowPriorityClass    = 3,
    DfsGlobalLowPriorityClass      = 4
} DFS_TARGET_PRIORITY_CLASS;

typedef struct  DFS TARGET PRIORITY {
    DFS_TARGET_PRIORITY_CLASS TargetPriorityClass;
    unsigned short            TargetPriorityRank;
    unsigned short            Reserved;
} DFS_TARGET_PRIORITY;

typedef struct  DFS STORAGE INFO {
             unsigned long State;
    [string] WCHAR       * ServerName;
    [string] WCHAR       * ShareName;
} DFS_STORAGE_INFO;

typedef struct  DFS STORAGE INFO 1 {
             unsigned long State;
    [string] WCHAR       * ServerName;
    [string] WCHAR       * ShareName;
    DFS_TARGET_PRIORITY TargetPriority;
} DFS STORAGE INFO 1;

typedef struct  DFSM ROOT LIST ENTRY {
    [string, unique] WCHAR * ServerShare;
} DFSM_ROOT_LIST_ENTRY;

typedef struct  DFSM ROOT LIST {
    DWORD   cEntries;
    [size is(cEntries)] DFSM ROOT LIST ENTRY Entry[];
} DFSM_ROOT_LIST;

typedef enum {
    DFS NAMESPACE VERSION ORIGIN COMBINED = 0,
    DFS NAMESPACE VERSION ORIGIN SERVER,
    DFS_NAMESPACE_VERSION_ORIGIN_DOMAIN
```

```
} DFS_NAMESPACE_VERSION_ORIGIN;

typedef struct _DFS_SUPPORTED_NAMESPACE_VERSION_INFO {
    unsigned long     DomainDfsMajorVersion;
    unsigned long     DomainDfsMinorVersion;
    unsigned long long  DomainDfsCapabilities;
    unsigned long     StandaloneDfsMajorVersion;
    unsigned long     StandaloneDfsMinorVersion;
    unsigned long long  StandaloneDfsCapabilities;
} DFS_SUPPORTED_NAMESPACE_VERSION_INFO,
 *PDFS SUPPORTED NAMESPACE VERSION INFO;

typedef struct _DFS_INFO_1 {
    [string] WCHAR *  EntryPath;
} DFS_INFO_1;

typedef struct  DFS INFO 2 {
    [string] WCHAR *  EntryPath;
    [string] WCHAR *  Comment;
             DWORD    State;
             DWORD    NumberOfStorages;
} DFS_INFO_2;

typedef struct _DFS_INFO_3 {
    [string] WCHAR *  EntryPath;
    [string] WCHAR *  Comment;
             DWORD    State;
             DWORD    NumberOfStorages;
    [size is(NumberOfStorages)] DFS STORAGE INFO * Storage;
} DFS_INFO_3;

typedef struct _DFS_INFO_4 {
    [string] WCHAR       * EntryPath;
    [string] WCHAR       * Comment;
             DWORD         State;
             unsigned long Timeout;
             GUID          Guid;
             DWORD         NumberOfStorages;
    [size_is(NumberOfStorages)] DFS_STORAGE_INFO * Storage;
} DFS INFO 4;

typedef struct  DFS INFO 5 {
    [string] WCHAR       * EntryPath;
    [string] WCHAR       * Comment;
             DWORD         State;
             unsigned long Timeout;
             GUID          Guid;
             unsigned long PropertyFlags;
             unsigned long MetadataSize;
             DWORD         NumberOfStorages;
} DFS_INFO_5;

typedef struct _DFS_INFO_6 {
    [string] WCHAR       * EntryPath;
    [string] WCHAR       * Comment;
             DWORD         State;
             unsigned long Timeout;
             GUID          Guid;
             unsigned long PropertyFlags;
             unsigned long MetadataSize;
             DWORD         NumberOfStorages;
    [size_is(NumberOfStorages)] DFS_STORAGE_INFO_1 * Storage;
} DFS INFO 6;

typedef struct _DFS_INFO_7 {
```

```
            GUID          GenerationGuid;
} DFS_INFO_7;

typedef struct _DFS_INFO_8 {
    [string] WCHAR       * EntryPath;
    [string] WCHAR       * Comment;
             DWORD         State;
             unsigned long Timeout;
             GUID          Guid;
             unsigned long PropertyFlags;
             unsigned long MetadataSize;
             PSECURITY_DESCRIPTOR pSecurityDescriptor;
             DWORD         NumberOfStorages;
} DFS_INFO_8,
 *LPDFS_INFO_8;

typedef struct  DFS INFO 9 {
    [string] WCHAR       * EntryPath;
    [string] WCHAR       * Comment;
             DWORD         State;
             unsigned long Timeout;
             GUID                  Guid;
             unsigned long         PropertyFlags;
             unsigned long         MetadataSize;
             PSECURITY_DESCRIPTOR pSecurityDescriptor;
             DWORD                 NumberOfStorages;
    [size_is(NumberOfStorages)]
             DFS_STORAGE_INFO_1 * Storage;
} DFS INFO 9,
 *LPDFS_INFO_9;

typedef struct _DFS_INFO_50 {
             unsigned long    NamespaceMajorVersion;
             unsigned long    NamespaceMinorVersion;
             unsigned __int64 NamespaceCapabilities;
} DFS INFO 50;

typedef struct _DFS_INFO_100 {
    [string]  WCHAR       * Comment;
} DFS INFO 100;

typedef struct  DFS INFO 101 {
    unsigned long         State;

} DFS_INFO_101;

typedef struct _DFS_INFO_102 {
    unsigned long Timeout;
} DFS_INFO_102;

typedef struct _DFS_INFO_103 {
    unsigned long PropertyFlagMask;
    unsigned long PropertyFlags;

} DFS_INFO_103;

typedef struct  DFS INFO 104 {
    DFS_TARGET_PRIORITY TargetPriority;
} DFS INFO 104;

typedef struct _DFS_INFO_105 {
    [string] WCHAR       * Comment;
             DWORD         State;
             unsigned long Timeout;
             unsigned long PropertyFlagMask;
```

