[MS-SRVS]:
Server Service Remote Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
10/22/2006	0.01		MCPP Milestone 1 Initial Availability
01/19/2007	1.0		MCPP Milestone 1
03/02/2007	1.1		Monthly release
04/03/2007	1.2		Monthly release
05/11/2007	1.3		Monthly release

Date	Revision History	Revision Class	Comments
06/01/2007	1.3.1	Editorial	Revised and edited the technical content.
07/03/2007	2.0	Major	Updated and revised the technical content.
07/20/2007	3.0	Major	Updated and revised the technical content.
08/10/2007	4.0	Major	Updated and revised the technical content.
09/28/2007	4.1	Minor	Updated the technical content.
10/23/2007	4.2	Minor	Updated the technical content.
11/30/2007	4.2.1	Editorial	Revised and edited the technical content.
01/25/2008	4.2.2	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies the Server Service Remote Protocol, which is a Microsoft proprietary protocol. The Server Service Remote Protocol is a **remote procedure call (RPC)**–based protocol that is used for remotely enabling file and printer sharing and **named pipe** access to the **server** through the Server Message Block (SMB) Protocol, as specified in [MS-SMB]. The protocol is also used for remote administration of servers that are running Windows.

1.1 Glossary

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

Client
Connection
DFS
DFS Link
DFS Namespace
DFS Root
Endpoint
Globally Unique Identifier (GUID)
Interface Definition Language (IDL)
Mailslot
Named Pipe
Opnum
Remote Procedure Call (RPC)
RPC Protocol Sequence
Server
Server Message Block (SMB)
Session
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

Standalone DFS Namespace: A **DFS namespace** whose configuration information is stored locally in the registry of the root **server**.

Sticky Share: A share that is available after a machine restarts.

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.

[CIFS] Leach, P. and Naik, D., "A Common Internet File System (CIFS/1.0) Protocol", March 1997, http://www.microsoft.com/about/legal/intellectualproperty/protocols/BSTD/CIFS/dr aft-leach-cifs-v1-spec-02.txt

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

[MS-DFSC] Microsoft Corporation, "Distributed File System (DFS): Referral Protocol Specification", June 2007.

[MS-DFSNM] Microsoft Corporation, "Distributed File System (DFS): Namespace Management Protocol Specification", September 2007.

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

[MS-EERR] Microsoft Corporation, "ExtendedError Remote Data Structure", January 2007.

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

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

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol Specification", July 2007.

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

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

[RFC1123] Braden, R., "Requirements for Internet Hosts–Application and Support", RFC 1123, October 1989, http://www.ietf.org/rfc/rfc1123.txt

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

1.2.2 Informative References

[MS-BRWS] Microsoft Corporation, "Common Internet File System (CIFS) Browser Protocol Specification", July 2007.

[MSDN-CoCreateGuid] Microsoft Corporation, "CoCreateGuid", http://msdn2.microsoft.com/en-us/library/ms688568.aspx

[PIPE] Microsoft Corporation, "Named Pipes", http://msdn2.microsoft.com/en-us/library/aa365590.aspx

1.3 Protocol Overview (Synopsis)

The Server Service Remote Protocol is designed for remotely querying and configuring a **Server Message Block (SMB)** server on a remote computer. By using this protocol, a **client** can query and configure information on the server such as active **connections**, **sessions**, shares, files, and transport protocols. Clients can also query and configure the server itself, for instance by setting the server's type, changing the services that are running on the server, or getting a list of all servers of a specific type in a domain.

This is an RPC-based protocol. The server does not maintain client state information. The operation of the protocol is stateless. No sequence of method calls is imposed on this protocol, with the exception of net share deletion, which requires a two-phase commit.

1.4 Relationship to Other Protocols

This protocol depends on RPC and SMB for its transport. This protocol uses RPC over named pipes, as specified in section 2.1. Named pipes use the SMB Protocol, as specified in [MS-SMB].

No other protocols depend on this protocol.

1.5 Prerequisites/Preconditions

The Server Service Remote Protocol is an RPC interface and, as a result, has the prerequisites that are specified in [MS-RPCE] section 1.5 as being common to RPC interfaces.

It is assumed that a Server Service Remote Protocol client has obtained the name of a remote machine that supports the Server Service Remote Protocol before this protocol is invoked. How a client invokes this protocol is outside the scope of this specification.

1.6 Applicability Statement

The Server Service Remote Protocol is applicable to environments that require management and monitoring of a file server. In particular, this protocol provides for the creation, deletion, and management of file shares on the server; and the monitoring and administering of users who access that file server. Therefore, this protocol is applicable to environments that require those features.

The Server Service Remote Protocol is primarily used for the management of file servers that use the SMB protocol, as specified in [MS-SMB]; however, it can also be applied to any file-sharing protocol as long as SMB is available for transport of this protocol.

1.7 Versioning and Capability Negotiation

There are no versioning issues for this protocol.

1.8 Vendor-Extensible Fields

This protocol does not define any vendor-extensible fields.

This protocol uses Win32 error codes. These values are taken from the Windows error number space defined in [MS-EERR]. Vendors SHOULD reuse those values with their indicated meaning. Choosing any other value runs the risk of a collision in the future.<1>

1.9 Standards Assignments

Parameter	Value	Reference
RPC Interface **UUID**	4b324fc8-1670-01d3-1278-5a47bf6ee188	Section 2.1
Pipe Name	\PIPE\srvsvc	Section 2.1

2 Messages

The following sections specify how Server Service Remote Protocol messages are transported and message syntax.

2.1 Transport

The RPC methods that the Server Service Remote Protocol exposes are available on one **endpoint**:

- srvsvc named pipe (RPC protseqs ncacn_np), as specified in [MS-RPCE] section 2.1.1.2.

The Server Service Remote Protocol endpoint is available only over named pipes. For more information about named pipes, see [PIPE].

This protocol MUST use the universally unique identifier (UUID) as specified in section 1.9. The RPC version number is 3.0.

This protocol allows any user to establish a connection to the RPC server. The protocol uses the underlying RPC protocol to retrieve the identity of the caller that made the method call, as specified in [MS-RPCE] section 3.3.3.4.3. The server SHOULD use this identity to perform method-specific access checks as described in section 3.1.4.<2>

2.2 Message Syntax

In addition to RPC base types defined in [C706] and [MS-RPCE], the data types given below are defined in the Microsoft Interface Definition Language (MIDL) specification for this RPC interface.

This protocol uses the following types, as specified in [MS-DTYP].

Type	Reference
DWORD	As specified in [MS-DTYP] section **2.2.7**
GUID	As specified in [MS-DTYP] section **2.3.2**
NET_API_STATUS	As specified in [MS-DTYP] section **2.2.36**
SECURITY_INFORMATION	As specified in [MS-DTYP] section **2.4.7**
WCHAR	As specified in [MS-DTYP] section **2.2.59**

2.2.1 Simple Data Types

2.2.1.1 SRVSVC_HANDLE

SRVSVC_HANDLE: A pointer to a null-terminated Unicode UTF-16 string that MUST specify the DNS name, as specified in [RFC1123] section 2.1, or NetBIOS name, as specified in [RFC1001] section 14 and [RFC1002] section 4, of the remote server on which the method is to execute. This string MUST begin with "\\" (two literal backslash characters).

This type is declared as follows:

```
typedef [handle, string] WCHAR* SRVSVC_HANDLE;
```

2.2.1.2 SHARE_DEL_HANDLE

SHARE_DEL_HANDLE: An RPC context handle returned by the **NetrShareDelStart** method, to be provided as a parameter to the **NetrShareDelCommit** method.

This type is declared as follows:

```
typedef [context_handle] VOID* SHARE_DEL_HANDLE;
```

2.2.1.3 PSHARE_DEL_HANDLE

PSHARE_DEL_HANDLE: Pointer to **SHARE_DEL_HANDLE**.

This type is declared as follows:

```
typedef SHARE_DEL_HANDLE* PSHARE_DEL_HANDLE;
```

2.2.2 Constants

2.2.2.1 Sessionclient Types

The following Unicode UTF-16 string values are used to specify the type of client that established the session.<3>

This is not an exhaustive list of possible values. Other values are possible for other client operating systems. The server MAY return an empty string. Clients MAY generate a string that appropriately describes the client operating system, or MAY not specify a string. Currently, there is no maximum length enforced by the RPC protocol for this value.<4>

Value	Meaning
"DOS LM 1.0"	LAN Manager for MS-DOS 1.0 clients
"DOS LM 2.0"	LAN Manager for MS-DOS 2.0 clients
"OS/2 LM 1.0"	LAN Manager for MS-OS/2 1.0 clients
"OS/2 LM 2.0"	LAN Manager for MS-OS/2 2.0 clients

2.2.2.2 MAX_PREFERRED_LENGTH

A constant of type DWORD and is set to -1. This value is valid as an input parameter to any method in section 3.1.4 that takes a *PreferedMaximumLength* parameter. When specified as an input parameter, this value indicates that the method MUST allocate as much space as the data requires.

2.2.2.3 Session User Flags

The following flags specify information that is related to how a user established a session.

Value	Name	Meaning
0x00000001	SESS_GUEST	The user specified by the sesi1_username member established the session by using a guest account.
0x00000002	SESS_NOENCRYPTION	The user specified by the sesi1_username member established the session without using password encryption.

2.2.2.4 Share Types

The following values are used to specify the type of a shared resource.

Value	Name	Meaning
0x00000000	STYPE_DISKTREE	Disk drive
0x00000001	STYPE_PRINTQ	Print queue
0x00000002	STYPE_DEVICE	Communication device
0x00000003	STYPE_IPC	Interprocess communication (IPC)

Zero or more of the following flags MAY be combined with the values in the previous table to further specify the characteristics of a shared resource.

Value	Name	Meaning
0x80000000	STYPE_SPECIAL	Special share reserved for interprocess communication (IPC$) or remote administration of the server (ADMIN$). Can also refer to administrative shares such as C$, D$, E$, and so forth.
0x40000000	STYPE_TEMPORARY	A temporary share that is not persisted for creation each time the file server initializes.

2.2.2.5 Client-Side Caching (CSC) States

The following values are used to specify states that provide hints to clients about whether to cache files by using the client-side caching with the SMB Protocol, as specified in [MS-SMB].

Value	Name	Meaning
0x00	CSC_CACHE_MANUAL_REINT	The client SHOULD NOT automatically cache every file that it opens from this share.
0x10	CSC_CACHE_AUTO_REINT	The client MAY cache every file that it opens from this share.

Value	Name	Meaning
0x20	CSC_CACHE_VDO	The client MAY cache every file that it opens from this share. Also, the client MAY satisfy the file requests from its local cache.
0x30	CSC_CACHE_NONE	The client MUST NOT cache any files from this share.

2.2.2.6 Platform IDs

The following values specify the information level to use for platform-specific information on the server.<5>

Name	Value (Decimal)
PLATFORM_ID_DOS	300
PLATFORM_ID_OS2	400
PLATFORM_ID_NT	500
PLATFORM_ID_OSF	600
PLATFORM_ID_VMS	700

2.2.2.7 Software Type Flags

One or more of the following flags MAY be combined to specify the type of software a file server is running. The SV_TYPE flags indicate the services that are available on the server.

Value	Name	Meaning
0x00000001	SV_TYPE_WORKSTATION	A server running the WorkStation Service
0x00000002	SV_TYPE_SERVER	A server running the Server Service
0x00000004	SV_TYPE_SQLSERVER	Any server running with SQL Server
0x00000008	SV_TYPE_DOMAIN_CTRL	Primary domain controller
0x00000010	SV_TYPE_DOMAIN_BAKCTRL	Backup domain controller
0x00000020	SV_TYPE_TIME_SOURCE	Server is available as a time source for network time synchronization
0x00000040	SV_TYPE_AFP	Apple File Protocol server
0x00000080	SV_TYPE_NOVELL	Novell server
0x00000100	SV_TYPE_DOMAIN_MEMBER	LAN Manager 2.x domain member
0x40000000	SV_TYPE_LOCAL_LIST_ONLY	Servers maintained by the browser
0x00000200	SV_TYPE_PRINTQ_SERVER	Server sharing print queue
0x00000400	SV_TYPE_DIALIN_SERVER	Server running dial-in service
0x00000800	SV_TYPE_XENIX_SERVER	Xenix server

Value	Name	Meaning
0x00004000	SV_TYPE_SERVER_MFPN	Microsoft File and Print for NetWare
0x00001000	SV_TYPE_NT	Windows Server 2003, Windows XP, Windows 2000, or Windows NT
0x00002000	SV_TYPE_WFW	Server running Windows for Workgroups
0x00008000	SV_TYPE_SERVER_NT	Windows Server 2003, Windows 2000 Server, or server that is not a domain controller
0x00010000	SV_TYPE_POTENTIAL_BROWSER	Server that can run the browser service
0x00020000	SV_TYPE_BACKUP_BROWSER	Server running a browser service as backup
0x00040000	SV_TYPE_MASTER_BROWSER	Server running the master browser service
0x00080000	SV_TYPE_DOMAIN_MASTER	Server running the domain master browser
0x80000000	SV_TYPE_DOMAIN_ENUM	Primary domain
0x00400000	SV_TYPE_WINDOWS	Windows Me, Windows 98, or Windows 95
0xFFFFFFFF	SV_TYPE_ALL	All servers
0x02000000	SV_TYPE_TERMINALSERVER	Terminal Server
0x10000000	SV_TYPE_CLUSTER_NT	Server clusters available in the domain
0x04000000	SV_TYPE_CLUSTER_VS_NT	Cluster virtual servers available in the domain

2.2.2.8 Name Types

The following values specify types of names that are used with the **NetprNameValidate**, **NetprNameCanonicalize**, and **NetprNameCompare** methods.

Value (Decimal)	NameType
1	NAMETYPE_USER
2	NAMETYPE_PASSWORD
3	NAMETYPE_GROUP
4	NAMETYPE_COMPUTER
5	NAMETYPE_EVENT
6	NAMETYPE_DOMAIN
7	NAMETYPE_SERVICE
8	NAMETYPE_NET
9	NAMETYPE_SHARE
10	NAMETYPE_MESSAGE

Value (Decimal)	NameType
11	NAMETYPE_MESSAGEDEST
12	NAMETYPE_SHAREPASSWORD
13	NAMETYPE_WORKGROUP

2.2.2.9 Path Types

The following values specify types of paths used with the **NetprPathType**, **NetprPathCanonicalize**, and **NetprPathCompare** methods.<6>

Name	Value (decimal)	Meaning
ITYPE_UNC_COMPNAME	4144	UNC ComputerName
ITYPE_UNC_WC	4145	UNC Wild Card ComputerName
ITYPE_UNC	4096	UNC Path (must not end with \)
ITYPE_UNC_WC_PATH	4097	UNC Path and WC (? or *)
ITYPE_UNC_SYS_SEM	6400	UNC Semaphore
ITYPE_UNC_SYS_SHMEM	6656	UNC Shared Memory
ITYPE_UNC_SYS_MSLOT	6144	UNC **Mailslot**
ITYPE_UNC_SYS_PIPE	6912	UNC Pipe
ITYPE_UNC_SYS_QUEUE	7680	UNC Queue
ITYPE_PATH_ABSND	8194	Absolute non dot path
ITYPE_PATH_ABSD	8198	Path beginning with \\. or <drive>:\
ITYPE_PATH_RELND	8192	Relative path non dot
ITYPE_PATH_RELD	8196	Relative path beginning with \\.
ITYPE_PATH_ABSND_WC	8195	ITYPE_PATH_ABSND and WC
ITYPE_PATH_ABSD_WC	8199	ITYPE_PATH_ABSD and WC(? or *)
ITYPE_PATH_RELND_WC	8193	ITYPE_PATH_RELND and WC
ITYPE_PATH_RELD_WC	8197	ITYPE_PATH_RELD and WC
ITYPE_PATH_SYS_SEM	10498	Local System Semaphore\path
ITYPE_PATH_SYS_SHMEM	10754	Local System Shared Memory\path
ITYPE_PATH_SYS_MSLOT	10242	Local System Mailslot\path
ITYPE_PATH_SYS_PIPE	11010	Local System Pipe\path
ITYPE_PATH_SYS_COMM	11266	Local System COMM\path

Name	Value (decimal)	Meaning
ITYPE_PATH_SYS_PRINT	11522	Local System PRINT\path
ITYPE_PATH_SYS_QUEUE	11778	Local System QUEUE\path
ITYPE_PATH_SYS_SEM_M	43266	Local System Semaphore
ITYPE_PATH_SYS_SHMEM_M	43522	Local System Shared Memory
ITYPE_PATH_SYS_MSLOT_M	43010	Local System Mailslot
ITYPE_PATH_SYS_PIPE_M	43778	Local System Pipe
ITYPE_PATH_SYS_COMM_M	44034	Local System COMM
ITYPE_PATH_SYS_PRINT_M	44290	Local System PRINT
ITYPE_PATH_SYS_QUEUE_M	44546	Local System QUEUE
ITYPE_DEVICE_DISK	16384	<drive>:
ITYPE_DEVICE_LPT	16400	LPT[1-9][:] or \DEV\LPT[1-9]
ITYPE_DEVICE_COM	16416	COM[1-9][:] or \DEV\COM[1-9]
ITYPE_DEVICE_CON	16448	CON port
ITYPE_DEVICE_NUL	16464	NUL port

2.2.2.10 Windows Error Codes

The following Windows error codes are referenced in this specification.