```
            unsigned long PropertyFlags;
} DFS_INFO_105;

typedef struct _DFS_INFO_106 {
    DWORD               State;
    DFS TARGET PRIORITY TargetPriority;
} DFS_INFO_106;

typedef struct _DFS_INFO_107 {
   [string] WCHAR       * Comment;
            DWORD         State;
            unsigned long Timeout;
            unsigned long PropertyFlagMask;
            unsigned long PropertyFlags;
            PSECURITY_DESCRIPTOR     pSecurityDescriptor;
} DFS_INFO_107;

typedef struct _DFS_INFO_150 {
            unsigned long        SdLengthReserved;
            PSECURITY_DESCRIPTOR pSecurityDescriptor;
} DFS_INFO_150;

typedef struct  DFS INFO 200 {
    [string] WCHAR       * FtDfsName;
} DFS_INFO_200;

typedef struct _DFS_INFO_300 {
             DWORD   Flags;
    [string] WCHAR * DfsName;
} DFS_INFO_300;

typedef [switch_type(unsigned long)] union _DFS_INFO_STRUCT {
    [case(1)]
        DFS INFO 1   * DfsInfo1;
    [case(2)]
        DFS INFO 2   * DfsInfo2;
    [case(3)]
        DFS_INFO_3   * DfsInfo3;
    [case(4)]
        DFS INFO 4   * DfsInfo4;
    [case(5)]
        DFS INFO 5   * DfsInfo5;
    [case(6)]
        DFS_INFO_6   * DfsInfo6;
    [case(7)]
        DFS INFO 7   * DfsInfo7;
    [case(8)]
        DFS INFO 8   * DfsInfo8;
    [case(9)]
        DFS_INFO_9   * DfsInfo9;
    [case(50)]
        DFS INFO 50  * DfsInfo50;
    [case(100)]
        DFS INFO 100 * DfsInfo100;
    [case(101)]
        DFS_INFO_101 * DfsInfo101;
    [case(102)]
        DFS_INFO_102 * DfsInfo102;
    [case(103)]
        DFS_INFO_103 * DfsInfo103;
    [case(104)]
        DFS_INFO_104 * DfsInfo104;
    [case(105)]
        DFS INFO 105 * DfsInfo105;
    [case(106)]
```

```
        DFS_INFO_106 * DfsInfo106;
    [case(107)]
        DFS_INFO_107 * DfsInfo107;
    [case(150)]
        DFS_INFO_150 * DfsInfo150;
    [default]
        ;
} DFS_INFO_STRUCT;


typedef struct  DFS INFO 1 CONTAINER {
    DWORD   EntriesRead;
    [size_is(EntriesRead)] DFS_INFO_1 * Buffer;
} DFS_INFO_1_CONTAINER;

typedef struct _DFS_INFO_2_CONTAINER {
    DWORD   EntriesRead;
    [size_is(EntriesRead)] DFS_INFO_2 * Buffer;
} DFS INFO 2 CONTAINER;

typedef struct _DFS_INFO_3_CONTAINER {
    DWORD   EntriesRead;
    [size is(EntriesRead)] DFS INFO 3 * Buffer;
} DFS_INFO_3_CONTAINER;

typedef struct _DFS_INFO_4_CONTAINER {
    DWORD   EntriesRead;
    [size_is(EntriesRead)] DFS_INFO_4 * Buffer;
} DFS INFO 4 CONTAINER;

typedef struct  DFS INFO 5 CONTAINER {
    DWORD   EntriesRead;
    [size_is(EntriesRead)] DFS_INFO_5 * Buffer;
} DFS INFO 5 CONTAINER;

typedef struct  DFS INFO 6 CONTAINER {
    DWORD   EntriesRead;
    [size_is(EntriesRead)] DFS_INFO_6 * Buffer;
} DFS_INFO_6_CONTAINER;

typedef struct _DFS_INFO_8_CONTAINER {
    DWORD   EntriesRead;
    [size_is(EntriesRead)] LPDFS_INFO_8 Buffer;
} DFS_INFO_8_CONTAINER,
 *LPDFS_INFO_8_CONTAINER;

typedef struct _DFS_INFO_9_CONTAINER {
    DWORD   EntriesRead;
    [size_is(EntriesRead)] LPDFS_INFO_9 Buffer;
} DFS_INFO_9_CONTAINER,
 *LPDFS_INFO_9_CONTAINER;

typedef struct _DFS_INFO_200_CONTAINER {
    DWORD   EntriesRead;
    [size_is(EntriesRead)] DFS_INFO_200 * Buffer;
} DFS_INFO_200_CONTAINER;

typedef struct _DFS_INFO_300_CONTAINER {
    DWORD   EntriesRead;
    [size_is(EntriesRead)] DFS_INFO_300 * Buffer;
} DFS_INFO_300_CONTAINER;

typedef struct  DFS INFO ENUM STRUCT {
    DWORD   Level;
    [switch_is(Level)] union {
```

```
        [case(1)]
            DFS_INFO_1_CONTAINER   * DfsInfo1Container;
        [case(2)]
            DFS_INFO_2_CONTAINER   * DfsInfo2Container;
        [case(3)]
            DFS_INFO_3_CONTAINER   * DfsInfo3Container;
        [case(4)]
            DFS_INFO_4_CONTAINER   * DfsInfo4Container;
        [case(5)]
            DFS_INFO_5_CONTAINER   * DfsInfo5Container;
        [case(6)]
            DFS_INFO_6_CONTAINER   * DfsInfo6Container;
        [case(8)]
            DFS_INFO_8_CONTAINER   * DfsInfo8Container;
        [case(9)]
            DFS_INFO_9_CONTAINER   * DfsInfo9Container;
        [case(200)]
            DFS_INFO_200_CONTAINER * DfsInfo200Container;
        [case(300)]
            DFS_INFO_300_CONTAINER * DfsInfo300Container;
    } DfsInfoContainer;
} DFS_INFO_ENUM_STRUCT;


DWORD NetrDfsManagerGetVersion();

NET_API_STATUS NetrDfsAdd(
    [in,string]             WCHAR           * DfsEntryPath,
    [in,string]             WCHAR           * ServerName,
    [in,unique,string]      WCHAR           * ShareName,
    [in,unique,string]      WCHAR           * Comment,
    [in]                    DWORD            Flags);

NET API STATUS NetrDfsRemove(
    [in,string]             WCHAR           * DfsEntryPath,
    [in,unique,string]      WCHAR           * ServerName,
    [in,unique,string]      WCHAR           * ShareName);

NET_API_STATUS NetrDfsSetInfo(
    [in,string]             WCHAR           * DfsEntryPath,
    [in,unique,string]      WCHAR           * ServerName,
    [in,unique,string]      WCHAR           * ShareName,
    [in]                    DWORD             Level,
    [in,switch_is(Level)]   DFS_INFO_STRUCT * DfsInfo);

NET API STATUS NetrDfsGetInfo(
    [in,string]             WCHAR           * DfsEntryPath,
    [in,unique,string]      WCHAR           * ServerName,
    [in,unique,string]      WCHAR           * ShareName,
    [in]                    DWORD             Level,
    [out,switch_is(Level)]  DFS_INFO_STRUCT * DfsInfo);

NET_API_STATUS NetrDfsEnum(
    [in]                    DWORD                  Level,
    [in]                    DWORD                  PrefMaxLen,
    [in,out,unique]         DFS_INFO_ENUM_STRUCT * DfsEnum,
    [in,out,unique]         DWORD                * ResumeHandle);

NET API STATUS NetrDfsMove(
    [in,string]             WCHAR           * DfsEntryPath,
    [in,string]             WCHAR           * NewDfsEntryPath,
    [in]                    unsigned long     Flags);

void Opnum7NotUsedOnWire();
```

```
void Opnum8NotUsedOnWire();

void Opnum9NotUsedOnWire();

NET_API_STATUS NetrDfsAddFtRoot(
    [in,string]             WCHAR           * ServerName,
    [in,string]             WCHAR           * DcName,
    [in,string]             WCHAR           * RootShare,
    [in,string]             WCHAR           * FtDfsName,
    [in,string]             WCHAR           * Comment,
    [in,string]             WCHAR           * ConfigDN,
    [in]                    BOOLEAN           NewFtDfs,
    [in]                    DWORD             ApiFlags,
    [in,out,unique]         DFSM_ROOT_LIST ** ppRootList);

NET_API_STATUS NetrDfsRemoveFtRoot(
    [in,string]             WCHAR           * ServerName,
    [in,string]             WCHAR           * DcName,
    [in,string]             WCHAR           * RootShare,
    [in,string]             WCHAR           * FtDfsName,
    [in]                    DWORD             ApiFlags,
    [in,out,unique]         DFSM_ROOT_LIST ** ppRootList);

NET_API_STATUS NetrDfsAddStdRoot(
    [in,string]             WCHAR           * ServerName,
    [in,string]             WCHAR           * RootShare,
    [in,string]             WCHAR           * Comment,
    [in]                    DWORD             ApiFlags);

NET_API_STATUS NetrDfsRemoveStdRoot(
    [in,string]             WCHAR           * ServerName,
    [in,string]             WCHAR           * RootShare,
    [in]                    DWORD             ApiFlags);

NET_API_STATUS NetrDfsManagerInitialize(
    [in,string]             WCHAR           * ServerName,
    [in]                    DWORD             Flags);

NET_API_STATUS NetrDfsAddStdRootForced(
    [in,string]             WCHAR           * ServerName,
    [in,string]             WCHAR           * RootShare,
    [in,string]             WCHAR           * Comment,
    [in,string]             WCHAR           * Share);

NET_API_STATUS NetrDfsGetDcAddress(
    [in,string]             WCHAR           * ServerName,
    [in,out,string]         WCHAR          ** DcName,
    [in,out]                BOOLEAN         * IsRoot,
    [in,out]                unsigned long   * Timeout);

NET_API_STATUS NetrDfsSetDcAddress(
    [in,string]             WCHAR           * ServerName,
    [in,string]             WCHAR           * DcName,
    [in]                    DWORD             Timeout,
    [in]                    DWORD             Flags);

NET API STATUS NetrDfsFlushFtTable(
    [in,string]             WCHAR           * DcName,
    [in,string]             WCHAR           * wszFtDfsName);

NET_API_STATUS NetrDfsAdd2(
    [in,string]             WCHAR           * DfsEntryPath,
    [in,string]             WCHAR           * DcName,
    [in,string]             WCHAR           * ServerName,
    [in,unique,string]      WCHAR           * ShareName,
```

```
    [in,unique,string]      WCHAR           * Comment,
    [in]                    DWORD             Flags,
    [in,out,unique]         DFSM_ROOT_LIST ** ppRootList);

NET_API_STATUS NetrDfsRemove2(
    [in,string]             WCHAR           * DfsEntryPath,
    [in,string]             WCHAR           * DcName,
    [in,unique,string]      WCHAR           * ServerName,
    [in,unique,string]      WCHAR           * ShareName,
    [in,out,unique]         DFSM_ROOT_LIST ** ppRootList);

NET_API_STATUS NetrDfsEnumEx(
    [in,string]             WCHAR               * DfsEntryPath,
    [in]                    DWORD                 Level,
    [in]                    DWORD                 PrefMaxLen,
    [in,out,unique]         DFS_INFO_ENUM_STRUCT * DfsEnum,
    [in,out,unique]         DWORD               * ResumeHandle);

NET API STATUS NetrDfsSetInfo2(
    [in,string]             WCHAR           * DfsEntryPath,
    [in,string]             WCHAR           * DcName,
    [in,unique,string]      WCHAR           * ServerName,
    [in,unique,string]      WCHAR           * ShareName,
    [in]                    DWORD             Level,
    [in,switch_is(Level)]   DFS_INFO_STRUCT * pDfsInfo,
    [in,out,unique]         DFSM_ROOT_LIST ** ppRootList);

NET_API_STATUS NetrDfsAddRootTarget(
    [in,unique,string]  LPWSTR  pDfsPath,
    [in,unique,string]  LPWSTR  pTargetPath,
    [in]                ULONG   MajorVersion,
    [in,unique,string]  LPWSTR  pComment,
    [in]                BOOLEAN NewNamespace,
    [in]                ULONG   Flags);

NET API STATUS NetrDfsRemoveRootTarget(
    [in,unique,string]  LPWSTR  pDfsPath,
    [in,unique,string]  LPWSTR  pTargetPath,
    [in]                ULONG   Flags);

NET_API_STATUS NetrDfsGetSupportedNamespaceVersion(
    [in]                DFS NAMESPACE VERSION ORIGIN   Origin,
    [in,unique,string] NETDFS_SERVER_OR_DOMAIN_HANDLE pName,
    [out]               PDFS_SUPPORTED_NAMESPACE_VERSION_INFO pVersionInfo);
}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Vista
- Windows Server 2008
- Windows Server 2003 R2
- Windows Server 2003 SP2
- Windows Server 2003 SP1
- Windows Server 2003
- Windows XP
- Windows 2000 Server
- Windows NT Server 4.0
- Windows NT 4.0

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 1.3: Windows NT 4.0 supports only stand-alone DFS namespaces.

<2> Section 1.4: All Windows clients support the SMB access protocol.

Windows Vista and Windows Server 2008 support SMB2.

Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 support the Microsoft WebDAV client.

Windows 2000, Windows 2000 Server, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 support the Microsoft NFS client.

<3> Section 1.5: Windows relies on manual coordination between human operators to ensure that only one DFS metadata modification is in progress at any time.

<4> Section 1.7: The Windows RPC Protocol returns RPC_S_PROCNUM_OUT_OF_RANGE to notify the client that an RPC method is out of range, as specified in [MS-RPCE].

<5> Section 2.2.3.5: Only Windows Server 2008 supports ABDE mode.

<6> Section 2.2.3.6: Windows stores its DFS metadata as a single entry in the AD for domainv1-based DFS. The DFS metadata, stored as a BLOB, is limited to 10 MB. The total number of roots and links is limited by that size. However, it is impossible to determine exactly what these limits would be because each metadata subentry size depends on comment string length, DFS link path length, DFS target path length, and so on.

A domainv2-based DFS has a single LDAP entry under the DFS configuration container that continues to represent each domainv2-based DFS namespace. Under the DFS namespace LDAP

entry, there is one LDAP entry per DFS link, and each LDAP entry contains the metadata corresponding to that entity, such as targets, property flags, and other information.

For stand-alone DFS, Windows servers store the DFS metadata in the registry and, hence, a maximum of 50,000 links is recommended. It is valid for a DFS service implementation to have unrestricted numbers of roots and links in a stand-alone DFS namespace and to talk to any existing Windows DFS client.

If a non-Windows–based DFS is not in a mixed Windows and non-Windows environment (that is, all of the root targets are non-Windows), it can also implement an unrestricted number of roots and links, interact with existing DFS clients, and support Windows link targets. It is only in a mixed-mode environment that the non-Windows DFS service MUST store the namespace metadata in one piece in the AD DS, which imposes the restriction on the numbers of roots and links, as previously discussed.