Value	Return code	Description
0x00000005	ERROR_ACCESS_DENIED	The user does not have access to the requested information.
0x0000007C	ERROR_INVALID_LEVEL	The value that is specified for the level parameter is invalid.
0x00000057	ERROR_INVALID_PARAMETER	One or more of the specified parameters is invalid.
0x000000EA	ERROR_MORE_DATA	More entries are available. Specify a large enough buffer to receive all entries.
0x00000000	NERR_Success	The server processed the message successfully.

2.2.2.11 SHARE_INFO Parameter Error Codes

When an invalid value is specified for a field of the **SHARE_INFO** structure, one of the following values MUST be used to indicate which field contains an invalid value. In the following table, "*" is a wildcard character.

Value (Decimal)	Name	Member Causing Error
1	SHARE_NETNAME_PARMNUM	shi*_netname
3	SHARE_TYPE_PARMNUM	shi*_type
4	SHARE_REMARK_PARMNUM	shi*_remark
5	SHARE_PERMISSIONS_PARMNUM	shi*_permissions
6	SHARE_MAX_USES_PARMNUM	shi*_max_uses
7	SHARE_CURRENT_USES_PARMNUM	shi*_current_uses
8	SHARE_PATH_PARMNUM	shi*_path
9	SHARE_PASSWD_PARMNUM	shi*_passwd
501	SHARE_FILE_SD_PARMNUM	shi*_security_descriptor

2.2.2.12 SERVER_INFO Parameter Error Codes

When an invalid value is specified for a field of the **SERVER_INFO** structure, one of the following values MUST be used to indicate which field contains an invalid value. In the following table, "*" is a wildcard character.

Name	Value (decimal)	Member
SV_PLATFORM_ID_PARMNUM	101	sv*_platform_id
SV_NAME_PARMNUM	102	sv*_name
SV_VERSION_MAJOR_PARMNUM	103	sv*_version_major
SV_VERSION_MINOR_PARMNUM	104	sv*_version_minor
SV_TYPE_PARMNUM	105	sv*_type
SV_COMMENT_PARMNUM	5	sv*_comment
SV_USERS_PARMNUM	107	sv*_users
SV_DISC_PARMNUM	10	sv*_disc
SV_HIDDEN_PARMNUM	16	sv*_hidden
SV_ANNOUNCE_PARMNUM	17	sv*_announce
SV_ANNDELTA_PARMNUM	18	sv*_anndelta
SV_USERPATH_PARMNUM	112	sv*_userpath
SV_SESSOPENS_PARMNUM	501	sv*_sessopens
SV_SESSVCS_PARMNUM	502	sv*_sessvcs
SV_OPENSEARCH_PARMNUM	503	sv*_opensearch
SV_SIZREQBUF_PARMNUM	504	sv*_sizreqbuf

Name	Value (decimal)	Member
SV_INITWORKITEMS_PARMNUM	505	sv*_initworkitems
SV_MAXWORKITEMS_PARMNUM	506	sv*_maxworkitems
SV_RAWWORKITEMS_PARMNUM	507	sv*_rawworkitems
SV_IRPSTACKSIZE_PARMNUM	508	sv*_irpstacksize
SV_MAXRAWBUFLEN_PARMNUM	509	sv*_maxrawbuflen
SV_SESSUSERS_PARMNUM	510	sv*_sessusers
SV_SESSCONNS_PARMNUM	511	sv*_sessconns
SV_MAXNONPAGEDMEMORYUSAGE_PARMNUM	512	sv*_maxnonpagedmemoryusage
SV_MAXPAGEDMEMORYUSAGE_PARMNUM	513	sv*_maxpagedmemoryusage
SV_ENABLESOFTCOMPAT_PARMNUM	514	sv*_enablesoftcompat
SV_ENABLEFORCEDLOGOFF_PARMNUM	515	sv*_enableforcedlogoff
SV_TIMESOURCE_PARMNUM	516	sv*_timesource
SV_ACCEPTDOWNLEVELAPIS_PARMNUM	517	sv*_acceptdownlevelapis
SV_LMANNOUNCE_PARMNUM	518	sv*_lmannounce
SV_DOMAIN_PARMNUM	519	sv*_domain
SV_MAXCOPYREADLEN_PARMNUM	520	sv*_maxcopyreadlen
SV_MAXCOPYWRITELEN_PARMNUM	521	sv*_maxcopywritelen
SV_MINKEEPSEARCH_PARMNUM	522	sv*_minkeepsearch
SV_MAXKEEPSEARCH_PARMNUM	523	sv*_maxkeepsearch
SV_MINKEEPCOMPLSEARCH_PARMNUM	524	sv*_minkeepcomplsearch
SV_MAXKEEPCOMPLSEARCH_PARMNUM	525	sv*_maxkeepcomplsearch
SV_THREADCOUNTADD_PARMNUM	526	sv*_threadcountadd
SV_NUMBLOCKTHREADS_PARMNUM	527	sv*_numblockthreads
SV_SCAVTIMEOUT_PARMNUM	528	sv*_scavtimeout
SV_MINRCVQUEUE_PARMNUM	529	sv*_minrcvqueue
SV_MINFREEWORKITEMS_PARMNUM	530	sv*_minfreeworkitems
SV_XACTMEMSIZE_PARMNUM	531	sv*_xactmemsize
SV_THREADPRIORITY_PARMNUM	532	sv*_threadpriority
SV_MAXMPXCT_PARMNUM	533	sv*_maxmpxct

Name	Value (decimal)	Member
SV_OPLOCKBREAKWAIT_PARMNUM	534	sv*_oplockbreakwait
SV_OPLOCKBREAKRESPONSEWAIT_PARMNUM	535	sv*_oplockbreakresponsewait
SV_ENABLEOPLOCKS_PARMNUM	536	sv*_enableoplocks
SV_ENABLEOPLOCKFORCECLOSE_PARMNUM	537	sv*_enableoplockforceclose
SV_ENABLEFCBOPENS_PARMNUM	538	sv*_enablefcbopens
SV_ENABLERAW_PARMNUM	539	sv*_enableraw
SV_ENABLESHAREDNETDRIVES_PARMNUM	540	sv*_enablesharednetdrives
SV_MINFREECONNECTIONS_PARMNUM	541	sv*_minfreeconnections
SV_MAXFREECONNECTIONS_PARMNUM	542	sv*_maxfreeconnections
SV_INITSESSTABLE_PARMNUM	543	sv*_initsesstable
SV_INITCONNTABLE_PARMNUM	544	sv*_initconntable
SV_INITFILETABLE_PARMNUM	545	sv*_initfiletable
SV_INITSEARCHTABLE_PARMNUM	546	sv*_initsearchtable
SV_ALERTSCHEDULE_PARMNUM	547	sv*_alertschedule
SV_ERRORTHRESHOLD_PARMNUM	548	sv*_errorthreshold
SV_NETWORKERRORTHRESHOLD_PARMNUM	549	sv*_networkerrorthreshold
SV_DISKSPACETHRESHOLD_PARMNUM	550	sv*_diskspacethreshold
SV_MAXLINKDELAY_PARMNUM	552	sv*_maxlinkdelay
SV_MINLINKTHROUGHPUT_PARMNUM	553	sv*_minlinkthroughput
SV_LINKINFOVALIDTIME_PARMNUM	554	sv*_linkinfovalidtime
SV_SCAVQOSINFOUPDATETIME_PARMNUM	555	sv*_scavqosinfoupdatetime
SV_MAXWORKITEMIDLETIME_PARMNUM	556	sv*_maxworkitemidletime

2.2.2.13 DFS Entry Flags

One or more of the following flags MAY be combined to specify details about a **DFS** entry that an SMB file server maintains. For more information about DFS entries, see [MS-DFSC].

Name	Value	Meaning
PKT_ENTRY_TYPE_CAIRO	0x0001	Entry refers to a Windows NT, Windows 2000, or Windows XP server.
PKT_ENTRY_TYPE_MACHINE	0x0002	Entry is a machine volume.
PKT_ENTRY_TYPE_NONCAIRO	0x0004	Entry refers to a pre-Windows NT server.

Name	Value	Meaning
PKT_ENTRY_TYPE_LEAFONLY	0x0008	Entry is a **DFS link**.
PKT_ENTRY_TYPE_OUTSIDE_MY_DOM	0x0010	Entry refers to volume in a foreign domain.
PKT_ENTRY_TYPE_INSITE_ONLY	0x0020	Only give Active Directory in-site referrals.
PKT_ENTRY_TYPE_REFERRAL_SVC	0x0080	Entry refers to a **DFS root**.
PKT_ENTRY_TYPE_PERMANENT	0x0100	Entry cannot be scavenged.
PKT_ENTRY_TYPE_LOCAL	0x0400	Entry refers to local volume.
PKT_ENTRY_TYPE_LOCAL_XPOINT	0x0800	Entry refers to an exit point.
PKT_ENTRY_TYPE_MACH_SHARE	0x1000	Entry refers to a private machine share.
PKT_ENTRY_TYPE_OFFLINE	0x2000	Entry refers to a volume that is offline.

2.2.3 Unions

2.2.3.1 CONNECT_ENUM_UNION

The **CONNECT_ENUM_UNION** union contains information about a connection. It is used in the definition of **CONNECTION_ENUM_STRUCT**.

```
typedef
[switch_type(DWORD)]
union _CONNECT_ENUM_UNION {
  [case(0)]
    CONNECT_INFO_0_CONTAINER* Level0;
  [case(1)]
    CONNECT_INFO_1_CONTAINER* Level1;
} CONNECT_ENUM_UNION;
```

Level0: A pointer to a structure containing information about a connection, as specified in section 2.2.4.3.

Level1: A pointer to a structure containing information about a connection, as specified in section 2.2.4.4.

2.2.3.2 FILE_ENUM_UNION

The **FILE_ENUM_UNION** union contains information about files, devices, and pipes. It is used in the definition of **FILE_ENUM_STRUCT**.

```
typedef
[switch_type(DWORD)]
union _FILE_ENUM_UNION {
  [case(2)]
    FILE_INFO_2_CONTAINER* Level2;
  [case(3)]
    FILE_INFO_3_CONTAINER* Level3;
```

```
} FILE_ENUM_UNION;
```

Level2: A pointer to a structure containing information about a file, device or pipe, as specified in section 2.2.4.8.

Level3: A pointer to a structure containing information about a file, device or pipe, as specified in section 2.2.4.9.

2.2.3.3 FILE_INFO

The **FILE_INFO** union contains information about a file, device, or pipe. This union is used by the **NetrFileGetInfo** method.

```
typedef
[switch_type(unsigned long)]
union _FILE_INFO {
  [case(2)]
    LPFILE_INFO_2 FileInfo2;
  [case(3)]
    LPFILE_INFO_3 FileInfo3;
} FILE_INFO,
 *PFILE_INFO,
 *LPFILE_INFO;
```

FileInfo2: A pointer to a structure that contains information about a file, device, or pipe. For more information, see **FILE_INFO_2 (section 2.2.4.6)**.

FileInfo3: A pointer to a structure that contains information about a file, device, or pipe. For more information, see **FILE_INFO_3 (section 2.2.4.7)**.

2.2.3.4 SESSION_ENUM_UNION

The **SESSION_ENUM_UNION** union contains information about sessions. It is used in the definition of **SESSION_ENUM_STRUCT**.

```
typedef
[switch_type(DWORD)]
union _SESSION_ENUM_UNION {
  [case(0)]
    SESSION_INFO_0_CONTAINER* Level0;
  [case(1)]
    SESSION_INFO_1_CONTAINER* Level1;
  [case(2)]
    SESSION_INFO_2_CONTAINER* Level2;
  [case(10)]
    SESSION_INFO_10_CONTAINER* Level10;
  [case(502)]
    SESSION_INFO_502_CONTAINER* Level502;
} SESSION_ENUM_UNION;
```

Level0: A pointer to a structure that contains information about sessions, as specified in section 2.2.4.11.

Level1: A pointer to a structure that contains information about sessions, as specified in section 2.2.4.17.

Level2: A pointer to a structure that contains information about sessions, as specified in section 2.2.4.18.

Level10: A pointer to a structure that contains information about sessions, as specified in section 2.2.4.19.

Level502: A pointer to a structure that contains information about sessions, as specified in section 2.2.4.20.

2.2.3.5 SHARE_ENUM_UNION

The **SHARE_ENUM_UNION** union contains information about shares. It is used in the definition of **SHARE_ENUM_STRUCT**.

```
typedef
[switch_type(DWORD)]
union _SHARE_ENUM_UNION {
  [case(0)]
    SHARE_INFO_0_CONTAINER* Level0;
  [case(1)]
    SHARE_INFO_1_CONTAINER* Level1;
  [case(2)]
    SHARE_INFO_2_CONTAINER* Level2;
  [case(501)]
    SHARE_INFO_501_CONTAINER* Level501;
  [case(502)]
    SHARE_INFO_502_CONTAINER* Level502;
} SHARE_ENUM_UNION;
```

Level0: A pointer to a structure that contains information about shares, as specified in section 2.2.4.31.

Level1: A pointer to a structure that contains information about shares, as specified in section 2.2.4.32.

Level2: A pointer to a structure that contains information about shares, as specified in section 2.2.4.33.

Level501: A pointer to a structure that contains information about shares, as specified in section 2.2.4.34.

Level502: A pointer to a structure that contains information about shares, as specified in section 2.2.4.35.

2.2.3.6 SHARE_INFO

The **SHARE_INFO** union contains information about a share.

```
typedef
[switch_type(unsigned long)]
union _SHARE_INFO {
  [case(0)]
    LPSHARE_INFO_0 ShareInfo0;
  [case(1)]
    LPSHARE_INFO_1 ShareInfo1;
  [case(2)]
    LPSHARE_INFO_2 ShareInfo2;
  [case(502)]
    LPSHARE_INFO_502_I ShareInfo502;
  [case(1004)]
    LPSHARE_INFO_1004 ShareInfo1004;
  [case(1006)]
    LPSHARE_INFO_1006 ShareInfo1006;
  [case(1501)]
    LPSHARE_INFO_1501_I ShareInfo1501;
  [case(1005)]
    LPSHARE_INFO_1005 ShareInfo1005;
  [case(501)]
    LPSHARE_INFO_501 ShareInfo501;
} SHARE_INFO,
 *PSHARE_INFO,
 *LPSHARE_INFO;
```

ShareInfo0: A pointer to a structure that contains information about a share, as specified in section 2.2.4.22.

ShareInfo1: A pointer to a structure that contains information about a share, as specified in section 2.2.4.23.

ShareInfo2: A pointer to a structure that contains information about a share, as specified in section 2.2.4.24.

ShareInfo502: A pointer to a structure that contains information about a share, as specified in section 2.2.4.26.

ShareInfo1004: A pointer to a structure that contains information about a share, as specified in section 2.2.4.27.

ShareInfo1006: A pointer to a structure that contains information about a share, as specified in section 2.2.4.29.

ShareInfo1501: A pointer to a structure that contains information about a share, as specified in section 2.2.4.30.

ShareInfo1005: A pointer to a structure that contains information about a share, as specified in section 2.2.4.28.

ShareInfo501: A pointer to a structure that contains information about a share, as specified in section 2.2.4.25.