<7> Section 2.2.3.7: Windows servers return a null GUID for stand-alone DFS namespaces.

<8> Section 2.2.3.9: Windows stores its DFS metadata as a single entry in the AD for domainv1-based DFS. The DFS metadata, stored as a BLOB, is limited to 10 MB. The total number of roots and links is limited by that size. However, it is impossible to determine exactly what these limits would be because each metadata subentry size depends on comment string length, DFS link path length, DFS target path length, and so on.

A domainv2-based DFS has a single LDAP entry under the DFS configuration container that continues to represent each domainV2-based DFS namespace. Under the DFS namespace LDAP entry, there is one LDAP entry per DFS link, and each LDAP entry contains the metadata corresponding to that entity, such as targets, property flags, and other information.

For stand-alone DFS, Windows servers store the DFS metadata in the registry and, hence, a maximum of 50,000 links is recommended. It is valid for a DFS service implementation to have unrestricted numbers of roots and links in a stand-alone DFS namespace and to talk to any existing Windows DFS client.

If a non-Windows–based DFS is not in a mixed Windows and non-Windows environment (that is, all of the root targets are non-Windows), it can also implement an unrestricted number of roots and links, interact with existing DFS clients, and support Windows link targets. It is only a mixed-mode environment that the non-Windows DFS service MUST store the namespace metadata in one piece in the AD DS, which imposes the restriction on the numbers of roots and links as discussed earlier.

<9> Section 2.2.3.10: This level is supported only in Windows Vista and Windows Server 2008.

To identify the DFS metadata format in use, two mandatory attributes are defined in the schema to hold the major version (ldapDisplayName = msDFS-SchemaMajorVersion) and minor version (ldapDisplayName = msDFS-SchemaMinorVersion) numbers. The **rangeUpper** attribute in the attribute schema for these version number attributes determines the format of the DFS metadata supported in the AD DS forest. The value of these attributes determines the DFS metadata format in use for an existing DFS namespace. The implementation of a domainv2-based DFS namespace has rangeUpper=2, rangeLower=2 for the NamespaceMajorVersion and rangeUpper=0, rangeLower=0 for the NamespaceMinorVersion.

A change to an existing, or the addition of a new, mandatory attribute increments the major version and sets the minor version to 0. A change to an existing, or the addition of a new, optional attribute increments the minor version without changing the major version.

The NamespaceMajorVersion and NamespaceMinorVersion determine the format of the DFS metadata supported in the AD DS forest.

	NamespaceMajorVersion	NamespaceMinorVersion
Domainv1-basedDFS in Windows Vista or Windows Server 2008	1	1
Domainv2-based DFS in Windows Vista or Windows Server 2008	2	0
Stand-alone DFS in Windows Vista or Windows Server 2008	1	2

<10> Section 2.2.3.10: This level is supported only in Windows Vista and Windows Server 2008.

To identify the DFS metadata format in use, two mandatory attributes are defined in the schema to hold the major version (ldapDisplayName = msDFS-SchemaMajorVersion) and minor version (ldapDisplayName = msDFS-SchemaMinorVersion) numbers. The **rangeUpper** attribute in the attribute schema for these version number attributes determines the format of the DFS metadata supported in the AD DS forest. The value of these attributes determines the DFS metadata format in use for an existing DFS namespace. The implementation of a domainv2-based DFS namespace has rangeUpper=2, rangeLower=2 for the NamespaceMajorVersion and rangeUpper=0, rangeLower=0 for the NamespaceMinorVersion.

A change to an existing, or the addition of a new, mandatory attribute increments the major version and sets the minor version to 0. A change to an existing, or the addition of a new, optional attribute increments the minor version without changing the major version.

The NamespaceMajorVersion and NamespaceMinorVersion determine the format of the DFS metadata supported in the AD DS forest.

	NamespaceMajorVersion	NamespaceMinorVersion
Domainv1-based DFS in Windows Vista or Windows Server 2008	1	1
Domainv2-based DFS in Windows Vista or Windows Server 2008	2	0
Stand-alone DFS in Windows Vista or Windows Server 2008	1	2

<11> Section 2.2.3.10: Only Windows Server 2008 supports ABDE mode.

<12> Section 2.2.4.3: Only Windows Server 2008 supports ABDE mode.

<13> Section 2.3.2: While a domainv1-based DFS namespace can be created in Windows Server 2003, Windows XP, and Windows 2000, creating a domainv2-based DFS namespace requires a Windows Vista AD DS domain functional level and Windows Server 2008 to host it.

Since the **relative distinguished name (RDN)** of a domainv2-based DFS namespace entry is based on the DFS namespace just as for a domainv1-based DFS namespace, there is no issue of a name collision with a domainv1-based DFS namespace and a domainv2-based DFS namespace having the same name. AD DS will fail an attempt to create such a scenario.

<14> Section 2.3.3.1.1: Windows 2000 Server performs a case-sensitive comparison of the name. Windows Server 2003 and Windows Server 2008 perform a case-insensitive comparison of the name.

<15> Section 2.3.3.1.1.1: Windows Server 2008 and Windows Server 2003 truncate this **ReservedBLOB** down to 4 bytes when updating an existing DFSNamespaceRootBLOB or DFSNamespaceLinkBLOB.

<16> Section 2.3.3.1.1.2: Windows Server 2008 and Windows Server 2003 use the same name for the **ShortPrefix** field and the **Prefix** field. Windows 2000 Server stores an 8.3 name in the **ShortPrefix** field.

<17> Section 2.3.3.1.1.2: Only Windows Server 2008 and Windows Server 2003 support DFS referral site costing.

<18> Section 2.3.3.1.1.2: Only Windows Server 2008 and Windows Server 2003 support DFS root scalability mode.

<19> Section 2.3.3.1.1.2: Only Windows Server 2008 and Windows Server 2003 SP1 support DFS client target failback.

<20> Section 2.3.3.1.1.3.1: Windows Server 2008, Windows Server 2003 R2, Windows Server 2003 SP2, and Windows Server 2003 SP1 use the **TargetTimeStamp** field for TargetPriorityRank and PriorityClass.

<21> Section 2.3.3.1.1.3.1: Windows Server 2008, Windows Server 2003 R2, Windows Server 2003 SP2, and Windows Server 2003 SP1 use the **TargetTimeStamp** field for Target PriorityRank and PriorityClass.

<22> Section 2.3.3.1.1.3.1: Windows Server 2008 and Windows Server 2003 always set this field to 0x00000002. Windows 2000 Server sets this field to 0x00000001 for DFS root targets and to 0x00000002 for DFS link targets.

<23> Section 2.3.3.1.1.4: Only Windows 2000 Server uses the SiteInformationBLOB. Windows Server 2008 and Windows Server 2003 preserve this BLOB if it already exists. When creating a new DFS namespace, Windows Server 2008 and Windows Server 2003 create this BLOB with a **SiteEntryCount** of 0 and do not create any SiteEntryBLOBs.

<24> Section 2.3.3.1.1.4: Windows 2000 Server determines the AD DS site of a DFS root target or DFS link target when it is added to a domain-based DFS namespace and stores it in this BLOB. The **NetrDfsManagerReportSiteInfo** method , as specified in [MS-SRVS], is issued to the DFS root target server or the DFS link target server that were added to determine the AD DS site information.

<25> Section 2.3.4.2: This is reserved for future use.

<26> Section 2.3.4.3: This is reserved for future use.

<27> Section 2.3.4.3: This is reserved for future use and MUST not be currently used.

<28> Section 2.3.4.4: This is reserved for future use and MUST not be currently used.

<29> Section 3.1.3: Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, and Windows 2000 clients create a separate binding for every method invocation.

<30> Section 3.1.5.1.1: Windows 2000 Server does not support a domain-based DFS namespace in the NetrDfsAdd method. To work around this behavior, Windows Server 2008, Windows Vista,

Windows Server 2003, and Windows XP clients invoke the NetrDfsAdd2 method, as specified when the **NetrDfsAdd** method fails with ERROR_NOT_SUPPORTED (0x00000032).

<31> Section 3.1.5.1.2: Windows 2000 Server does not support a domain-based DFS namespace in the NetrDfsRemove method. To work around this behavior, Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP clients invoke the NetrDfsRemove2 method, as previously specified when the NetrDfsRemove method fails with ERROR_NOT_SUPPORTED (0x00000032).

<32> Section 3.1.5.1.3: Windows 2000 Server does not support a domain-based DFS namespace in the NetrDfsSetInfo method. To work around this behavior, Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP clients invoke the NetrDfsSetInfo2 method, as previously specified when the NetrDfsSetInfo method fails with ERROR_NOT_SUPPORTED (0x00000032).

<33> Section 3.1.5.1.3: Windows 2000 Server does not support DFS_VOLUME_STATE_RESYNCHRONIZE for the **State** field of **DFS_INFO_101** for a *Level* parameter value of 101.

<34> Section 3.1.5.1.4: For Level values other than 200, Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP clients first call the **NetrDfsManagerGetVersion** method. If the returned version value is 0x00000004 or greater, the client calls the NetrDfsEnumEx method. If the returned version value is less than 0x00000004, the client calls the NetrDfsEnum method. For level 200, Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP clients always call the NetrDfsEnumEx method on the PDC.

<35> Section 3.1.5.1.4: The Windows 2000 Server client does not call either NetrDfsEnum or NetrDfsEnumEx for level 200; rather, it determines the list of domain-based DFS namespaces through an LDAP query directly on the DFS configuration container.

<36> Section 3.1.5.1.4: Windows clients rely on human operators to detect inconsistent results in displayed output and to request a new enumeration.

<37> Section 3.1.5.2.1: Windows 2000 Server does not support a domain-based DFS namespace in the **NetrDfsAdd** method. To work around this behavior, Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP clients invoke the **NetrDfsAdd2** method when the **NetrDfsAdd** method fails with ERROR_NOT_SUPPORTED (0x00000032).

<38> Section 3.1.5.2.2: Windows 2000 Server does not support a domain-based DFS namespace in the NetrDfsRemove method. To work around this behavior, Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP clients invoke the NetrDfsRemove2 method when the NetrDfsRemove method fails with ERROR_NOT_SUPPORTED (0x00000032).

<39> Section 3.1.5.2.3: Windows 2000 Server does not support a domain-based DFS namespace in the NetrDfsSetInfo method. To work around this behavior, Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP clients invoke the NetrDfsSetInfo2 method when the NetrDfsSetInfo method fails with ERROR_NOT_SUPPORTED (0x00000032).

<40> Section 3.1.5.2.3: Windows 2000 Server does not support DFS_VOLUME_STATE_RESYNCHRONIZE for the **State** field of **DFS_INFO_101** for the *Level* parameter value of 101.

<41> Section 3.1.5.3.1: Only Windows 2000 Server returns other existing DFS root targets of the DFS namespace in the *ppRootList* parameter. Windows Server 2008 and Windows Server 2003 do not return any information in the *ppRootList* parameter.

<42> Section 3.1.5.3.1: Windows clients fail the NetrDfsAddFtRoot operation on the client if an IP address is given as the *ServerName* parameter. This is because Windows clients attempt to use the *ServerName* parameter as the security principal when updating the ACL of the AD DS object of a

domain-based DFS namespace. Because AD does not permit an IP address to be used as a security principal, a Windows client will fail on the ACL update before sending the NetrDfsAddFtRoot request message.

<43> Section 3.1.5.3.2: Only Windows 2000 Server returns other existing DFS root targets of the DFS namespace in the *ppRootList* parameter. Windows Server 2008 and Windows Server 2003 do not return any information in the *ppRootList* parameter.

<44> Section 3.2.1: Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, Windows 2000, and Windows 2000 Server cache the DFS metadata as an optimization.

<45> Section 3.2.4: Windows 2000 Server does not support DFS_VOLUME_STATE_RESYNCHRONIZE for the **State** field of **DFS_INFO_101** for the *Level* parameter 101. It returns a list of domain-based DFS root targets for the client to notify of the change in DFS metadata in the NetrDfsAdd2, NetrDfsRemove2, NetrDfsSetInfo2, NetrDfsAddFtRoot, and **NetrDfsRemoveFtRoot** methods.

<46> Section 3.2.5: Windows uses only the error code values, as specified in [MS-ERREF].

<47> Section 3.2.5: This method is supported only in Windows Server 2008.

<48> Section 3.2.5: This method is supported only in Windows Server 2008.

<49> Section 3.2.5: This method is supported only in Windows Server 2008.

<50> Section 3.2.5: For historical reasons, Windows 2000 Server uses method opnums 7, 8, and 9 for local RPC calls to itself. These methods are never called over the network by Windows clients or servers (to another server).

Opnums reserved for local use apply to Windows as follows.

Opnum	Description
6-9	Only used locally, never remotely, by Windows 2000 Server and not by any other Windows version.

<51> Section 3.2.5.1.1: This method is not supported in Windows Server 2008 and Windows Server 2003; they will return ERROR_NOT_SUPPORTED (0x00000032) if this method is called.

<52> Section 3.2.5.1.1: This method is not supported in Windows NT 4.0.

<53> Section 3.2.5.1.2: For computers running on Windows NT 4.0, **NetrDfsManagerGetVersion** returns 0x00000001; on Windows 2000 Server, **NetrDfsManagerGetVersion** returns 0x00000002; on Windows Server 2003 and Windows Server 2008, **NetrDfsManagerGetVersion** returns 0x00000004.

<54> Section 3.2.5.1.2: The RPC interface version has remained constant since Windows NT 4.0. Windows 2000 Server adds new methods to the same RPC interface. Also, Windows 2000 Server supports hosting of at most one DFS namespace. Hence, applications could potentially use the version information returned to handle version differences.

<55> Section 3.2.5.1.2: A Windows NT Server 4.0 can host at most one DFS namespace. All versions of Windows Server 2003, except Windows Server 2003, Standard Edition, support hosting of more than one DFS namespace per server. In default configurations, Windows Server 2003, Standard Edition supports hosting of at most one DFS namespace per server.