2.2.3.7 SERVER_INFO

The **SERVER_INFO** union contains information about a server.

```
typedef
[switch_type(DWORD)]
union _SERVER_INFO {
  [case(100)]
    LPSERVER_INFO_100 ServerInfo100;
  [case(101)]
    LPSERVER_INFO_101 ServerInfo101;
  [case(102)]
    LPSERVER_INFO_102 ServerInfo102;
  [case(502)]
    LPSERVER_INFO_502 ServerInfo502;
  [case(503)]
    LPSERVER_INFO_503 ServerInfo503;
  [case(599)]
    LPSERVER_INFO_599 ServerInfo599;
  [case(1005)]
    LPSERVER_INFO_1005 ServerInfo1005;
  [case(1107)]
    LPSERVER_INFO_1107 ServerInfo1107;
  [case(1010)]
    LPSERVER_INFO_1010 ServerInfo1010;
  [case(1016)]
    LPSERVER_INFO_1016 ServerInfo1016;
  [case(1017)]
    LPSERVER_INFO_1017 ServerInfo1017;
  [case(1018)]
    LPSERVER_INFO_1018 ServerInfo1018;
  [case(1501)]
    LPSERVER_INFO_1501 ServerInfo1501;
  [case(1502)]
    LPSERVER_INFO_1502 ServerInfo1502;
  [case(1503)]
    LPSERVER_INFO_1503 ServerInfo1503;
  [case(1506)]
    LPSERVER_INFO_1506 ServerInfo1506;
  [case(1510)]
    LPSERVER_INFO_1510 ServerInfo1510;
  [case(1511)]
    LPSERVER_INFO_1511 ServerInfo1511;
  [case(1512)]
    LPSERVER_INFO_1512 ServerInfo1512;
  [case(1513)]
    LPSERVER_INFO_1513 ServerInfo1513;
  [case(1514)]
    LPSERVER_INFO_1514 ServerInfo1514;
  [case(1515)]
    LPSERVER_INFO_1515 ServerInfo1515;
  [case(1516)]
    LPSERVER_INFO_1516 ServerInfo1516;
  [case(1518)]
    LPSERVER_INFO_1518 ServerInfo1518;
  [case(1523)]
    LPSERVER_INFO_1523 ServerInfo1523;
  [case(1528)]
    LPSERVER_INFO_1528 ServerInfo1528;
  [case(1529)]
    LPSERVER_INFO_1529 ServerInfo1529;
  [case(1530)]
```

```
    LPSERVER_INFO_1530 ServerInfo1530;
  [case(1533)]
    LPSERVER_INFO_1533 ServerInfo1533;
  [case(1534)]
    LPSERVER_INFO_1534 ServerInfo1534;
  [case(1535)]
    LPSERVER_INFO_1535 ServerInfo1535;
  [case(1536)]
    LPSERVER_INFO_1536 ServerInfo1536;
  [case(1538)]
    LPSERVER_INFO_1538 ServerInfo1538;
  [case(1539)]
    LPSERVER_INFO_1539 ServerInfo1539;
  [case(1540)]
    LPSERVER_INFO_1540 ServerInfo1540;
  [case(1541)]
    LPSERVER_INFO_1541 ServerInfo1541;
  [case(1542)]
    LPSERVER_INFO_1542 ServerInfo1542;
  [case(1543)]
    LPSERVER_INFO_1543 ServerInfo1543;
  [case(1544)]
    LPSERVER_INFO_1544 ServerInfo1544;
  [case(1545)]
    LPSERVER_INFO_1545 ServerInfo1545;
  [case(1546)]
    LPSERVER_INFO_1546 ServerInfo1546;
  [case(1547)]
    LPSERVER_INFO_1547 ServerInfo1547;
  [case(1548)]
    LPSERVER_INFO_1548 ServerInfo1548;
  [case(1549)]
    LPSERVER_INFO_1549 ServerInfo1549;
  [case(1550)]
    LPSERVER_INFO_1550 ServerInfo1550;
  [case(1552)]
    LPSERVER_INFO_1552 ServerInfo1552;
  [case(1553)]
    LPSERVER_INFO_1553 ServerInfo1553;
  [case(1554)]
    LPSERVER_INFO_1554 ServerInfo1554;
  [case(1555)]
    LPSERVER_INFO_1555 ServerInfo1555;
  [case(1556)]
    LPSERVER_INFO_1556 ServerInfo1556;
} SERVER_INFO,
 *PSERVER_INFO,
 *LPSERVER_INFO;
```

ServerInfo100: A pointer to a structure containing information about a server, as specified in section 2.2.4.38.

ServerInfo101: A pointer to a structure containing information about a server, as specified in section 2.2.4.39.

ServerInfo102: A pointer to a structure containing information about a server, as specified in section 2.2.4.40.

ServerInfo502: A pointer to a structure containing information about a server, as specified in section 2.2.4.41.

ServerInfo503: A pointer to a structure containing information about a server, as specified in section 2.2.4.42.

ServerInfo599: A pointer to a structure containing information about a server, as specified in section 2.2.4.43.

ServerInfo1005: A pointer to a structure containing information about a server, as specified in section 2.2.4.44.

ServerInfo1107: A pointer to a structure containing information about a server, as specified in section 2.2.4.45.

ServerInfo1010: A pointer to a structure containing information about a server, as specified in section 2.2.4.46.

ServerInfo1016: A pointer to a structure containing information about a server, as specified in section 2.2.4.47.

ServerInfo1017: A pointer to a structure containing information about a server, as specified in section 2.2.4.48.

ServerInfo1018: A pointer to a structure containing information about a server, as specified in section 2.2.4.49.

ServerInfo1501: A pointer to a structure containing information about a server, as specified in section 2.2.4.50.

ServerInfo1502: A pointer to a structure containing information about a server, as specified in section 2.2.4.51.

ServerInfo1503: A pointer to a structure containing information about a server, as specified in section 2.2.4.52.

ServerInfo1506: A pointer to a structure containing information about a server, as specified in section 2.2.4.53.

ServerInfo1510: A pointer to a structure containing information about a server, as specified in section 2.2.4.54.

ServerInfo1511: A pointer to a structure containing information about a server, as specified in section 2.2.4.55.

ServerInfo1512: A pointer to a structure containing information about a server, as specified in section 2.2.4.56.

ServerInfo1513: A pointer to a structure containing information about a server, as specified in section 2.2.4.57.

ServerInfo1514: A pointer to a structure containing information about a server, as specified in section 2.2.4.58.

ServerInfo1515: A pointer to a structure containing information about a server, as specified in section 2.2.4.59

ServerInfo1516: A pointer to a structure containing information about a server, as specified in section 2.2.4.60.

ServerInfo1518: A pointer to a structure containing information about a server, as specified in section 2.2.4.61.

ServerInfo1523: A pointer to a structure containing information about a server, as specified in section 2.2.4.62.

ServerInfo1528: A pointer to a structure containing information about a server, as specified in section 2.2.4.63.

ServerInfo1529: A pointer to a structure containing information about a server, as specified in section 2.2.4.64.

ServerInfo1530: A pointer to a structure containing information about a server, as specified in section 2.2.4.65.

ServerInfo1533: A pointer to a structure containing information about a server, as specified in section 2.2.4.66.

ServerInfo1534: A pointer to a structure containing information about a server, as specified in section 2.2.4.67.

ServerInfo1535: A pointer to a structure containing information about a server, as specified in section 2.2.4.68.

ServerInfo1536: A pointer to a structure containing information about a server, as specified in section 2.2.4.69.

ServerInfo1538: A pointer to a structure containing information about a server, as specified in section 2.2.4.70.

ServerInfo1539: A pointer to a structure containing information about a server, as specified in section 2.2.4.71.

ServerInfo1540: A pointer to a structure containing information about a server, as specified in section 2.2.4.72.

ServerInfo1541: A pointer to a structure containing information about a server, as specified in section 2.2.4.73.

ServerInfo1542: A pointer to a structure containing information about a server, as specified in section 2.2.4.74.

ServerInfo1543: A pointer to a structure containing information about a server, as specified in section 2.2.4.75.

ServerInfo1544: A pointer to a structure containing information about a server, as specified in section 2.2.4.76.

ServerInfo1545: A pointer to a structure containing information about a server, as specified in section 2.2.4.77.

ServerInfo1546: A pointer to a structure containing information about a server, as specified in section 2.2.4.78.

ServerInfo1547: A pointer to a structure containing information about a server, as specified in section 2.2.4.79.

ServerInfo1548: A pointer to a structure containing information about a server, as specified in section 2.2.4.80.

ServerInfo1549: A pointer to a structure containing information about a server, as specified in section 2.2.4.81.

ServerInfo1550: A pointer to a structure containing information about a server, as specified in section 2.2.4.82.

ServerInfo1552: A pointer to a structure containing information about a server, as specified in section 2.2.4.83.

ServerInfo1553: A pointer to a structure containing information about a server, as specified in section 2.2.4.84.

ServerInfo1554: A pointer to a structure containing information about a server, as specified in section 2.2.4.85.

ServerInfo1555: A pointer to a structure containing information about a server, as specified in section 2.2.4.86.

ServerInfo1556: A pointer to a structure containing information about a server, as specified in section 2.2.4.87.

2.2.3.8 SERVER_XPORT_ENUM_UNION

The **SERVER_XPORT_ENUM_UNION** union contains information about file server transports.

```
typedef
[switch_type(DWORD)]
union _SERVER_XPORT_ENUM_UNION {
  [case(0)]
    PSERVER_XPORT_INFO_0_CONTAINER Level0;
  [case(1)]
    PSERVER_XPORT_INFO_1_CONTAINER Level1;
  [case(2)]
    PSERVER_XPORT_INFO_2_CONTAINER Level2;
  [case(3)]
    PSERVER_XPORT_INFO_3_CONTAINER Level3;
} SERVER_XPORT_ENUM_UNION;
```

Level0: A pointer to a structure containing information about file server transports, as specified in section 2.2.4.94.

Level1: A pointer to a structure containing information about file server transports, as specified in section 2.2.4.95.

Level2: A pointer to a structure containing information about file server transports, as specified in section 2.2.4.96.

Level3: A pointer to a structure containing information about file server transports, as specified in section 2.2.4.97.

2.2.3.9 TRANSPORT_INFO

The **TRANSPORT_INFO** union contains information about a transport over which a file server is operational.

```
typedef
[switch_type(unsigned long)]
union _TRANSPORT_INFO {
  [case(0)]
    SERVER_TRANSPORT_INFO_0 Transport0;
  [case(1)]
    SERVER_TRANSPORT_INFO_1 Transport1;
  [case(2)]
    SERVER_TRANSPORT_INFO_2 Transport2;
  [case(3)]
    SERVER_TRANSPORT_INFO_3 Transport3;
} TRANSPORT_INFO,
 *PTRANSPORT_INFO,
 *LPTRANSPORT_INFO;
```

Transport0: A pointer to a structure containing information about a file server transport, as specified in section .

Transport1: A pointer to a structure containing information about a file server transport, as specified in section 2.2.4.91.

Transport2: A pointer to a structure containing information about a file server transport, as specified in section 2.2.4.92.

Transport3: A pointer to a structure containing information about a file server transport, as specified in section 2.2.4.93.

2.2.4 Structures

2.2.4.1 CONNECTION_INFO_0

The **CONNECTION_INFO_0** structure contains the identifier of a connection.

```
typedef struct _CONNECTION_INFO_0 {
  DWORD coni0_id;
} CONNECTION_INFO_0,
 *PCONNECTION_INFO_0,
 *LPCONNECTION_INFO_0;
```

coni0_id: Specifies a connection identifier. For more information, see Abstract Data Model (section 3.1.1).

2.2.4.2 CONNECTION_INFO_1

The **CONNECTION_INFO_1** structure contains the identifier of a connection, the number of open files, the connection time, the number of users on the connection, and the type of connection.

```
typedef struct _CONNECTION_INFO_1 {
  DWORD coni1_id;
  DWORD coni1_type;
  DWORD coni1_num_opens;
  DWORD coni1_num_users;
  DWORD coni1_time;
  [string] wchar_t* coni1_username;
  [string] wchar_t* coni1_netname;
} CONNECTION_INFO_1,
 *PCONNECTION_INFO_1,
 *LPCONNECTION_INFO_1;
```

coni1_id: Specifies a connection identifier.

coni1_type: Specifies the type of connection made from the local device name to the shared resource. It MUST be one of the values listed in section 2.2.2.3.

coni1_num_opens: Specifies the number of files that are currently opened using the connection.

coni1_num_users: Specifies the number of users on the connection.

coni1_time: Specifies the number of seconds that the connection has been established.

coni1_username: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the user that is associated with the connection.

coni1_netname: A pointer to a null-terminated Unicode UTF-16 string that specifies either the share name of the server's shared resource, or the DNS name, as specified in [RFC1123] section 2.1, or NetBIOS name, as specified in [RFC1001] section 14 and [RFC1002] section 4, which is the computer name of the client. The value of this member depends on which name was specified as the *qualifier* parameter to the **NetrConnectionEnum (section 3.1.4.1)** method. The name that is not specified in the *qualifier* parameter to **NetrConnectionEnum** MUST be automatically returned in the coni1_netname field.

2.2.4.3 CONNECT_INFO_0_CONTAINER

The **CONNECT_INFO_0_CONTAINER** structure contains a value that indicates the number of entries that the **NetrConnectionEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _CONNECT_INFO_0_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPCONNECTION_INFO_0 Buffer;
} CONNECT_INFO_0_CONTAINER,
 *PCONNECT_INFO_0_CONTAINER,
 *LPCONNECT_INFO_0_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **CONNECTION_INFO_0** entries returned by the method.

2.2.4.4 CONNECT_INFO_1_CONTAINER

The **CONNECT_INFO_1_CONTAINER** structure contains a value that indicates the number of entries that the **NetrConnectionEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _CONNECT_INFO_1_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPCONNECTION_INFO_1 Buffer;
} CONNECT_INFO_1_CONTAINER,
 *PCONNECT_INFO_1_CONTAINER,
 *LPCONNECT_INFO_1_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: Pointer to the **CONNECTION_INFO_1** entries returned by the method.

2.2.4.5 CONNECT_ENUM_STRUCT

The **CONNECT_ENUM_STRUCT** structure specifies the information level that the client requests when invoking the **NetrConnectionEnum** method and encapsulates the **CONNECT_ENUM_UNION** union that receives the entries that are enumerated by the server.

```
typedef struct _tag_CONNECT_ENUM_STRUCT {
  DWORD Level;
  [switch_is(Level)] CONNECT_ENUM_UNION ConnectInfo;
} CONNECT_ENUM_STRUCT,
 *PCONNECT_ENUM_STRUCT,
 *LPCONNECT_ENUM_STRUCT;
```

Level: Specifies the information level of the data. It MUST be one of the following values.

Value	Meaning
0	CONNECT_INFO_0_CONTAINER
1	CONNECT_INFO_1_CONTAINER

ConnectInfo: Contains either a **CONNECT_INFO_0_CONTAINER** structure or a **CONNECT_INFO_1_CONTAINER** structure depending on the value of the **Level** parameter. The enumerated elements are returned in this member.

2.2.4.6 FILE_INFO_2

The **FILE_INFO_2** structure contains the identifier for a file, device, or pipe.

```
typedef struct _FILE_INFO_2 {
```

```
  DWORD fi2_id;
} FILE_INFO_2,
 *PFILE_INFO_2,
 *LPFILE_INFO_2;
```

fi2_id: Specifies a DWORD value that contains the identifier that is assigned to the file, device, or pipe when it was opened. See section 3.1.1 for details.

2.2.4.7 FILE_INFO_3

The **FILE_INFO_3** structure contains the identifier and other pertinent information about files, devices, and pipes.

```
typedef struct _FILE_INFO_3 {
  DWORD fi3_id;
  DWORD fi3_permissions;
  DWORD fi3_num_locks;
  [string] wchar_t* fi3_path_name;
  [string] wchar_t* fi3_username;
} FILE_INFO_3,
 *PFILE_INFO_3,
 *LPFILE_INFO_3;
```

fi3_id: Specifies a DWORD value that contains the identifier that is assigned to the file, device, or pipe when it was opened. See section 3.1.1 for details.

fi3_permissions: Specifies a DWORD value that contains the access permissions that are associated with the opening application. This member MUST be a combination of one or more of the following values.

Value	Meaning
PERM_FILE_READ 0x1	Permission to read a resource, and, by default, execute the resource.
PERM_FILE_WRITE 0x2	Permission to write to a resource.
PERM_FILE_CREATE 0x4	Permission to create a resource; data can be written when creating the resource.

fi3_num_locks: Specifies a DWORD value that contains the number of file locks on the file, device, or pipe.

fi3_path_name: A pointer to a string that specifies the path of the opened file, device, or pipe.

fi3_username: A pointer to a string that specifies which user opened the file, device, or pipe.

2.2.4.8 FILE_INFO_2_CONTAINER

The **FILE_INFO_2_CONTAINER** structure contains a value that indicates the number of entries that the **NetrFileEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _FILE_INFO_2_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPFILE_INFO_2 Buffer;
} FILE_INFO_2_CONTAINER,
 *PFILE_INFO_2_CONTAINER,
 *LPFILE_INFO_2_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **FILE_INFO_2** entries returned by the method.

2.2.4.9 FILE_INFO_3_CONTAINER

The **FILE_INFO_3_CONTAINER** structure contains a value that indicates the number of entries that the **NetrFileEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _FILE_INFO_3_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPFILE_INFO_3 Buffer;
} FILE_INFO_3_CONTAINER,
 *PFILE_INFO_3_CONTAINER,
 *LPFILE_INFO_3_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **FILE_INFO_3** entries returned by the method.

2.2.4.10 FILE_ENUM_STRUCT

The **FILE_ENUM_STRUCT** structure specifies the information level that the client requests in the **NetrFileEnum** method and encapsulates the **FILE_ENUM_UNION** union that receives the entries that are enumerated by the server.

```
typedef struct _FILE_ENUM_STRUCT {
  DWORD Level;
  [switch_is(Level)] FILE_ENUM_UNION FileInfo;
} FILE_ENUM_STRUCT,
 *PFILE_ENUM_STRUCT,
 *LPFILE_ENUM_STRUCT;
```

Level: Specifies the information level of the data. It MUST be one of the following values.

Value	Meaning
2	FILE_INFO_2_CONTAINER
3	FILE_INFO_2_CONTAINER

FileInfo: Contains a file info container structure whose type MUST be determined by the *Level* parameter as shown in the previous table. The enumerated elements are returned in this member.

2.2.4.11 SESSION_INFO_0

The **SESSION_INFO_0** structure contains the name of the computer that established the session.

```
typedef struct _SESSION_INFO_0 {
  [string] wchar_t* sesi0_cname;
} SESSION_INFO_0,
 *PSESSION_INFO_0,
 *LPSESSION_INFO_0;
```

sesi0_cname: A pointer to a null-terminated Unicode UTF-16 string that MUST specify the DNS name, as specified in [RFC1123] section 2.1, or NetBIOS name, as specified in [RFC1001] section 14 and [RFC1002] section 4, of the computer that established the session.

2.2.4.12 SESSION_INFO_1

The **SESSION_INFO_1** structure contains information about the session, including the name of the computer; name of the user; and open files, pipes, and devices that are on the computer.

```
typedef struct _SESSION_INFO_1 {
  [string] wchar_t* sesi1_cname;
  [string] wchar_t* sesi1_username;
  DWORD sesi1_num_opens;
  DWORD sesi1_time;
  DWORD sesi1_idle_time;
  DWORD sesi1_user_flags;
} SESSION_INFO_1,
 *PSESSION_INFO_1,
 *LPSESSION_INFO_1;
```

sesi1_cname: A pointer to a null-terminated Unicode UTF-16 string specifying the DNS name, as specified in [RFC1123] section 2.1, or NetBIOS name, as specified in [RFC1001] section 14 and [RFC1002] section 4, of the computer that established the session.

sesi1_username: A pointer to a null-terminated Unicode UTF-16 string specifying the name of the user who established the session.

sesi1_num_opens: Specifies a DWORD value that contains the number of files, devices, and pipes that were opened during the session.

sesi1_time: Specifies a DWORD value that contains the number of seconds since the session was created.

sesi1_idle_time: Specifies a DWORD value that contains the number of seconds the session has been idle.

sesi1_user_flags: Specifies a DWORD value that MUST specify how the user established the session. This member MUST be a combination of one or more of the values that are defined in 2.2.2.3.

2.2.4.13 SESSION_INFO_2

The **SESSION_INFO_2** structure contains information about the session, including the name of the computer; name of the user; open files, pipes, and devices that are on the computer; and the type of client that established the session.

```
typedef struct _SESSION_INFO_2 {
  [string] wchar_t* sesi2_cname;
  [string] wchar_t* sesi2_username;
  DWORD sesi2_num_opens;
  DWORD sesi2_time;
  DWORD sesi2_idle_time;
  DWORD sesi2_user_flags;
  [string] wchar_t* sesi2_cltype_name;
} SESSION_INFO_2,
 *PSESSION_INFO_2,
 *LPSESSION_INFO_2;
```

sesi2_cname: A pointer to a null-terminated Unicode UTF-16 string specifying the DNS name, as specified in [RFC1123] section 2.1, or NetBIOS name, as specified in [RFC1001] section 14 and [RFC1002] section 4, of the computer that established the session.

sesi2_username: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the user who established the session.

sesi2_num_opens: Specifies a DWORD value that contains the number of files, devices, and pipes that were opened during the session.

sesi2_time: Specifies a DWORD value that contains the number of seconds the session has been active.

sesi2_idle_time: Specifies a DWORD value that contains the number of seconds the session has been idle.

sesi2_user_flags: Specifies a DWORD value that describes how the user established the session. This member MUST be a combination of one or more of the values that are defined in section 2.2.2.3.

sesi2_cltype_name: A pointer to a null-terminated Unicode UTF-16 string that specifies the type of client that established the session. The server simply stores this string, as specified in section 2.2.2.1, and its value does not modify the behavior of the protocol.

2.2.4.14 SESSION_INFO_10

The **SESSION_INFO_10** structure contains information about the session, including the name of the computer, name of the user, and active and idle times for the session.

```
typedef struct _SESSION_INFO_10 {
  [string] wchar_t* sesi10_cname;
  [string] wchar_t* sesi10_username;
```

```
  DWORD sesi10_time;
  DWORD sesi10_idle_time;
} SESSION_INFO_10,
 *PSESSION_INFO_10,
 *LPSESSION_INFO_10;
```

sesi10_cname: A pointer to a null-terminated Unicode UTF-16 string specifying the DNS name, as specified in [RFC1123] section 2.1, or NetBIOS name, as specified in [RFC1001] section 14 and [RFC1002] section 4, of the computer that established the session.

sesi10_username: A pointer to a null-terminated Unicode UTF-16 string specifying the name of the user who established the session.

sesi10_time: Specifies the number of seconds the session has been active.

sesi10_idle_time: Specifies the number of seconds the session has been idle.

2.2.4.15 SESSION_INFO_502

The **SESSION_INFO_502** structure contains information about the session, including the name of the computer; the name of the user; open files, pipes, and devices that are on the computer; the client type; session active and idle times; how the user established the session; and the name of the transport that the client is using.

```
typedef struct _SESSION_INFO_502 {
  [string] wchar_t* sesi502_cname;
  [string] wchar_t* sesi502_username;
  DWORD sesi502_num_opens;
  DWORD sesi502_time;
  DWORD sesi502_idle_time;
  DWORD sesi502_user_flags;
  [string] wchar_t* sesi502_cltype_name;
  [string] wchar_t* sesi502_transport;
} SESSION_INFO_502,
 *PSESSION_INFO_502,
 *LPSESSION_INFO_502;
```

sesi502_cname: A pointer to a null-terminated Unicode UTF-16 string specifying the DNS name, as specified in [RFC1123] section 2.1, or the NetBIOS name, as specified in [RFC1001] section 14 and [RFC1002] section 4, of the computer that established the session.

sesi502_username: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the user who established the session.

sesi502_num_opens: Specifies the number of files, devices, and pipes that were opened during the session.

sesi502_time: Specifies the number of seconds the session has been active.

sesi502_idle_time: Specifies the number of seconds the session has been idle.

sesi502_user_flags: Specifies a value that describes how the user established the session. This member MUST be a combination of one or more of the values that are listed in section 2.2.2.3.

sesi502_cltype_name: A pointer to a null-terminated Unicode UTF-16 string that specifies the type of client that established the session. The server simply stores this string, as specified in section 2.2.2.1, and its value does not modify the behavior of the protocol.

sesi502_transport: Specifies the name of the transport that the client is using to communicate with the server.

2.2.4.16 SESSION_INFO_0_CONTAINER

The **SESSION_INFO_0_CONTAINER** structure contains a value that indicates the number of entries that the **NetrSessionEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SESSION_INFO_0_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSESSION_INFO_0 Buffer;
} SESSION_INFO_0_CONTAINER,
 *PSESSION_INFO_0_CONTAINER,
 *LPSESSION_INFO_0_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **SESSION_INFO_0** entries returned by the method.

2.2.4.17 SESSION_INFO_1_CONTAINER

The **SESSION_INFO_1_CONTAINER** structure contains a value that indicates the number of entries that the **NetrSessionEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SESSION_INFO_1_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSESSION_INFO_1 Buffer;
} SESSION_INFO_1_CONTAINER,
 *PSESSION_INFO_1_CONTAINER,
 *LPSESSION_INFO_1_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **SESSION_INFO_1** entries returned by the method.

2.2.4.18 SESSION_INFO_2_CONTAINER

The **SESSION_INFO_2_CONTAINER** structure contains a value that indicates the number of entries that the **NetrSessionEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SESSION_INFO_2_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSESSION_INFO_2 Buffer;
} SESSION_INFO_2_CONTAINER,
 *PSESSION_INFO_2_CONTAINER,
 *LPSESSION_INFO_2_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **SESSION_INFO_2** entries returned by the method.

2.2.4.19 SESSION_INFO_10_CONTAINER

The **SESSION_INFO_10_CONTAINER** structure contains a value that indicates the number of entries that the **NetrSessionEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SESSION_INFO_10_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSESSION_INFO_10 Buffer;
} SESSION_INFO_10_CONTAINER,
 *PSESSION_INFO_10_CONTAINER,
 *LPSESSION_INFO_10_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **SESSION_INFO_10** entries returned by the method.

2.2.4.20 SESSION_INFO_502_CONTAINER

The **SESSION_INFO_502_CONTAINER** structure contains a value that indicates the number of entries that the **NetrSessionEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SESSION_INFO_502_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSESSION_INFO_502 Buffer;
} SESSION_INFO_502_CONTAINER,
 *PSESSION_INFO_502_CONTAINER,
 *LPSESSION_INFO_502_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **SESSION_INFO_502** entries returned by the method.

2.2.4.21 SESSION_ENUM_STRUCT

The **SESSION_ENUM_STRUCT** structure specifies the information level that the client requests in the **NetrSessionEnum** method and encapsulates the **SESSION_ENUM_UNION** union that receives the entries that are enumerated by the server.

```
typedef struct _SESSION_ENUM_STRUCT {
  DWORD Level;
  [switch_is(Level)] SESSION_ENUM_UNION SessionInfo;
} SESSION_ENUM_STRUCT,
 *PSESSION_ENUM_STRUCT,
 *LPSESSION_ENUM_STRUCT;
```

Level: Specifies the information level of the data. It MUST be one of the following values.

Value	Meaning
0	SESSION_INFO_0_CONTAINER
1	SESSION_INFO_1_CONTAINER
2	SESSION_INFO_2_CONTAINER
10	SESSION_INFO_10_CONTAINER
502	SESSION_INFO_502_CONTAINER

SessionInfo: Contains a session info container whose type is specified by the *Level* parameter, as shown in the above table. The enumerated session entries are returned in this member.

2.2.4.22 SHARE_INFO_0

The **SHARE_INFO_0** structure contains the name of the shared resource.

```
typedef struct _SHARE_INFO_0 {
  [string] wchar_t* shi0_netname;
} SHARE_INFO_0,
 *PSHARE_INFO_0,
 *LPSHARE_INFO_0;
```

shi0_netname: A pointer to a null-terminated Unicode UTF-16 string specifying the share name of a resource.

2.2.4.23 SHARE_INFO_1

The **SHARE_INFO_1** structure contains information about the shared resource, including the name and type of the resource, and a comment associated with the resource.

```
typedef struct _SHARE_INFO_1 {
  [string] wchar_t* shi1_netname;
  DWORD shi1_type;
```

```
  [string] wchar_t* shi1_remark;
} SHARE_INFO_1,
 *PSHARE_INFO_1,
 *LPSHARE_INFO_1;
```

shi1_netname: A pointer to a null-terminated Unicode UTF-16 string specifying the share name of a resource. The server MUST ignore this member when processing the **NetrShareSetInfo** method.

shi1_type: Specifies a DWORD value that contains the type of the shared resource. The server MUST ignore this member when processing the **NetrShareSetInfo** method. It MUST be one of the Share Type values in section 2.2.2.4.

shi1_remark: A pointer to a null-terminated Unicode UTF-16 string specifying an optional comment about the shared resource.

2.2.4.24 SHARE_INFO_2

The **SHARE_INFO_2** structure contains information about the shared resource, including name of the resource, type and permissions, and the number of current connections.

```
typedef struct _SHARE_INFO_2 {
  [string] wchar_t* shi2_netname;
  DWORD shi2_type;
  [string] wchar_t* shi2_remark;
  DWORD shi2_permissions;
  DWORD shi2_max_uses;
  DWORD shi2_current_uses;
  [string] wchar_t* shi2_path;
  [string] wchar_t* shi2_passwd;
} SHARE_INFO_2,
 *PSHARE_INFO_2,
 *LPSHARE_INFO_2;
```

shi2_netname: A pointer to a null-terminated Unicode UTF-16 string specifying the share name of a resource. The server MUST ignore this member when processing the **NetrShareSetInfo** method.

shi2_type: Specifies a DWORD value that contains the type of the shared resource. The server MUST ignore this member when processing the **NetrShareSetInfo** method. It MUST be one of the values in section 2.2.2.4.

shi2_remark: A pointer to a null-terminated Unicode UTF-16 string that contains an optional comment about the shared resource.

shi2_permissions: This field is not used. The client MUST send a value of 0 and the server MUST ignore the value of this parameter upon receipt.

shi2_max_uses: Specifies a DWORD value that indicates the maximum number of concurrent connections that the shared resource can accommodate. If the value specified in **shi2_max_uses** is -1, the maximum number of connections MUST be unlimited.

shi2_current_uses: Specifies a DWORD value that indicates the number of current connections to the resource. The server MUST ignore this member upon receipt.

shi2_path: A pointer to a null-terminated Unicode UTF-16 string specifying the local path for the shared resource. For disks, **shi2_path** is the path that is being shared. For print queues, **shi2_path** is the name of the print queue that is being shared. The server MUST ignore this member when processing the **NetrShareSetInfo** method.

shi2_passwd: This field is not used. The client MUST send a NULL (zero length) string and the server MUST ignore the value of this parameter upon receipt.

2.2.4.25 SHARE_INFO_501

The **SHARE_INFO_501** structure contains information about the shared resource, including the name and type of the resource, and a comment that is associated with the resource.

```
typedef struct _SHARE_INFO_501 {
  [string] wchar_t* shi501_netname;
  DWORD shi501_type;
  [string] wchar_t* shi501_remark;
  DWORD shi501_flags;
} SHARE_INFO_501,
 *PSHARE_INFO_501,
 *LPSHARE_INFO_501;
```

shi501_netname: A pointer to a null-terminated Unicode UTF-16 string specifying the name of a shared resource.

shi501_type: Specifies a DWORD value that indicates the type of share. It MUST be one of the values that is listed in section 2.2.2.4.

shi501_remark: A pointer to a null-terminated Unicode UTF-16 string specifying an optional comment about the shared resource.

shi501_flags: Reserved; MUST be zero.

2.2.4.26 SHARE_INFO_502_I

The **SHARE_INFO_502_I** structure contains information about the shared resource, including the name of the resource, type and permissions, number of connections, and other pertinent information.

```
typedef struct _SHARE_INFO_502_I {
  [string] WCHAR* shi502_netname;
  DWORD shi502_type;
  [string] WCHAR* shi502_remark;
  DWORD shi502_permissions;
  DWORD shi502_max_uses;
  DWORD shi502_current_uses;
  [string] WCHAR* shi502_path;
  [string] WCHAR* shi502_passwd;
  DWORD shi502_reserved;
  [size_is(shi502_reserved)] unsigned char* shi502_security_descriptor;
} SHARE_INFO_502_I,
```

```
 *PSHARE_INFO_502_I,
 *LPSHARE_INFO_502_I;
```

shi502_netname: A pointer to a null-terminated Unicode UTF-16 string specifying the name of a shared resource. The server MUST ignore this member when processing the **NetrShareSetInfo (section 3.1.4.11)** method.

shi502_type: Specifies a DWORD value that indicates the type of share. The server MUST ignore this member when processing the **NetrShareSetInfo** method. It MUST be one of the values that are listed in section 2.2.2.4.

shi502_remark: A pointer to a null-terminated Unicode UTF-16 string specifying an optional comment about the shared resource.

shi502_permissions: This field is not used. The client MUST send a value of 0 and the server MUST ignore the value of this parameter upon receipt.

shi502_max_uses: Specifies a DWORD value that indicates the maximum number of concurrent connections that the shared resource can accommodate. If the value specified **shi502_max_uses** is -1, the maximum number of connections MUST be unlimited.

shi502_current_uses: Specifies a DWORD value that indicates the number of current connections to the resource. The server MUST ignore this member upon receipt.

shi502_path: A pointer to a null-terminated Unicode UTF-16 string that contains the local path for the shared resource. For disks, **shi502_path** is the path that is being shared. For print queues, **shi502_path** is the name of the print queue that is being shared. The server MUST ignore this member when processing the **NetrShareSetInfo** method.

shi502_passwd: This field is not used. The client MUST send a NULL (zero length) string and the server MUST ignore the value of this parameter upon receipt.

shi502_reserved: Length of security descriptor that is being passed in **shi502_security_descriptor**. The server MUST accept any valid DWORD value for this field.

shi502_security_descriptor: Specifies the SECURITY_DESCRIPTOR, as described in [MS-DTYP] section 2.4.6, that is associated with this share.

2.2.4.27 SHARE_INFO_1004

The **SHARE_INFO_1004** structure contains a comment that is associated with the shared resource.

```
typedef struct _SHARE_INFO_1004 {
  [string] wchar_t* shi1004_remark;
} SHARE_INFO_1004,
 *PSHARE_INFO_1004,
 *LPSHARE_INFO_1004;
```

shi1004_remark: A pointer to a null-terminated Unicode UTF-16 string that contains an optional comment about the shared resource.