<56> Section 3.2.5.1.2: Windows clients use the returned value to decide when to call NetrDfsEnum versus NetrDfsEnumEx. For more information, see section 3.1.5.1.4.

<57> Section 3.2.5.1.2: Windows Server 2003 SP1, Windows Server 2003 SP2, Windows Server 2003 R2, and Windows Server 2008 support **NetrDfsMove (Opnum 6)**.

<58> Section 3.2.5.1.3: Windows 2000 Server requires that the *DFS_ADD_VOLUME* flags parameter be specified when creating a new link, and Windows Server 2003 and Windows Server 2008 do not.

<59> Section 3.2.5.1.3: The **msDFS-Commentv2** field in the Windows Server 2008 metadata is updated with the value that is passed on as input.

<60> Section 3.2.5.1.3: **NetrDfsAdd** in Windows 2000 Server supports only a stand-alone DFS namespace. Windows 2000 Server returns ERROR_NOT_SUPPORTED (0x00000032) if **NetrDfsAdd** is called on a domain-based DFS namespace.

<61> Section 3.2.5.1.3: In Windows Server 2003 and Windows Server 2008, **NetrDfsAdd** supports both stand-alone and domain-based DFS namespaces.

<62> Section 3.2.5.1.4: This method does not support domain-based DFS namespaces in Windows 2000 Server; NetrDfsRemove2 is used instead. Windows 2000 Server will return ERROR_NOT_SUPPORTED (0x00000032) if **NetrDfsRemove** is called on a domain-based DFS namespace.

<63> Section 3.2.5.1.4: In Windows Server 2003, the **NetrDfsRemove** method is functionally equivalent to NetrDfsRemove2.

<64> Section 3.2.5.1.5: Windows 2000 and Windows Server 2008 allow the target state of a root target or a link target to be set to either DFS_STORAGE_STATE_ONLINE or DFS_STORAGE_STATE_OFFLINE. Windows Server 2003 does not allow the target state of a root target to be set to DFS_STORAGE_STATE_OFFLINE.

<65> Section 3.2.5.1.5: Windows Server 2008 allows setting the target state of a root target or a link target to either DFS_STORAGE_STATE_ONLINE or to DFS_STORAGE_STATE_OFFLINE. Windows Server 2003 does not allow setting the target state of a root target to DFS_STORAGE_STATE_OFFLINE.

<66> Section 3.2.5.1.5: If the *Level* parameter is not equal to one of the valid values, the server fails the call with ERROR_INVALID_LEVEL (0x0000007C).

<67> Section 3.2.5.1.5: The **NetrDfsSetInfo** method supports only the stand-alone DFS namespace in Windows 2000 Server. **NetrDfsSetInfo** supports both stand-alone and domain-based DFS namespaces in Windows Server 2003 and Windows Server 2008. *Level* parameter values 103, 104, 105, and 106 are valid only in Windows Server 2003 SP1, Windows Server 2003 SP2, Windows Server 2003 R2, and Windows Server 2008. *Level* parameter values 107 and 150 are supported only in Windows Vista and Windows Server 2008.

<68> Section 3.2.5.1.6: If the *Level* parameter is not equal to one of the valid values, the server fails the call with ERROR_INVALID_LEVEL (0x0000007C).

<69> Section 3.2.5.1.6: Levels 5, 6, and 7 are supported in Windows Server 2003 SP1, Windows Server 2003 SP2, Windows Server 2003 R2, and Windows Server 2008. Level 7 is supported for domain-based DFS only. It is used to determine whether the DFS metadata of the namespace has changed. Level 50 is supported only in Windows Vista and Windows Server 2008.

<70> Section 3.2.5.1.7: If the *Level* parameter is not equal to one of the valid values, the server fails the call with ERROR_INVALID_LEVEL (0x0000007C).

<71> Section 3.2.5.1.7: *Level* parameter values 5 and 6 are valid only in Windows Server 2003 SP1, Windows Server 2003 SP2, Windows Server 2003 R2, and Windows Server 2008. *Level* parameter values 8 and 9 are supported only in Windows Vista and Windows Server 2008.

<72> Section 3.2.5.1.7: In Windows Server 2003 and Windows Server 2008, the *PrefMaxLen* parameter specifies the number of entries returned, independently of the total size of each entry. In Windows 2000 Server, the *PrefMaxLen* parameter determines the total size of the data returned.

<73> Section 3.2.5.1.7: Windows servers use the *ResumeHandle* parameter as an index into the collection of enumerable items. Due to intervening or concurrent updates, a resumed enumeration may return nonunique or incomplete results.

<74> Section 3.2.5.1.7: The **NetrDfsEnum** method is used only with Windows 2000 Server because there is no parameter to specify the name of a DFS namespace. In Windows Server 2003 and Windows Server 2008, the DFS server can successfully process this method only if it is not hosting more than one DFS namespace root target.

<75> Section 3.2.5.1.7: The server MUST use the same DFS metadata fields as specified in section **NetrDfsGetInfo** to return the required information for a domain-based DFS namespace depending on the *Level* parameter.

<76> Section 3.2.5.1.8: Windows servers perform DFS link move operations atomically for domainv1-based DFS namespaces. Move operations in stand-alone DFS namespaces and domainv2-based DFS namespaces are not atomic.

<77> Section 3.2.5.1.8: Windows servers fail calls with paths that contain wildcards or relative pathname components.

<78> Section 3.2.5.1.8: If there is a conflict between an existing file and a pathname component in the destination path or DFS link of a move operation, Windows servers rename the existing file by appending a ".{GUID}" to the file name, where {GUID} is a newly generated GUID.

<79> Section 3.2.5.1.8: Windows servers remove intermediate directories in the pathname of a source DFS link if they are empty.

<80> Section 3.2.5.1.8: This method is supported only in Windows Server 2003 SP1, Windows Server 2003 SP2, Windows Server 2003 R2, and Windows Server 2008.

<81> Section 3.2.5.1.9: This method is supported only in Windows Vista and Windows Server 2008.

<82> Section 3.2.5.1.10: This method is supported only in Windows Vista and Windows Server 2008.

<83> Section 3.2.5.1.11: This method is supported only in Windows Vista and Windows Server 2008.

<84> Section 3.2.5.2.1: This method supports both stand-alone DFS namespaces and domain-based DFS namespaces in Windows 2000 Server, Windows Server 2003, and Windows Server 2008.

The DFS_RESTORE_VOLUME bit of the *Flags* parameter is used only with Windows 2000 Server.

Windows Server 2003 and Windows Server 2008 ignore the *DcName* and *ppRootList* parameters.

To support down-level compatibility with Windows 2000 Server, Windows clients issue a **NetrDfsSetDcAddress** to each root target listed in *ppRootList* specifying the name of the PDC used for the *DcName* parameter, the NET_DFS_SETDC_INIT_PKT and the NET_DFS_SETDC_TIMEOUT flags for the *Flags* parameter, and a value of 0x00001C20 (7,200 seconds or 2 hours) for the *Timeout* parameter.

This method is not supported in Windows NT 4.0.

<85> Section 3.2.5.2.1: Windows 2000 Server requires the DFS_ADD_VOLUME *Flags* parameter to be specified when creating a new link, Windows Server 2003 and Windows Server 2008 do not.

<86> Section 3.2.5.2.2: This method supports both stand-alone DFS namespaces and domain-based DFS namespaces in Windows 2000 Server, Windows Server 2003, and Windows Server 2008.

The *ppRootList* parameter is not used in Windows Server 2003 and Windows Server 2008.

To support down-level compatibility with Windows 2000 Server, Windows clients issue a **NetrDfsSetDcAddress** to each root target listed in *ppRootList* specifying the name of the PDC used for the *DcName* parameter, the NET_DFS_SETDC_INIT_PKT and the NET_DFS_SETDC_TIMEOUT flags for the *Flags* parameter, and a value of 0x00001C20 (7,200 seconds or 2 hours) for the *Timeout* parameter.

This method is not supported for computers running on Windows NT 4.0.

<87> Section 3.2.5.2.3: While Windows 2000 Server can host at most one root target, Windows Server 2003 and Windows Server 2008 can host more than one root target on the same server. This precludes meaningful use of the **NetrDfsEnum** method by Windows Server 2003 and Windows Server 2008 because **NetrDfsEnum** does not have a parameter to specify the DFS namespace of interest. Hence, the **NetrDfsEnumEx** method is used on Windows Server 2003 and Windows Server 2008.

<88> Section 3.2.5.2.3: Windows NT 4.0 does not support the **NetrDfsEnumEx** method.

<89> Section 3.2.5.2.3: *Level* parameter values 5 and 6 are valid only in Windows Server 2003 SP1, Windows Server 2003 SP2, Windows Server 2003 R2, and Windows Server 2008. *Level* parameter values 8 and 9 are supported only in Windows Vista and Windows Server 2008.

<90> Section 3.2.5.2.3: If the *Level* parameter is not equal to one of the valid values, the server fails the call with ERROR_INVALID_LEVEL (0x0000007C).

<91> Section 3.2.5.2.3: In Windows Server 2003 and Windows Server 2008, the *PrefMaxLen* parameter determines the number of entries returned independently of the total size of each entry. In Windows 2000 Server, this determines the total size of the data returned.

<92> Section 3.2.5.2.3: Due to the possibility of concurrent updates to the DFS namespace, completeness and uniqueness of information returned is not guaranteed when *ResumeHandle* is used to continue an enumeration.

<93> Section 3.2.5.2.4: In Windows 2000 Server, the **NetrDfsSetInfo (Opnum 3)** method supports only stand-alone DFS namespaces. Windows Server 2003 and Windows Server 2008 support both stand-alone and domain-based DFS namespaces for **NetrDfsSetInfo2 (Opnum 22)**.

<94> Section 3.2.5.2.4: Windows NT 4.0 does not support this method.

<95> Section 3.2.5.2.4: Windows 2000 and Windows Server 2008 allow the target state of a root target or a link target to be set to either DFS_STORAGE_STATE_ONLINE or to

DFS_STORAGE_STATE_OFFLINE. Windows Server 2003 does not allow the target state of a root target to be set to DFS_STORAGE_STATE_OFFLINE.

<96> Section 3.2.5.2.4: Windows Server 2008 allows the target state of a root target or a link target to be set to either DFS_STORAGE_STATE_ONLINE or to DFS_STORAGE_STATE_OFFLINE. Windows Server 2003 does not allow the target state of a root target to be set to DFS_STORAGE_STATE_OFFLINE.

<97> Section 3.2.5.2.4: *Level* parameter values 103, 104, 105, and 106 are valid only in Windows Server 2003 SP1, Windows Server 2003 SP2, Windows Server 2003 R2, and Windows Server 2008. *Level* parameter values 107 and 150 are supported only in Windows Vista and Windows Server 2008.

<98> Section 3.2.5.2.4: If the *Level* parameter is not equal to one of the valid values, the server fails the call with ERROR_INVALID_LEVEL (0x0000007C).

<99> Section 3.2.5.3.1: No information is returned through the *ppRootList* parameter in Windows Server 2003 and Windows Server 2008.

To support down-level compatibility with Windows 2000 Server, Windows clients issue a **NetrDfsSetDcAddress (Opnum 17)** method to each root target listed in *ppRootList* specifying the name of the PDC used for the *DcName* parameter, the NET_DFS_SETDC_INIT_PKT and the NET_DFS_SETDC_TIMEOUT flags for the *Flags* parameter, and a value of 0x00001C20 (7,200 seconds or 2 hours) for the *Timeout* parameter.

Windows NT 4.0 does not support this method.

<100> Section 3.2.5.3.2: Windows NT 4.0 does not support the **NetrDfsRemoveFtRoot** method.

<101> Section 3.2.5.3.2: Windows servers do not remove the AD DS object of a domain-based DFS namespace if the last DFS root target is being removed. Windows clients remove the AD DS object of the DFS namespace on successful return from this method.

<102> Section 3.2.5.3.2: The *ppRootList* parameter is not referenced in Windows Server 2003 and Windows Server 2008. In Windows 2000, a list of remaining root targets of the DFS namespace is returned when the RPC call succeeds.

To support down-level compatibility with Windows 2000, Windows clients issue a **NetrDfsSetDcAddress (Opnum 17)** to each root target listed in *ppRootList* specifying the name of the PDC used for the *DcName* parameter, the NET_DFS_SETDC_INIT_PKT and the NET_DFS_SETDC_TIMEOUT flags for the *Flags* parameter, and a value of 0x00001C20 (7,200 seconds or 2 hours) for the *Timeout* parameter.

<103> Section 3.2.5.4.1: The **NetrDfsAddStdRoot (Opnum 12)** method can also be used for clustered DFS with Windows Server 2003 and Windows Server 2008.

<104> Section 3.2.5.4.1: Windows NT 4.0does not support this method.

<105> Section 3.2.5.4.2: Windows NT 4.0 does not support the **NetrDfsRemoveStdRoot** method.

<106> Section 3.2.5.4.3: The **NetrDfsAddStdRootForced** method is used to create a clustered DFS namespace in Windows 2000 Server. This call allows an offline share to host the DFS root.

<107> Section 3.2.5.4.3: Windows Server 2003 and Windows Server 2008 do not support the **NetrDfsAddStdRootForced** method. Use the **NetrDfsAddStdRoot (Opnum 12)** method instead.

<108> Section 3.2.5.4.3: Windows NT 4.0 does not support the **NetrDfsAddStdRootForced (Opnum 15)** method.

<109> Section 3.2.5.5.1: Windows NT 4.0 does not support the **NetrDfsGetDcAddress** method.

<110> Section 3.2.5.5.1: Windows clients ignore the value returned in the *DcName* parameter; Windows Server 2003 and Windows Server 2008 return a blank name.

<111> Section 3.2.5.5.1: Windows Server 2003 and Windows Server 2008 always return FALSE in the *IsRoot* parameter. While Windows Server 2003 Standard Edition supports the ability to host only one DFS namespace, it returns FALSE in the *IsRoot* parameter even when it is hosting a DFS namespace.