2.2.4.28 SHARE_INFO_1005

The **SHARE_INFO_1005** structure contains information about the shared resource.<7>

```
typedef struct _SHARE_INFO_1005 {
  DWORD shi1005_flags;
} SHARE_INFO_1005,
 *PSHARE_INFO_1005,
 *LPSHARE_INFO_1005;
```

shi1005_flags: Specifies a **DWORD** bitmask value that MUST contain zero or more of the following values. The bit locations that are named CSC_MASK in the following table MUST contain a client-side caching state value as given in section 2.2.2.5. The server MUST ignore SHI1005_FLAGS_DFS and SHI1005_FLAGS_DFS_ROOT when processing the **NetrShareSetInfo** method.<8>

Value	Meaning
SHI1005_FLAGS_DFS 0x00000001	The specified share is present in a DFS tree structure.
SHI1005_FLAGS_DFS_ROOT 0x00000002	The specified share is the root volume in a DFS tree structure.
CSC_MASK	Provides a mask for one of the four possible client-side caching (CSC) (section 2.2.2.5) states.
SHI1005_FLAGS_RESTRICT_EXCLUSIVE_OPENS 0x00000100	The specified share disallows exclusive file opens that deny reads to an open file.
SHI1005_FLAGS_FORCE_SHARED_DELETE 0x00000200	Shared files in the specified share can be forcibly deleted.
SHI1005_FLAGS_ALLOW_NAMESPACE_CACHING 0x00000400	Clients are allowed to cache the namespace of the specified share.
SHI1005_FLAGS_ACCESS_BASED_DIRECTORY_ENUM 0x00000800	The server will filter directory entries based on the access permissions of the client.

2.2.4.29 SHARE_INFO_1006

The **SHARE_INFO_1006** structure specifies the maximum number of concurrent connections that the shared resource can accommodate.

```
typedef struct _SHARE_INFO_1006 {
  DWORD shi1006_max_uses;
} SHARE_INFO_1006,
 *PSHARE_INFO_1006,
 *LPSHARE_INFO_1006;
```

shi1006_max_uses: Specifies a DWORD value that indicates the maximum number of concurrent connections that the shared resource can accommodate. If the value specified in **shi1006_max_uses** is -1, the maximum number of connections MUST be unlimited.

2.2.4.30 SHARE_INFO_1501_I

The **SHARE_INFO_1501_I** structure contains a security descriptor in self-relative format and a DWORD that contains its length.<9>

```
typedef struct _SHARE_INFO_1501_I {
  DWORD shi1501_reserved;
  [size_is(shi1501_reserved)] unsigned char* shi1501_security_descriptor;
} SHARE_INFO_1501_I,
 *PSHARE_INFO_1501_I,
 *LPSHARE_INFO_1501_I;
```

shi1501_reserved: Length of security descriptor that is being passed in **shi501_security_descriptor**.

shi1501_security_descriptor: Specifies the SECURITY_DESCRIPTOR, as specified in [MS-DTYP] section 2.4.6, that is associated with this share.

2.2.4.31 SHARE_INFO_0_CONTAINER

The **SHARE_INFO_0_CONTAINER** structure contains a value that indicates the number of entries that the **NetrShareEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SHARE_INFO_0_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSHARE_INFO_0 Buffer;
} SHARE_INFO_0_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **SHARE_INFO_0** entries returned by the method.

2.2.4.32 SHARE_INFO_1_CONTAINER

The **SHARE_INFO_1_CONTAINER** structure contains a value that indicates the number of entries that the **NetrShareEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SHARE_INFO_1_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSHARE_INFO_1 Buffer;
} SHARE_INFO_1_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **SHARE_INFO_1** entries returned by the method.

2.2.4.33 SHARE_INFO_2_CONTAINER

The **SHARE_INFO_2_CONTAINER** structure contains a value that indicates the number of entries that the **NetrShareEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SHARE_INFO_2_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSHARE_INFO_2 Buffer;
} SHARE_INFO_2_CONTAINER,
 *PSHARE_INFO_2_CONTAINER,
 *LPSHARE_INFO_2_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **SHARE_INFO_2** entries returned by the method.

2.2.4.34 SHARE_INFO_501_CONTAINER

The **SHARE_INFO_501_CONTAINER** structure contains a value that indicates the number of entries that the **NetrShareEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SHARE_INFO_501_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSHARE_INFO_501 Buffer;
} SHARE_INFO_501_CONTAINER,
 *PSHARE_INFO_501_CONTAINER,
 *LPSHARE_INFO_501_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **SHARE_INFO_501** entries returned by the method.

2.2.4.35 SHARE_INFO_502_CONTAINER

The **SHARE_INFO_502_CONTAINER** structure contains a value that indicates the number of entries that the **NetrShareEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SHARE_INFO_502_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSHARE_INFO_502_I Buffer;
} SHARE_INFO_502_CONTAINER,
 *PSHARE_INFO_502_CONTAINER,
 *LPSHARE_INFO_502_CONTAINER;
```

EntriesRead: Number of entries returned by the method.

Buffer: A pointer to the **SHARE_INFO_502_I** entries returned by the method.

2.2.4.36 SHARE_ENUM_STRUCT

The **SHARE_ENUM_STRUCT** structure specifies the information level that the client requests in the **NetrShareEnum** method and encapsulates the **SHARE_ENUM_UNION** union that receives the entries enumerated by the server.

```
typedef struct _SHARE_ENUM_STRUCT {
  DWORD Level;
  [switch_is(Level)] SHARE_ENUM_UNION ShareInfo;
} SHARE_ENUM_STRUCT,
 *PSHARE_ENUM_STRUCT,
 *LPSHARE_ENUM_STRUCT;
```

Level: Specifies the information level of the data. It MUST be one of the following values.

Value	Meaning
0	SHARE_INFO_0_CONTAINER
1	SHARE_INFO_1_CONTAINER
2	SHARE_INFO_2_CONTAINER
501	SHARE_INFO_501_CONTAINER
502	SHARE_INFO_502_CONTAINER

ShareInfo: Contains a share information container whose type is specified by the *Level* parameter as the table above shows. The enumerated share entries are returned in this member.

2.2.4.37 STAT_SERVER_0

The **STAT_SERVER_0** structure contains statistical information about the server.

```
typedef struct _STAT_SERVER_0 {
  DWORD sts0_start;
  DWORD sts0_fopens;
  DWORD sts0_devopens;
  DWORD sts0_jobsqueued;
  DWORD sts0_sopens;
  DWORD sts0_stimedout;
  DWORD sts0_serrorout;
  DWORD sts0_pwerrors;
  DWORD sts0_permerrors;
  DWORD sts0_syserrors;
  DWORD sts0_bytessent_low;
  DWORD sts0_bytessent_high;
  DWORD sts0_bytesrcvd_low;
  DWORD sts0_bytesrcvd_high;
  DWORD sts0_avresponse;
  DWORD sts0_reqbufneed;
  DWORD sts0_bigbufneed;
} STAT_SERVER_0,
```

```
*PSTAT_SERVER_0,
*LPSTAT_SERVER_0;
```

sts0_start: Specifies a DWORD value that indicates the time when statistics collection started (or when the statistics were last cleared). The value MUST be stored as the number of seconds that have elapsed since 00:00:00, January 1, 1970, Greenwich Mean Time (GMT). To calculate the length of time that statistics have been collected, subtract the value of this member from the present time.

sts0_fopens: Specifies a DWORD value that indicates the number of files that have been opened on a server. This MUST include the number of times named pipes are opened.

sts0_devopens: Specifies a DWORD value that indicates the number of times a server device has been opened.

sts0_jobsqueued: Specifies a DWORD value that indicates the number of server print jobs spooled.

sts0_sopens: Specifies a DWORD value that indicates the number of sessions that have been established to a server.

sts0_stimedout: Specifies a DWORD value that indicates the number of times a session automatically disconnected.

sts0_serrorout: Specifies a DWORD value that indicates the number of times a session failed with an error.

sts0_pwerrors: Specifies a DWORD value that indicates the number of password violations that the server has detected.

sts0_permerrors: Specifies a DWORD value that indicates the number of access permission errors that have occurred on the server.

sts0_syserrors: Specifies a DWORD value that indicates the number of system errors that have occurred on the server.

sts0_bytessent_low: Specifies the low-order DWORD of the number of server bytes sent on the network.

sts0_bytessent_high: Specifies the high-order DWORD of the number of server bytes sent on the network.

sts0_bytesrcvd_low: Specifies the low-order DWORD of the number of server bytes received from the network.

sts0_bytesrcvd_high: Specifies the high-order DWORD of the number of server bytes received from the network.

sts0_avresponse: Specifies a DWORD value that indicates the average server response time (in milliseconds).

sts0_reqbufneed: Specifies a DWORD value that indicates the number of times the server required a request buffer but failed to allocate one.

sts0_bigbufneed: Specifies a DWORD value that indicates the number of times the server required a large buffer but failed to allocate one.

2.2.4.38 SERVER_INFO_100

The **SERVER_INFO_100** structure contains information about the specified server, including the name and platform. It MUST be used only to query information about a server.

```
typedef struct _SERVER_INFO_100 {
  DWORD sv100_platform_id;
  [string] wchar_t* sv100_name;
} SERVER_INFO_100,
 *PSERVER_INFO_100,
 *LPSERVER_INFO_100;
```

sv100_platform_id: Specifies the information level to use for platform-specific information. This member MUST be one of the values that are listed in section 2.2.2.6.

sv100_name: A pointer to a null-terminated Unicode UTF-16 string specifying the DNS name (as specified in [RFC1123] section 2.1) or NetBIOS name (as specified in [RFC1001] section 14 and [RFC1002] section 4) of a server.

2.2.4.39 SERVER_INFO_101

The **SERVER_INFO_101** structure contains information about the specified server, including name, platform, type of server, and associated software. For a description about the fields, see the description for **SERVER_INFO_102**. sv101_xxx denotes the same information as sv102_xxx.

```
typedef struct _SERVER_INFO_101 {
  DWORD sv101_platform_id;
  [string] wchar_t* sv101_name;
  DWORD sv101_version_major;
  DWORD sv101_version_minor;
  DWORD sv101_type;
  [string] wchar_t* sv101_comment;
} SERVER_INFO_101,
 *PSERVER_INFO_101,
 *LPSERVER_INFO_101;
```

2.2.4.40 SERVER_INFO_102

The **SERVER_INFO_102** structure contains information about the specified server, including the name, platform, and type of server, attributes, and associated software.

```
typedef struct _SERVER_INFO_102 {
  DWORD sv102_platform_id;
  [string] wchar_t* sv102_name;
  DWORD sv102_version_major;
  DWORD sv102_version_minor;
  DWORD sv102_type;
  [string] wchar_t* sv102_comment;
  DWORD sv102_users;
```

```
  long sv102_disc;
  int sv102_hidden;
  DWORD sv102_announce;
  DWORD sv102_anndelta;
  DWORD sv102_licenses;
  [string] wchar_t* sv102_userpath;
} SERVER_INFO_102,
 *PSERVER_INFO_102,
 *LPSERVER_INFO_102;
```

sv102_platform_id: Specifies the information level to use for platform-specific information. This member can be one of the values that are listed in PLATFORM IDs.<10>

sv102_name: A pointer to a null-terminated Unicode UTF-16 string specifying the DNS name (as specified in [RFC1123] section 2.1) or NetBIOS name (as specified in [RFC1001] section 14 and [RFC1002] section 4) of a server.<11>

sv102_version_major: Specifies the major release version number of the operating system.<12>

sv102_version_minor: Specifies the minor release version number of the operating system.<13>

sv102_type: Specifies the type of software the computer is running. This member MUST be a combination of one or more of the values that are listed in section 2.2.2.7.<14>

sv102_comment: Optional pointer to a null-terminated Unicode UTF-16 string specifying a comment that describes the server.

sv102_users: Specifies the number of users who can attempt to log on to the server.<15>

sv102_disc: Specifies the automatic disconnect time, in minutes. A session MUST be disconnected if it is idle longer than the period of time that the **sv102_disc** member specifies. If the value of **sv102_disc** is SV_NODISC (0xFFFFFFFF), automatic disconnect MUST NOT be enabled.<16>

sv102_hidden: A Boolean that specifies whether the server is hidden or visible to other computers in the same network domain. MUST be set to TRUE (1) to indicate that the server is hidden; MUST be set to FALSE (0) to indicate that the server is visible.<17>

sv102_announce: Specifies the network announce rate, in seconds. This rate determines how often the server is announced to other computers on the network for discovery by using the Microsoft CIFS Browser Protocol. For more information, see [MS-BRWS].<18>

sv102_anndelta: Specifies the delta value for the announce rate, in milliseconds. This value specifies how much the announce rate can vary from the period of time that is specified in the **sv102_announce** member. The delta value allows randomly varied announce rates.<19>

sv102_licenses: Unused. MUST be set to 0 and ignored on receipt.<20>

sv102_userpath: A pointer to a null-terminated Unicode UTF-16 string specifying the path to user directories.

2.2.4.41 SERVER_INFO_502

The **SERVER_INFO_502** structure contains information about a specified server. For a description of the fields, see the description for **SERVER_INFO_599**. sv502_xxx denotes the same information as sv599_xxx.

```
typedef struct _SERVER_INFO_502 {
  DWORD sv502_sessopens;
  DWORD sv502_sessvcs;
  DWORD sv502_opensearch;
  DWORD sv502_sizreqbuf;
  DWORD sv502_initworkitems;
  DWORD sv502_maxworkitems;
  DWORD sv502_rawworkitems;
  DWORD sv502_irpstacksize;
  DWORD sv502_maxrawbuflen;
  DWORD sv502_sessusers;
  DWORD sv502_sessconns;
  DWORD sv502_maxpagedmemoryusage;
  DWORD sv502_maxnonpagedmemoryusage;
  int sv502_enablesoftcompat;
  int sv502_enableforcedlogoff;
  int sv502_timesource;
  int sv502_acceptdownlevelapis;
  int sv502_lmannounce;
} SERVER_INFO_502,
 *PSERVER_INFO_502,
 *LPSERVER_INFO_502;
```

2.2.4.42 SERVER_INFO_503

The **SERVER_INFO_503** structure contains information about a specified server. For a description of the fields, see the description for **SERVER_INFO_599**. sv503_xxx denotes the same information as sv599_xxx.

```
typedef struct _SERVER_INFO_503 {
  DWORD sv503_sessopens;
  DWORD sv503_sessvcs;
  DWORD sv503_opensearch;
  DWORD sv503_sizreqbuf;
  DWORD sv503_initworkitems;
  DWORD sv503_maxworkitems;
  DWORD sv503_rawworkitems;
  DWORD sv503_irpstacksize;
  DWORD sv503_maxrawbuflen;
  DWORD sv503_sessusers;
  DWORD sv503_sessconns;
  DWORD sv503_maxpagedmemoryusage;
  DWORD sv503_maxnonpagedmemoryusage;
  int sv503_enablesoftcompat;
  int sv503_enableforcedlogoff;
  int sv503_timesource;
  int sv503_acceptdownlevelapis;
  int sv503_lmannounce;
  [string] wchar_t* sv503_domain;
```

```
  DWORD sv503_maxcopyreadlen;
  DWORD sv503_maxcopywritelen;
  DWORD sv503_minkeepsearch;
  DWORD sv503_maxkeepsearch;
  DWORD sv503_minkeepcomplsearch;
  DWORD sv503_maxkeepcomplsearch;
  DWORD sv503_threadcountadd;
  DWORD sv503_numblockthreads;
  DWORD sv503_scavtimeout;
  DWORD sv503_minrcvqueue;
  DWORD sv503_minfreeworkitems;
  DWORD sv503_xactmemsize;
  DWORD sv503_threadpriority;
  DWORD sv503_maxmpxct;
  DWORD sv503_oplockbreakwait;
  DWORD sv503_oplockbreakresponsewait;
  int sv503_enableoplocks;
  int sv503_enableoplockforceclose;
  int sv503_enablefcbopens;
  int sv503_enableraw;
  int sv503_enablesharednetdrives;
  DWORD sv503_minfreeconnections;
  DWORD sv503_maxfreeconnections;
} SERVER_INFO_503,
 *PSERVER_INFO_503,
 *LPSERVER_INFO_503;
```

2.2.4.43 SERVER_INFO_599

The **SERVER_INFO_599** structure contains information about a specified server. These fields involve implementation-specific details of an SMB file server and may vary in how they apply to any given implementation. For more information, see section 3.1.4.18.