The client-side wrapper of the **NetrDfsAddFtRoot (Opnum 10)** RPC method in Windows 2000 Server, Windows XP, Windows Vista, Windows Server 2003, and Windows Server 2008 uses the **NetrDfsGetDcAddress** method to determine whether the server to which the RPC is to be issued is already hosting a DFS namespace. If the value returned in the *IsRoot* parameter is TRUE, the **NetrDfsAddFtRoot ()** call is failed at the client side itself. This is meant for Windows 2000 Server, which supports the ability to host at most one DFS namespace. This is why Windows Server 2003 and Windows Server 2008 always return FALSE for the *IsRoot* parameter.

<112> Section 3.2.5.5.1: In Windows 2000 Server, the default time-out value is 2 hours. This value can be overridden by calling **NetrDfsSetDcAddress (Opnum 17)**.

<113> Section 3.2.5.5.2: Windows NT 4.0 does not support the **NetrDfsSetDcAddress** method.

<114> Section 3.2.5.5.2: To support down-level compatibility with Windows 2000 Server, Windows clients issue a **NetrDfsSetDcAddress (Opnum 17)** to each DFS root target returned in the *ppRootList* parameter from an invocation of **NetrDfsAdd2 (Opnum 19)**, **NetrDfsRemove2 (Opnum 20)**, **NetrDfsSetInfo2 (Opnum 22)**, **NetrDfsAddFtRoot (Opnum 10)**, or **NetrDfsRemoveFtRoot (Opnum 11)** methods. **NetrDfsSetDcAddress (Opnum 17)** specifies the name of the PDC used for the *DcName* parameter, the NET_DFS_SETDC_INIT_PKT and the NET_DFS_SETDC_TIMEOUT flags for the *Flags* parameter, and a value of 0x00001C20 (7,200 seconds or 2 hours) for the *Timeout* parameter.

<115> Section 3.3.5.1.1: Windows Server 2003 and Windows Server 2008 fail NetrDfsEnum level 200 with ERROR_INVALID_PARAMETER (0x00000057); NetrDfsEnum MUST be used for level 200 instead. *Level* parameter value 200 is not supported in Windows 2000 Server.

The NetrDfsEnum method is not supported in Windows NT 4.0.

<116> Section 3.3.5.2.1: Windows NT 4.0 does not support the NetrDfsRemoveFtRoot method.

<117> Section 3.3.5.2.2: Windows Server 2003 and Windows Server 2008 do not support this method and will fail with ERROR_NOT_SUPPORTED (0x00000032).

On a successful call to the **NetrDfsAddFtRoot** or NetrDfsRemoveFtRoot methods, Windows clients call the NetrDfsFlushFtTable method on the PDC of the AD DS domain of the DFS root target server. For more information, see sections 3.1.5.3.1 and 3.1.5.3.2.

This method is not supported in Windows NT 4.0.

<118> Section 5.1: Windows servers use the RPC Protocol to retrieve the identity of the caller, as specified in [MS-RPCE], section 3.3.3.5.3. The server uses the underlying Windows security subsystem to determine the permissions for the caller. If the caller does not have the required permissions to execute a specific method, the method call fails with ERROR_ACCESS_DENIED (0x00000005).

8 Appendix C: XML Schema of XML Document Stored in msDFS-TargetListv2 Attribute

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://schemas.microsoft.com/dfs/2007/03"
            targetNamespace="http://schemas.microsoft.com/dfs/2007/03"
            elementFormDefault="qualified">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Schema document for DFS targets (root or link) for use by the
      domainV2 code in the DFS service. An XML document conforming
      to this schema is stored as the value of an attribute of the
      LDAP entry corresponding to a DFS namespace root or DFS link
      and contains the information on the targets of that DFS
      namespace root or DFS link.

      Conventions:
      - There is a target namespace for this schema document.
          This means instances instances must also declare the same
          namespace for them to be validated using this schema.
      - The elementFormDefault attribute is set to qualified so
          that an instance conforming to this schema can set this
          schema document's namespace as its default namespace and
          have all all unqualified element-type names be considered
          part of the default namespace.
      - Data are in elements, metadata in attributes.

    </xsd:documentation>
  </xsd:annotation>

  <xsd:attributeGroup name="VersionGroup">
     <xsd:attribute name="majorVersion" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:unsignedByte">
          <xsd:minInclusive value="2"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="minorVersion" type="xsd:unsignedByte"
                   use="required"/>

  </xsd:attributeGroup>

  <xsd:simpleType name="TargetStateType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
        Type used for specifying the state of a target.

        This is global to support extension or redefinition.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:token">
      <xsd:enumeration value="online"/>
      <xsd:enumeration value="offline"/>
    </xsd:restriction>
  </xsd:simpleType>
```

```
<xsd:simpleType name="TargetPriorityClassType">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Type used for specifying the priority class of a target.

      This is global to support extension or redefinition.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="siteCostNormal"/>
    <xsd:enumeration value="globalHigh"/>
    <xsd:enumeration value="siteCostHigh"/>
    <xsd:enumeration value="siteCostLow"/>
    <xsd:enumeration value="globalLow"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="TargetPriorityRankType">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Type used for specifying the priority rank of a target.

      This is global to support extension or redefinition.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:unsignedByte">
    <xsd:maxInclusive value="31"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="targets">
  <xsd:complexType>

    <xsd:sequence>

      <xsd:element name="target" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:annotation>
            <xsd:documentation xml:lang="en">
              This is an anonymous complex type with simple content:
              i.e. it allows character data only with no children.
              Of course, attributes are allowed.

              A pattern restriction is used to ensure a UNC path.
              Pathname components cannot have embedded forward
              slashes (/).
            </xsd:documentation>
          </xsd:annotation>

          <xsd:simpleContent>
            <xsd:restriction base="xsd:anyType">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:pattern
                       value="\\\\([^/\\])+(\\([^/\\])+)+(\\)?"/>
                </xsd:restriction>
              </xsd:simpleType>
```

```
                <xsd:attribute name="state" type="TargetStateType"
                               default="online"/>
                <xsd:attribute name="priorityClass"
                               type="TargetPriorityClassType"
                               default="siteCostNormal"/>
                <xsd:attribute name="priorityRank"
                               type="TargetPriorityRankType"
                               default="0"/>

              </xsd:restriction>

            </xsd:simpleContent>

          </xsd:complexType>
        </xsd:element>

      </xsd:sequence>

      <xsd:attributeGroup ref="VersionGroup"/>

      <xsd:attribute name="targetCount" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:unsignedInt">
            <xsd:minInclusive value="1"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>

      <xsd:attribute name="totalStringLengthInBytes"
                     type="xsd:unsignedInt" use="required">
        <xsd:annotation>
          <xsd:documentation>
            To permit a single-pass parsing, this attribute contains
            the length (in bytes) of all strings (including NULL
            termination) that will be retained in that form in an
            in-memory representation.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>

    </xsd:complexType>
  </xsd:element>

</xsd:schema>

<!-- Editor settings. DO NOT delete -->
<!-- vi: set ts=2 sw=2 filetype=xml: -->
```

9 Index