```
typedef struct _SERVER_INFO_599 {
  DWORD sv599_sessopens;
  DWORD sv599_sessvcs;
  DWORD sv599_opensearch;
  DWORD sv599_sizreqbuf;
  DWORD sv599_initworkitems;
  DWORD sv599_maxworkitems;
  DWORD sv599_rawworkitems;
  DWORD sv599_irpstacksize;
  DWORD sv599_maxrawbuflen;
  DWORD sv599_sessusers;
  DWORD sv599_sessconns;
  DWORD sv599_maxpagedmemoryusage;
  DWORD sv599_maxnonpagedmemoryusage;
  int sv599_enablesoftcompat;
  int sv599_enableforcedlogoff;
  int sv599_timesource;
  int sv599_acceptdownlevelapis;
  int sv599_lmannounce;
  [string] wchar_t* sv599_domain;
  DWORD sv599_maxcopyreadlen;
  DWORD sv599_maxcopywritelen;
```

```
  DWORD sv599_minkeepsearch;
  DWORD sv599_maxkeepsearch;
  DWORD sv599_minkeepcomplsearch;
  DWORD sv599_maxkeepcomplsearch;
  DWORD sv599_threadcountadd;
  DWORD sv599_numblockthreads;
  DWORD sv599_scavtimeout;
  DWORD sv599_minrcvqueue;
  DWORD sv599_minfreeworkitems;
  DWORD sv599_xactmemsize;
  DWORD sv599_threadpriority;
  DWORD sv599_maxmpxct;
  DWORD sv599_oplockbreakwait;
  DWORD sv599_oplockbreakresponsewait;
  int sv599_enableoplocks;
  int sv599_enableoplockforceclose;
  int sv599_enablefcbopens;
  int sv599_enableraw;
  int sv599_enablesharednetdrives;
  DWORD sv599_minfreeconnections;
  DWORD sv599_maxfreeconnections;
  DWORD sv599_initsesstable;
  DWORD sv599_initconntable;
  DWORD sv599_initfiletable;
  DWORD sv599_initsearchtable;
  DWORD sv599_alertschedule;
  DWORD sv599_errorthreshold;
  DWORD sv599_networkerrorthreshold;
  DWORD sv599_diskspacethreshold;
  DWORD sv599_reserved;
  DWORD sv599_maxlinkdelay;
  DWORD sv599_minlinkthroughput;
  DWORD sv599_linkinfovalidtime;
  DWORD sv599_scavqosinfoupdatetime;
  DWORD sv599_maxworkitemidletime;
} SERVER_INFO_599,
 *PSERVER_INFO_599,
 *LPSERVER_INFO_599;
```

sv599_sessopens: Specifies the number of files that can be open in one session.<21>

sv599_sessvcs: Specifies the maximum number of sessions permitted per client.<22>

sv599_opensearch: Specifies the number of search operations that can be carried out simultaneously.<23>

sv599_sizreqbuf: Specifies the size, in bytes, of each server buffer.<24>

sv599_initworkitems: Specifies the initial number of receive buffers, or work items, that the server uses.<25>

sv599_maxworkitems: Specifies the maximum number of receive buffers, or work items, the server can allocate. If this limit is reached, the transport MUST initiate flow control.<26>

sv599_rawworkitems: Specifies the number of special work items the server uses for raw mode I/O. A larger value for this member can increase performance, but it requires more memory.<27>

sv599_irpstacksize: Specifies the number of stack locations that the server allocated in I/O request packets (IRPs).<28>

sv599_maxrawbuflen: The server MAY validate the value on receipt to ensure it is within the allowed range.<29>

sv599_sessusers: Specifies the maximum number of users that can be logged on to the server using a single session.<30>

sv599_sessconns: Specifies the maximum number of tree connections that can be made on the server using a single session.<31>

sv599_maxpagedmemoryusage: Specifies the maximum size of pageable memory in bytes that the server can allocate at any one time.<32>

sv599_maxnonpagedmemoryusage: Specifies the maximum size of nonpaged memory in bytes that the server can allocate at any one time.<33>

sv599_enablesoftcompat: A Boolean that specifies the SoftCompatibility capability of the SMB File Server. This field MUST be set to TRUE (1) to enable the SoftCompatibility feature, and MUST be set to FALSE (0) to disable the SoftCompatibility feature.<34>

sv599_enableforcedlogoff: A Boolean that specifies whether the server MUST force a client to disconnect, even if the client has open files, after the client's logon time has expired. This field MUST be set to TRUE (1) for the server to force a client to disconnect under those circumstances, and MUST be set to FALSE (0) if a client disconnect MUST NOT be performed under those circumstances.<35>

sv599_timesource: A Boolean that specifies whether the server is a reliable time source.

sv599_acceptdownlevelapis: A Boolean that specifies whether the server accepts method calls from previous-generation LAN Manager clients. This field MUST be set to TRUE (1) to enable the server to accept method calls from previous-generation LAN Manager clients, and MUST be set to FALSE (0) to disable the server from accepting method calls from previous LAN Manager clients.<36>

sv599_lmannounce: A Boolean that specifies whether the server is visible to LAN Manager 2.x clients.<37>

sv599_domain: A pointer to a Unicode UTF character string that specifies the name of the server's domain. This field cannot be modified by clients.

sv599_maxcopyreadlen: The server MAY validate the value on receipt to ensure it is within the allowed range.<38>

sv599_maxcopywritelen: The server MAY validate the value on receipt to ensure it is within the allowed range.<39>

sv599_minkeepsearch: The server MAY validate the value on receipt to ensure it is within the allowed range.<40>

sv599_maxkeepsearch: Specifies the length of time, in seconds, that the SMB file server retains information about incomplete search operations.<41>

sv599_minkeepcomplsearch: The server MAY validate the value on receipt to ensure it is within the allowed range.<42>

sv599_maxkeepcomplsearch: The server MAY validate the value on receipt to ensure it is within the allowed range.<43>

sv599_threadcountadd: Unused. This field MUST be ignored on receipt.

sv599_numblockthreads: Unused. This field MUST be ignored on receipt.

sv599_scavtimeout: Specifies the period of time in seconds that the scavenger thread on the SMB file server remains idle before waking up to service requests.<44><45>

sv599_minrcvqueue: Specifies the minimum number of free receive work items the server requires before it begins to allocate more.<46>

sv599_minfreeworkitems: Specifies the minimum number of available receive work items that the server requires to begin processing a server message block.<47>

sv599_xactmemsize: Specifies the size, in bytes, of the shared memory region used to process server methods.<48>

sv599_threadpriority: Specifies the priority of all server threads in relation to the base priority of the process.<49>

sv599_maxmpxct: Specifies the maximum number of outstanding requests that any one client can send to the server.<50>

sv599_oplockbreakwait: Specifies the period of time, in seconds, to wait before timing out an opportunistic lock break request.<51>

sv599_oplockbreakresponsewait: Specifies the period of time, in seconds, that the server waits for a client to respond to an opportunistic lock break request from the server.<52>

sv599_enableoplocks: A Boolean that specifies whether the server allows clients to use opportunistic locks on files. Opportunistic locks are a significant performance enhancement, but they have the potential to cause lost cached data on some networks, particularly wide-area networks. This field MUST be set to TRUE (1) to enable clients to use opportunistic locks on files, and MUST be set to FALSE (0) to restrict clients from using opportunistic locks on files.<53>

sv599_enableoplockforceclose: Unused. MUST be set to 0 and ignored on receipt.

sv599_enablefcbopens: Specifies whether several MS-DOS File Control Blocks (FCBs) are placed in a single location accessible to the server. If enabled, this option can save resources on the server. This field MUST be set to TRUE (1) to place multiple MS-DOS FCBs in a single location accessible to the server, and MUST be set to FALSE (0) otherwise.<54>

sv599_enableraw: Specifies whether the server processes raw SMBs. If enabled, this allows more data to transfer per transaction and improves performance. However, it is possible that processing raw SMBs can impede performance on certain networks. This field MUST be set to TRUE (1) to indicate that the server processes raw SMBs, and MUST be set to FALSE (0) to indicate that the server does not process raw SMBs. The server MUST maintain the value of this member.<55>

sv599_enablesharednetdrives: Specifies whether the server allows redirected server drives to be shared.<56>

sv599_minfreeconnections: Specifies the minimum number of free connection blocks that are maintained per endpoint. The server MUST set these aside to handle bursts of requests by clients to connect to the server.<57>

sv599_maxfreeconnections: Specifies the maximum number of free connection blocks that are maintained per endpoint. The server MUST set these aside to handle bursts of requests by clients to connect to the server.<58>

sv599_initsesstable: Specifies the initial session table size for the server.<59>

sv599_initconntable: Specifies the initial connection table size for the server.<60>

sv599_initfiletable: Specifies the initial file table size for the server.<61>

sv599_initsearchtable: Specifies the initial search table size for the server.<62>

sv599_alertschedule: Specifies the time, in minutes, between two invocations of the alerter algorithm on the Microsoft SMB File Server.<63><64>

sv599_errorthreshold: Specifies the number of failed operations (non-network) the SMB file server needs to see before raising an administrative alert.<65>

sv599_networkerrorthreshold: Specifies the percentage of failed network operations the SMB file server needs to see before raising an administrative alert.<66>

sv599_diskspacethreshold: Specifies the percent free disk at which to raise an administrative alert.<67>

sv599_reserved: Reserved. MUST be 0.

sv599_maxlinkdelay: Specifies the maximum link delay, in seconds, for the SMB file server.<68>

sv599_minlinkthroughput: Specifies the minimum link throughput, in bytes/sec, for the SMB file server.<69>

sv599_linkinfovalidtime: Specifies the time interval, in seconds, during which the SMB file server can use the computed link information before having to compute it again.<70>

sv599_scavqosinfoupdatetime: Specifies the time interval after which the SMB file server scavenger thread has to update quality of service (QoS) information.<71>

sv599_maxworkitemidletime: Specifies the maximum work item idle time in seconds.<72>

2.2.4.44 SERVER_INFO_1005

The **SERVER_INFO_1005** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1005 {
  [string] wchar_t* sv1005_comment;
} SERVER_INFO_1005,
 *PSERVER_INFO_1005,
 *LPSERVER_INFO_1005;
```

sv1005_comment: Optional pointer to a null-terminated Unicode UTF-16 string specifying a comment that describes the server.<73>

2.2.4.45 SERVER_INFO_1107

The **SERVER_INFO_1107** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1107 {
  DWORD sv1107_users;
} SERVER_INFO_1107,
 *PSERVER_INFO_1107,
 *LPSERVER_INFO_1107;
```

sv1107_users: Specifies the number of users who can attempt to log on to the server.<74>

2.2.4.46 SERVER_INFO_1010

The **SERVER_INFO_1010** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1010 {
  long sv1010_disc;
} SERVER_INFO_1010,
 *PSERVER_INFO_1010,
 *LPSERVER_INFO_1010;
```

sv1010_disc: Specifies the automatic disconnect time, in minutes. A session MUST be disconnected if it is idle longer than the period of time that the **sv1010_disc** member specifies. If the value of **sv1010_disc** is SV_NODISC (0xFFFFFFFF), automatic disconnect MUST NOT be not enabled.<75>

2.2.4.47 SERVER_INFO_1016

The **SERVER_INFO_1016** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1016 {
  int sv1016_hidden;
} SERVER_INFO_1016,
 *PSERVER_INFO_1016,
 *LPSERVER_INFO_1016;
```

sv1016_hidden: Boolean that specifies whether the server is hidden or visible to other computers in the same network domain. Set TRUE (1) to indicate that the server is hidden; set FALSE (0) to indicate that the server is visible.<76>

2.2.4.48 SERVER_INFO_1017

The **SERVER_INFO_1017** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1017 {
  DWORD sv1017_announce;
} SERVER_INFO_1017,
 *PSERVER_INFO_1017,
 *LPSERVER_INFO_1017;
```

sv1017_announce: Specifies the network announce rate, in seconds. This rate determines how often the server is announced to other computers on the network for discovery by using the Microsoft CIFS Browser Protocol. For more information, see [MS-BRWS].<77>

2.2.4.49 SERVER_INFO_1018

The **SERVER_INFO_1018** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1018 {
  DWORD sv1018_anndelta;
} SERVER_INFO_1018,
 *PSERVER_INFO_1018,
 *LPSERVER_INFO_1018;
```

sv1018_anndelta: Specifies the delta value for the announce rate, in milliseconds. This value specifies how much the announce rate MAY vary from the period of time that is specified in the **sv1018_announce** member. The delta value enables the server to set randomly varied announce rates in the range **sv1018_announce** to **sv1018_announce**+**sv1018_anndelta**, inclusive, to prevent many servers from announcing at the same time.<78>

2.2.4.50 SERVER_INFO_1501

The **SERVER_INFO_1501** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1501 {
  DWORD sv1501_sessopens;
} SERVER_INFO_1501,
 *PSERVER_INFO_1501,
 *LPSERVER_INFO_1501;
```

sv1501_sessopens: Specifies the number of files that can be open in one session.<79>

2.2.4.51 SERVER_INFO_1502

The **SERVER_INFO_1502** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1502 {
  DWORD sv1502_sessvcs;
} SERVER_INFO_1502,
 *PSERVER_INFO_1502,
 *LPSERVER_INFO_1502;
```

sv1502_sessvcs: Specifies the maximum number of sessions that are permitted per client.<80>

2.2.4.52 SERVER_INFO_1503

The **SERVER_INFO_1503** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1503 {
  DWORD sv1503_opensearch;
} SERVER_INFO_1503,
 *PSERVER_INFO_1503,
 *LPSERVER_INFO_1503;
```

sv1503_opensearch: Specifies the maximum number of entries in the internal table that is used to hold information about outstanding search operations.<81>

2.2.4.53 SERVER_INFO_1506

The **SERVER_INFO_1506** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1506 {
  DWORD sv1506_maxworkitems;
} SERVER_INFO_1506,
 *PSERVER_INFO_1506,
 *LPSERVER_INFO_1506;
```

sv1506_maxworkitems: Specifies the maximum number of receive buffers, or work items, that the server can allocate. If this limit is reached, the transport MUST initiate flow control, which could incur performance cost.<82>

2.2.4.54 SERVER_INFO_1510

The **SERVER_INFO_1510** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1510 {
  DWORD sv1510_sessusers;
} SERVER_INFO_1510,
 *PSERVER_INFO_1510,
 *LPSERVER_INFO_1510;
```

sv1510_sessusers: Specifies the maximum number of users that can be logged on to the server using a single session.<83>

2.2.4.55 SERVER_INFO_1511

The **SERVER_INFO_1511** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1511 {
  DWORD sv1511_sessconns;
```

```
} SERVER_INFO_1511,
 *PSERVER_INFO_1511,
 *LPSERVER_INFO_1511;
```

sv1511_sessconns: Specifies the maximum number of tree connections that can be made on the server by using a single session.<84>

2.2.4.56 SERVER_INFO_1512

The **SERVER_INFO_1512** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1512 {
  DWORD sv1512_maxnonpagedmemoryusage;
} SERVER_INFO_1512,
 *PSERVER_INFO_1512,
 *LPSERVER_INFO_1512;
```

sv1512_maxnonpagedmemoryusage: Specifies the maximum number of bytes of non-pageable memory that the server can allocate at any one time.<85>

2.2.4.57 SERVER_INFO_1513

The **SERVER_INFO_1513** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1513 {
  DWORD sv1513_maxpagedmemoryusage;
} SERVER_INFO_1513,
 *PSERVER_INFO_1513,
 *LPSERVER_INFO_1513;
```

sv1513_maxpagedmemoryusage: Specifies the maximum number of bytes of pageable memory that the server can allocate at any one time.<86>

2.2.4.58 SERVER_INFO_1514

The **SERVER_INFO_1514** structure contains information about a specified server.

```
typedef struct SERVER_INFO_1514 {
  int sv1514_enablesoftcompat;
} SERVER_INFO_1514,
 *PSERVER_INFO_1514,
 *LPSERVER_INFO_1514;
```

sv1514_enablesoftcompat: Boolean that specifies the SoftCompatibility capability of the Windows SMB File Server. This member affects open mode when the client does not have read/write permission to the file that it is accessing. If this feature is enabled, the server MUST use FILE_SHARE_READ as the share access parameter to CreateFile and MUST NOT

mark the open as compatibility mode open. Otherwise, the server MUST set the share access parameter to CreateFile to 0, and MUST mark the open as compatibility mode open.<87>

2.2.4.59 SERVER_INFO_1515

The **SERVER_INFO_1515** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1515 {
  int sv1515_enableforcedlogoff;
} SERVER_INFO_1515,
 *PSERVER_INFO_1515,
 *LPSERVER_INFO_1515;
```

sv1515_enableforcedlogoff: A Boolean that specifies whether the server MUST force a client to disconnect, even if the client has open files, after the client's logon time has expired.<88>

2.2.4.60 SERVER_INFO_1516

The **SERVER_INFO_1516** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1516 {
  int sv1516_timesource;
} SERVER_INFO_1516,
 *PSERVER_INFO_1516,
 *LPSERVER_INFO_1516;
```

sv1516_timesource: A Boolean that specifies whether the server is a reliable time source.

2.2.4.61 SERVER_INFO_1518

The **SERVER_INFO_1518** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1518 {
  int sv1518_lmannounce;
} SERVER_INFO_1518,
 *PSERVER_INFO_1518,
 *LPSERVER_INFO_1518;
```

sv1518_lmannounce: A Boolean that specifies whether the server is visible to LAN Manager 2.x clients.<89>

2.2.4.62 SERVER_INFO_1523

The **SERVER_INFO_1523** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1523 {
  DWORD sv1523_maxkeepsearch;
} SERVER_INFO_1523,
 *PSERVER_INFO_1523,
```

```
 *LPSERVER_INFO_1523;
```

sv1523_maxkeepsearch: Specifies the time, in seconds, that the SMB file server retains information about incomplete search operations.<90>

2.2.4.63 SERVER_INFO_1528

The **SERVER_INFO_1528** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1528 {
  DWORD sv1528_scavtimeout;
} SERVER_INFO_1528,
 *PSERVER_INFO_1528,
 *LPSERVER_INFO_1528;
```

sv1528_scavtimeout: Specifies the time, in seconds, that the SMB file server waits between attempts to scavenge internal resources.<91>

2.2.4.64 SERVER_INFO_1529

The **SERVER_INFO_1529** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1529 {
  DWORD sv1529_minrcvqueue;
} SERVER_INFO_1529,
 *PSERVER_INFO_1529,
 *LPSERVER_INFO_1529;
```

sv1529_minrcvqueue: Specifies the minimum number of free receive work items that the server requires before it begins to allocate more.<92>

2.2.4.65 SERVER_INFO_1530

The **SERVER_INFO_1530** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1530 {
  DWORD sv1530_minfreeworkitems;
} SERVER_INFO_1530,
 *PSERVER_INFO_1530,
 *LPSERVER_INFO_1530;
```

sv1530_minfreeworkitems: Specifies the minimum number of available receive work items that the server requires to begin processing an SMB.<93>

2.2.4.66 SERVER_INFO_1533

The **SERVER_INFO_1533** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1533 {
  DWORD sv1533_maxmpxct;
} SERVER_INFO_1533,
 *PSERVER_INFO_1533,
 *LPSERVER_INFO_1533;
```

sv1533_maxmpxct: Specifies the maximum number of outstanding requests that any one client can send to the server.<94>

2.2.4.67 SERVER_INFO_1534

The **SERVER_INFO_1534** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1534 {
  DWORD sv1534_oplockbreakwait;
} SERVER_INFO_1534,
 *PSERVER_INFO_1534,
 *LPSERVER_INFO_1534;
```

sv1534_oplockbreakwait: Specifies the time, in seconds, to wait before timing out an opportunistic lock break request.<95>

2.2.4.68 SERVER_INFO_1535

The **SERVER_INFO_1535** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1535 {
  DWORD sv1535_oplockbreakresponsewait;
} SERVER_INFO_1535,
 *PSERVER_INFO_1535,
 *LPSERVER_INFO_1535;
```

sv1535_oplockbreakresponsewait: Specifies the time, in seconds, that the server waits for a client to respond to an opportunistic lock break request from the server.<96>

2.2.4.69 SERVER_INFO_1536

The **SERVER_INFO_1536** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1536 {
  int sv1536_enableoplocks;
} SERVER_INFO_1536,
 *PSERVER_INFO_1536,
 *LPSERVER_INFO_1536;
```

sv1536_enableoplocks: A Boolean that specifies whether the server allows clients to use opportunistic locks on files. Opportunistic locks are a significant performance enhancement,

but they have the potential to cause lost cached data on some networks, particularly wide-area networks.<97>

2.2.4.70 SERVER_INFO_1538

The **SERVER_INFO_1538** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1538 {
  int sv1538_enablefcbopens;
} SERVER_INFO_1538,
 *PSERVER_INFO_1538,
 *LPSERVER_INFO_1538;
```

sv1538_enablefcbopens: A Boolean that specifies whether several MS-DOS File Control Blocks (FCBs) are placed in a single location that is accessible to the server. If enabled, this option can save resources on the server.<98>

2.2.4.71 SERVER_INFO_1539

The **SERVER_INFO_1539** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1539 {
  int sv1539_enableraw;
} SERVER_INFO_1539,
 *PSERVER_INFO_1539,
 *LPSERVER_INFO_1539;
```

sv1539_enableraw: Specifies whether the server processes raw SMBs. If enabled, this option allows more data to transfer per transaction and improves performance. However, it is possible that processing raw SMBs can impede performance on certain networks. The server MUST maintain the value of this member.<99>

2.2.4.72 SERVER_INFO_1540

The **SERVER_INFO_1540** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1540 {
  int sv1540_enablesharednetdrives;
} SERVER_INFO_1540,
 *PSERVER_INFO_1540,
 *LPSERVER_INFO_1540;
```

sv1540_enablesharednetdrives: A Boolean that specifies whether the server allows redirected server drives to be shared.<100>

2.2.4.73 SERVER_INFO_1541

The **SERVER_INFO_1541** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1541 {
  int sv1541_minfreeconnections;
} SERVER_INFO_1541,
 *PSERVER_INFO_1541,
 *LPSERVER_INFO_1541;
```

sv1541_minfreeconnections: Specifies the minimum number of free connection blocks that are maintained per endpoint. The server sets these aside to handle bursts of requests by clients to connect to the server.<101>

2.2.4.74 SERVER_INFO_1542

The **SERVER_INFO_1542** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1542 {
  int sv1542_maxfreeconnections;
} SERVER_INFO_1542,
 *PSERVER_INFO_1542,
 *LPSERVER_INFO_1542;
```

sv1542_maxfreeconnections: Specifies the maximum number of free connection blocks that are maintained per endpoint. The server sets these aside to handle bursts of requests by clients to connect to the server.<102>

2.2.4.75 SERVER_INFO_1543

The **SERVER_INFO_1543** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1543 {
  DWORD sv1543_initsesstable;
} SERVER_INFO_1543,
 *PSERVER_INFO_1543,
 *LPSERVER_INFO_1543;
```

sv1543_initsesstable: Specifies the initial session table size for the server in terms of the number of records (array elements).<103>

2.2.4.76 SERVER_INFO_1544

The **SERVER_INFO_1544** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1544 {
  DWORD sv1544_initconntable;
} SERVER_INFO_1544,
 *PSERVER_INFO_1544,
 *LPSERVER_INFO_1544;
```

sv1544_initconntable: Specifies the initial connection table size for the server in terms of the number of records (array elements).<104>

2.2.4.77 SERVER_INFO_1545

The **SERVER_INFO_1545** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1545 {
  DWORD sv1545_initfiletable;
} SERVER_INFO_1545,
 *PSERVER_INFO_1545,
 *LPSERVER_INFO_1545;
```

sv1545_initfiletable: Specifies the initial file table size for the server in terms of the number of records (array elements).<105>

2.2.4.78 SERVER_INFO_1546

The **SERVER_INFO_1546** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1546 {
  DWORD sv1546_initsearchtable;
} SERVER_INFO_1546,
 *PSERVER_INFO_1546,
 *LPSERVER_INFO_1546;
```

sv1546_initsearchtable: Specifies the initial search table size for the server in terms of the number of records (array elements).<106>

2.2.4.79 SERVER_INFO_1547

The **SERVER_INFO_1547** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1547 {
  DWORD sv1547_alertschedule;
} SERVER_INFO_1547,
 *PSERVER_INFO_1547,
 *LPSERVER_INFO_1547;
```

sv1547_alertschedule: Specifies the time, in minutes, between two invocations of the alerter algorithm on the Microsoft SMB File Server.<107>

2.2.4.80 SERVER_INFO_1548

The **SERVER_INFO_1548** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1548 {
  DWORD sv1548_errorthreshold;
} SERVER_INFO_1548,
```

```
 *PSERVER_INFO_1548,
 *LPSERVER_INFO_1548;
```

sv1548_errorthreshold: Specifies the minimum number of failed non-network operations that are seen by the SMB file server before it raises an administrative alert.<108>

2.2.4.81 SERVER_INFO_1549

The **SERVER_INFO_1549** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1549 {
  DWORD sv1549_networkerrorthreshold;
} SERVER_INFO_1549,
 *PSERVER_INFO_1549,
 *LPSERVER_INFO_1549;
```

sv1549_networkerrorthreshold: Specifies the minimum percentage of failed network operations the SMB file server needs to see before raising an administrative alert. An alert MUST be raised when (the number of failed network operations / the number of all attempted network operations) * 100 is greater than or equal to this value.<109>

2.2.4.82 SERVER_INFO_1550

The **SERVER_INFO_1550** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1550 {
  DWORD sv1550_diskspacethreshold;
} SERVER_INFO_1550,
 *PSERVER_INFO_1550,
 *LPSERVER_INFO_1550;
```

sv1550_diskspacethreshold: Specifies the minimum percentage of free disk space needed before the server MUST raise an administrative alert.<110>

2.2.4.83 SERVER_INFO_1552

The **SERVER_INFO_1552** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1552 {
  DWORD sv1552_maxlinkdelay;
} SERVER_INFO_1552,
 *PSERVER_INFO_1552,
 *LPSERVER_INFO_1552;
```

sv1552_maxlinkdelay: Specifies the *maxlinkdelay* parameter, in seconds, for the SMB file server.<111>

2.2.4.84 SERVER_INFO_1553

The **SERVER_INFO_1553** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1553 {
  DWORD sv1553_minlinkthroughput;
} SERVER_INFO_1553,
 *PSERVER_INFO_1553,
 *LPSERVER_INFO_1553;
```

sv1553_minlinkthroughput: Specifies the minimum link throughput parameter, in bytes per second, for the SMB file server.<112>

2.2.4.85 SERVER_INFO_1554

The **SERVER_INFO_1554** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1554 {
  DWORD sv1554_linkinfovalidtime;
} SERVER_INFO_1554,
 *PSERVER_INFO_1554,
 *LPSERVER_INFO_1554;
```

sv1554_linkinfovalidtime: Specifies the time interval, in seconds, during which the SMB file server SHOULD use the computed link information before having to compute it again.<113>

2.2.4.86 SERVER_INFO_1555

The **SERVER_INFO_1555** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1555 {
  DWORD sv1555_scavqosinfoupdatetime;
} SERVER_INFO_1555,
 *PSERVER_INFO_1555,
 *LPSERVER_INFO_1555;
```

sv1555_scavqosinfoupdatetime: Specifies the time interval (in seconds) after which the SMB file server scavenger thread MUST update QoS information.<114>

2.2.4.87 SERVER_INFO_1556

The **SERVER_INFO_1556** structure contains information about a specified server.

```
typedef struct _SERVER_INFO_1556 {
  DWORD sv1556_maxworkitemidletime;
} SERVER_INFO_1556,
 *PSERVER_INFO_1556,
 *LPSERVER_INFO_1556;
```

sv1556_maxworkitemidletime: Specifies the maximum work item idle time in seconds.<115>

2.2.4.88 DISK_INFO

The **DISK_INFO** structure contains information (the drive letter) about the disk device on the server.

```
typedef struct _DISK_INFO {
  [string] WCHAR Disk[3];
} DISK_INFO,
 *PDISK_INFO,
 *LPDISK_INFO;
```

Disk: The drive identifier of the disk device. This MUST consist of two Unicode UTF-16 characters followed by a NULL (for example, "A:\0"). The first character in this string MUST be a drive letter in the range 'A' through 'Z', inclusive. The second character MUST be the ':' character.

2.2.4.89 DISK_ENUM_CONTAINER

The **DISK_ENUM_CONTAINER** structure contains a value that indicates the number of entries that the **NetrServerDiskEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _DISK_ENUM_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead), length_is(EntriesRead)]
    LPDISK_INFO Buffer;
} DISK_ENUM_CONTAINER;
```

EntriesRead: Number of entries the method returns.

Buffer: A pointer to the **DISK_INFO** entries that the method returns.

2.2.4.90 SERVER_TRANSPORT_INFO_0

The **SERVER_TRANSPORT_INFO_0** structure contains information about the specified transport protocol, including the name, address, and location on the network. The definitions of fields in this structure are specified in section 2.2.4.93. Fields having names of the form svti0_xxx MUST be defined as in the corresponding SERVER_TRANSPORT_INFO_3 fields with names of the form svti3_xxx.

```
typedef struct _SERVER_TRANSPORT_INFO_0 {
  DWORD svti0_numberofvcs;
  [string] wchar_t* svti0_transportname;
  [size_is(svti0_transportaddresslength)]
    unsigned char* svti0_transportaddress;
  DWORD svti0_transportaddresslength;
  [string] wchar_t* svti0_networkaddress;
```

```
} SERVER_TRANSPORT_INFO_0,
 *PSERVER_TRANSPORT_INFO_0,
 *LPSERVER_TRANSPORT_INFO_0;
```

svti0_numberofvcs: Specifies a DWORD value that indicates the number of clients that are connected to the server and that are using the transport protocol that is specified by the **svti0_transportname** member.

svti0_transportname: Pointer to a null-terminated Unicode string that contains the implementation-specific name of a transport device that implements support for the transport, for example:

\Device\NetBT_Tcpip_{2C9725F4-151A-11D3-AEEC-C3B211BD350B}

svti0_transportaddress: Pointer to a variable that contains the transport address that the server is using on the transport device and that is specified by the **svti0_transportname** member.

This member is usually the NetBIOS name that the server is using. In these instances, the name MUST be 16 characters long, and the last character MUST be a blank character (0x20).

svti0_transportaddresslength: Specifies a DWORD value that contains the length, in bytes, of the **svti0_transportaddress** member. For NetBIOS names, the value of this member is 16 (decimal).

svti0_networkaddress: Pointer to a null-terminated character string that contains the address that the network adapter is using. The string is transport specific.

2.2.4.91 SERVER_TRANSPORT_INFO_1

The **SERVER_TRANSPORT_INFO_1** structure contains information about the specified transport protocol, including the name, address, and location on the network. The definitions of fields in this structure are specified in section 2.2.4.93. Fields having names of the form svti0_xxx MUST be defined as in the corresponding SERVER_TRANSPORT_INFO_3 fields with names of the form svti3_xxx.

```
typedef struct _SERVER_TRANSPORT_INFO_1 {
  DWORD svti1_numberofvcs;
  [string] wchar_t* svti1_transportname;
  [size_is(svti1_transportaddresslength)]
    unsigned char* svti1_transportaddress;
  DWORD svti1_transportaddresslength;
  [string] wchar_t* svti1_networkaddress;
  [string] wchar_t* svti1_domain;
} SERVER_TRANSPORT_INFO_1,
 *PSERVER_TRANSPORT_INFO_1,
 *LPSERVER_TRANSPORT_INFO_1;
```

svti1_numberofvcs: Specifies a DWORD value that indicates the number of clients that are connected to the server and that are using the transport protocol specified by the **svti1_transportname** member.

svti1_transportname: Pointer to a null-terminated Unicode string that contains the implementation-specific name of a device that implements support for the transport, for example:

\Device\NetBT_Tcpip_{2C9725F4-151A-11D3-AEEC-C3B211BD350B}

svti1_transportaddress: Pointer to a variable that contains the transport address the server is using on the transport device that is specified by the **svti1_transportname** member.

This member is usually the NetBIOS name that the server is using. In these instances, the name MUST be 16 characters long, and the last character MUST be a blank character (0x20).

svti1_transportaddresslength: Specifies a DWORD value that contains the length, in bytes, of the **svti1_transportaddress** member. For NetBIOS names, the value of this member is 16 (decimal).

svti1_networkaddress: Pointer to a null-terminated character string that contains the address that the network adapter is using. The string is transport-specific.

svti1_domain: Pointer to a null-terminated character string that contains the name of the domain to which the server MUST announce its presence.

2.2.4.92 SERVER_TRANSPORT_INFO_2

The **SERVER_TRANSPORT_INFO_2** structure contains information about the specified transport protocol, including the name and address. The definitions of fields in this structure are specified in section 2.2.4.93. Fields having names of the form svti0_xxx MUST be defined as in the corresponding SERVER_TRANSPORT_INFO_3 fields with names of the form svti3_xxx.

```
typedef struct _SERVER_TRANSPORT_INFO_2 {
  DWORD svti2_numberofvcs;
  [string] wchar_t* svti2_transportname;
  [size_is(svti2_transportaddresslength)]
    unsigned char* svti2_transportaddress;
  DWORD svti2_transportaddresslength;
  [string] wchar_t* svti2_networkaddress;
  [string] wchar_t* svti2_domain;
  unsigned long svti2_flags;
} SERVER_TRANSPORT_INFO_2,
 *PSERVER_TRANSPORT_INFO_2,
 *LPSERVER_TRANSPORT_INFO_2;
```

svti2_numberofvcs: Specifies a DWORD value that indicates the number of clients that are connected to the server and that are using the transport protocol specified by the **svti2_transportname** member.

svti2_transportname: Pointer to a null-terminated Unicode string that contains the implementation-specific name of a device that implements support for the transport, for example:

\Device\NetBT_Tcpip_{2C9725F4-151A-11D3-AEEC-C3B211BD350B}

svti2_transportaddress: Pointer to a variable that contains the transport address that the server is using on the transport device that is specified by the **svti2_transportname** member.

This member is usually the NetBIOS name that the server is using. In these instances, the name MUST be 16 characters long, and the last character MUST be a blank character (0x20).

svti2_transportaddresslength: Specifies a DWORD value that contains the length, in bytes, of the **svti2_transportaddress** member. For NetBIOS names, the value of this member is 16 (decimal).

svti2_networkaddress: Pointer to a null-terminated character string that contains the address that the network adapter is using. The string is transport-specific.

svti2_domain: Pointer to a null-terminated character string that contains the name of the domain to which the server should announce its presence.

svti2_flags: This member MUST be a combination of zero or more of the following values.

Value	Meaning
SVTI2_REMAP_PIPE_NAMES 0x00000002	If this value is set for an endpoint, then client requests that arrive over the transport to open a named pipe are rerouted (remapped) to the local pipe name $$\ServerName\PipeName.

2.2.4.93 SERVER_TRANSPORT_INFO_3

The **SERVER_TRANSPORT_INFO_3** structure contains information about the specified transport protocol, including the name, address and password (credentials).

```
typedef struct _SERVER_TRANSPORT_INFO_3 {
  DWORD svti3_numberofvcs;
  [string] wchar_t* svti3_transportname;
  [size_is(svti3_transportaddresslength)]
    unsigned char* svti3_transportaddress;
  DWORD svti3_transportaddresslength;
  [string] wchar_t* svti3_networkaddress;
  [string] wchar_t* svti3_domain;
  unsigned long svti3_flags;
  DWORD svti3_passwordlength;
  unsigned char svti3_password[256];
} SERVER_TRANSPORT_INFO_3,
 *PSERVER_TRANSPORT_INFO_3,
 *LPSERVER_TRANSPORT_INFO_3;
```

svti3_numberofvcs: Specifies a DWORD value that indicates the number of clients that are connected to the server and that are using the transport protocol that is specified by the **svti3_transportname** member.

svti3_transportname: A pointer to a null-terminated Unicode string that contains the implementation-specific name of a device that implements support for the transport. This field is provided by the transport driver and may depend on the physical network adapter over which the transport runs.<116>

\Device\NetBT_Tcpip_{2C9725F4-151A-11D3-AEEC-C3B211BD350B}

svti3_transportaddress: A pointer to a variable that contains the transport address that the server is using on the transport device that is specified by the **svti3_transportname** member. <117>

This member is usually the NetBIOS name that the server is using. In these instances, the name MUST be 16 characters long, and the last character MUST be a blank character (0x20).

svti3_transportaddresslength: Specifies a DWORD value that contains the length, in bytes, of the **svti3_transportaddress** member.<118>

svti3_networkaddress: A pointer to a null-terminated character string that contains the address that the network adapter is using. The string is transport-specific. The server MUST ignore this field on receipt.<119>

svti3_domain: A pointer to a null-terminated character string that contains the name of the domain to which the server MUST announce its presence.

svti3_flags: This member MUST be a combination of zero or more of the following values.

Value	Meaning
SVTI2_REMAP_PIPE_NAMES 0x00000002	If this value is set for an endpoint, client requests that arrive over the transport to open a named pipe MUST be rerouted (remapped) to the local pipe name $$\ServerName\PipeName.

svti3_passwordlength: Specifies a DWORD value that indicates the number of valid bytes in the **svti3_password** member.

svti3_password: Specifies the credentials to use for the new transport address. If the **svti3_passwordlength** member is zero, the credentials for the server MUST be used.

2.2.4.94 SERVER_XPORT_INFO_0_CONTAINER

The **SERVER_XPORT_INFO_0_CONTAINER** structure contains a value that indicates the number of entries that the **NetrServerTransportEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SERVER_XPORT_INFO_0_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSERVER_TRANSPORT_INFO_0 Buffer;
} SERVER_XPORT_INFO_0_CONTAINER,
 *PSERVER_XPORT_INFO_0_CONTAINER;
```

EntriesRead: Number of entries the method returns.

Buffer: A pointer to the **SERVER_TRANSPORT_INFO_0** entries that the method returns.

2.2.4.95 SERVER_XPORT_INFO_1_CONTAINER

The **SERVER_XPORT_INFO_1_CONTAINER** structure contains a value that indicates the number of entries that the **NetrServerTransportEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SERVER_XPORT_INFO_1_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSERVER_TRANSPORT_INFO_1 Buffer;
} SERVER_XPORT_INFO_1_CONTAINER,
 *PSERVER_XPORT_INFO_1_CONTAINER;
```

EntriesRead: Number of entries the method returns.

Buffer: A pointer to the **SERVER_TRANSPORT_INFO_1** entries that the method returns.

2.2.4.96 SERVER_XPORT_INFO_2_CONTAINER

The **SERVER_XPORT_INFO_2_CONTAINER** structure contains a value that indicates the number of entries that the **NetrServerTransportEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SERVER_XPORT_INFO_2_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSERVER_TRANSPORT_INFO_2 Buffer;
} SERVER_XPORT_INFO_2_CONTAINER,
 *PSERVER_XPORT_INFO_2_CONTAINER;
```

EntriesRead: Number of entries the method returns.

Buffer: A pointer to the **SERVER_TRANSPORT_INFO_2** entries that the method returns.

2.2.4.97 SERVER_XPORT_INFO_3_CONTAINER

The **SERVER_XPORT_INFO_3_CONTAINER** structure contains a value that indicates the number of entries that the **NetrServerTransportEnum** method returns and a pointer to the buffer that contains the entries.

```
typedef struct _SERVER_XPORT_INFO_3_CONTAINER {
  DWORD EntriesRead;
  [size_is(EntriesRead)] LPSERVER_TRANSPORT_INFO_3 Buffer;
} SERVER_XPORT_INFO_3_CONTAINER,
 *PSERVER_XPORT_INFO_3_CONTAINER;
```

EntriesRead: Number of entries the method returns.

Buffer: A pointer to the **SERVER_TRANSPORT_INFO_3** entries that the method returns.

2.2.4.98 SERVER_XPORT_ENUM_STRUCT

The **SERVER_XPORT_ENUM_STRUCT** structure specifies the information level that the client requests in the **NetrServerTransportEnum** method and encapsulates the **SERVER_XPORT_ENUM_UNION** union that receives the entries that are enumerated by the server.

```
typedef struct _SERVER_XPORT_ENUM_STRUCT {
  DWORD Level;
  [switch_is(Level)] SERVER_XPORT_ENUM_UNION XportInfo;
} SERVER_XPORT_ENUM_STRUCT,
 *PSERVER_XPORT_ENUM_STRUCT,
 *LPSERVER_XPORT_ENUM_STRUCT;
```

Level: Specifies the information level of the data. It MUST be one of the following values:

Value	Meaning
0	SERVER_XPORT_INFO_0_CONTAINER
1	SERVER_XPORT_INFO_1_CONTAINER
2	SERVER_XPORT_INFO_2_CONTAINER
3	SERVER_XPORT_INFO_3_CONTAINER

XportInfo: Contains information about file server transports in the format that is determined by **Level**, as shown in the table above. This member receives the enumerated information.

2.2.4.99 TIME_OF_DAY_INFO

The **TIME_OF_DAY_INFO** structure contains information about the time of day from a remote server.

```
typedef struct TIME_OF_DAY_INFO {
  DWORD tod_elapsedt;
  DWORD tod_msecs;
  DWORD tod_hours;
  DWORD tod_mins;
  DWORD tod_secs;
  DWORD tod_hunds;
  long tod_timezone;
  DWORD tod_tinterval;
  DWORD tod_day;
  DWORD tod_month;
  DWORD tod_year;
  DWORD tod_weekday;
} TIME_OF_DAY_INFO,
 *PTIME_OF_DAY_INFO,
 *LPTIME_OF_DAY_INFO;
```

tod_elapsedt: Specifies a DWORD value that MUST contain the number of seconds since 00:00:00, January 1, 1970, GMT.

tod_msecs: Specifies a DWORD value that contains the number of milliseconds from an arbitrary starting point (system reset).

tod_hours: Specifies a DWORD value that contains the current hour. This value MUST be in the range 0 through 23, inclusive.

tod_mins: Specifies a DWORD value that contains the current minute. This value MUST be in the range 0 through 59, inclusive.

tod_secs: Specifies a DWORD value that contains the current second. This value MUST be in the range 0 through 59, inclusive.

tod_hunds: Specifies a DWORD value that contains the current hundredth second (0.01 second). This value MUST be in the range 0 through 99, inclusive.

tod_timezone: Specifies the time zone of the server. This value MUST be calculated, in minutes, from Greenwich Mean Time (GMT). For time zones that are west of Greenwich, the value MUST be positive; for time zones that are east of Greenwich, the value MUST be negative. A value of –1 MUST indicate that the time zone is undefined

tod_tinterval: Specifies a DWORD value that contains the time interval for each tick of the clock. Each integral integer MUST represent one ten-thousandth second (0.0001 second).

tod_day: Specifies a DWORD value that contains the day of the month. This value MUST be in the range 1 through 31, inclusive.

tod_month: Specifies a DWORD value that contains the month of the year. This value MUST be in the range 1 through 12, inclusive.

tod_year: Specifies a DWORD value that contains the year.

tod_weekday: Specifies a DWORD value that contains the day of the week. This value MUST be in the range 0 through 6, inclusive, where 0 is Sunday, 1 is Monday, and so on.

2.2.4.100 ADT_SECURITY_DESCRIPTOR

The **ADT_SECURITY_DESCRIPTOR** structure contains a security descriptor in self-relative format and a value that includes the length of the buffer that contains the descriptor. For more information, see [MS-DTYP] section 2.4.6.

```
typedef struct _ADT_SECURITY_DESCRIPTOR {
  DWORD Length;
  [size_is(Length)] unsigned char* Buffer;
} ADT_SECURITY_DESCRIPTOR,
 *PADT_SECURITY_DESCRIPTOR;
```

Length: Length of Buffer.

Buffer: Buffer for security descriptor in self-relative form. For more information, see [MS-DTYP].

2.2.4.101 NET_DFS_ENTRY_ID

The **NET_DFS_ENTRY_ID** structure specifies a DFS local partition.

```
typedef struct _NET_DFS_ENTRY_ID {
  GUID Uid;
  [string] WCHAR* Prefix;
} NET_DFS_ENTRY_ID,
 *LPNET_DFS_ENTRY_ID;
```

Uid: Specifies the unique identifier for the partition.

Prefix: A pointer to a null-terminated Unicode UTF-16 string that contains the path prefix for the partition.

2.2.4.102 NET_DFS_ENTRY_ID_CONTAINER

The **NET_DFS_ENTRY_ID_CONTAINER** structure contains a pointer to a buffer that contains **NET_DFS_ENTRY_ID** entries and a value that indicates the count of entries in the buffer.

```
typedef struct _NET_DFS_ENTRY_ID_CONTAINER {
  unsigned long Count;
  [size_is(Count)] LPNET_DFS_ENTRY_ID Buffer;
} NET_DFS_ENTRY_ID_CONTAINER,
 *LPNET_DFS_ENTRY_ID_CONTAINER;
```

Count: The count of buffer array entries returned by the method.

Buffer: An array of **NET_DFS_ENTRY_ID** entries returned by the method.

2.2.4.103 DFS_SITENAME_INFO

The **DFS_SITE_NAME_INFO** structure specifies a site name.

```
typedef struct _DFS_SITENAME_INFO {
  unsigned long SiteFlags;
  [string, unique] WCHAR* SiteName;
} DFS_SITENAME_INFO,
 *PDFS_SITENAME_INFO,
 *LPDFS_SITENAME_INFO;
```

SiteFlags: This member MUST be a combination of zero or more of the following values.

Value	Meaning
DFS_SITE_PRIMARY 0x00000001	The site name was returned by the DsrGetSiteName method, as specified in [MS-NRPC] section **3.5.4.2.6**.

SiteName: A pointer to a null-terminated Unicode UTF-16 string that specifies a unique site name.

2.2.4.104 DFS_SITELIST_INFO

The **DFS_SITELIST_INFO** structure contains a value that indicates the count of entries and an array of **DFS_SITELIST_INFO** entries that the **NetrDfsManagerReportSiteInfo** method returns.

```
typedef struct _DFS_SITELIST_INFO {
```

```
  unsigned long cSites;
  [size_is(cSites)] DFS_SITENAME_INFO Site[];
} DFS_SITELIST_INFO,
 *PDFS_SITELIST_INFO,
 *LPDFS_SITELIST_INFO;
```

cSites: Count of site array entries returned by the method.

Site: Array of **DFS_SITENAME_INFO** entries that the method returns.

3 Protocol Details

The methods in this RPC interface all return 0x00000000 to indicate success and a nonzero, implementation-specific, error code to indicate failure. Unless otherwise specified, a server-side implementation of this protocol may choose any nonzero Win32 error value to signify an error condition, as specified in Vendor-Extensible Fields (section 1.8). The client side of the Server Service Remote Protocol MUST NOT interpret returned error codes. The client side of the Server Service Remote Protocol MUST simply return error codes to the invoking application without taking any protocol action.

Note that the terms "client side" and "server side" refer to the initiating and receiving ends of the protocol respectively rather than to client or server versions of an operating system. These methods MUST all behave the same, regardless whether the server side of the protocol is running in a client or server version of an operating system.

The following sections specify details of the Server Service Remote Protocol, including abstract data models and message processing rules.

3.1 Server Details

The server responds to messages it receives from the client.

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The 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 behaviors are consistent with that described in this specification.

A server implementing this RPC interface contains several logical elements: an SMB file server, one or more network protocol transports, and a list of shared resources that the server is making available. There could also be virtual shares and services that provide SMB file server referrals for these virtual shares.<120>

One or more network protocol transports MUST be associated with a SMB file server. A transport is a protocol that MUST logically be below the file server and MUST provide reliable delivery of file server messages. If a transport is associated with a file server, it is said to be bound to or enabled for the server. The act of associating a transport with the file server is referred to as binding. The binding between a file server and a transport is represented by a "transport handle."

Transports MAY be dynamically bound (or enabled) and unbound (or disabled) from a file server. The server opens a transport handle when a transport is bound and closes it upon unbind. A transport MUST be bound to the file server for the server to receive messages through the transport. A transport has an implementation-specific name; transport names are unique on a per-computer basis.<121>

When a transport is bound to a file server, the server MUST perform the appropriate transport binding, as specified in [MS-SMB], section 2.1, for the requested transport.

The file server can make available multiple sets of resources (that is, files, printers, pipes, disks, and mailslots) for access by Common Internet File System (CIFS) clients over the network. Each set is referred to as a share and identified by a unique network name. Shares can be made dynamically available, and the act of making a share available is referred to as adding a share. Shares can also be made unavailable dynamically, which is referred to as removing a share. The server MUST keep a list of all active shares that are identified by a share identifier. If the share is marked **sticky**, the

same information MUST be stored in persistent storage. The server MUST support two-phase deletion of shares.<122>

The SMB server MUST keep a list of all active sessions that a session identifier identifies. After a session has been established, a client can connect to a share on the server by establishing a connection to the shared resource. The SMB server MUST keep a list of all active connections that a connection identifier identifies. After a client connects to a share, the client can access individual resources in the share. The SMB server MUST keep a list of all open resources (generically referred as files) that file identifiers identify.

The server MAY<123> keep track of several implementation-dependent statistics on the server performance that clients can query by calling the **NetrServerStatisticsGet** method.

If the server supports DFS, as specified in [MS-DFSC], then it MUST provide a software component called a DFS driver that processes all messages pertaining to DFS. These messages are specified in section **NetrDfsGetVersion (Opnum 43) (section 3.1.4.35)** through section **NetrDfsManagerReportSiteInfo (Opnum 52) (section 3.1.4.43)**. The server MUST keep a list of the DFS shares and links and the associated information about the shares and links.

3.1.2 Timers

Not applicable.

3.1.3 Initialization

The server MUST initialize server data stored in the server service (like server type), start the file server, and recreate any sticky shares.

3.1.4 Message Processing Events and Sequencing Rules

Methods in RPC Opnum Order

Method	Description
Opnum0NotUsedOnWire	Reserved for local use. Opnum: 0
Opnum1NotUsedOnWire	Reserved for local use. Opnum: 1
Opnum2NotUsedOnWire	Reserved for local use. Opnum: 2
Opnum3NotUsedOnWire	Reserved for local use. Opnum: 3
Opnum4NotUsedOnWire	Reserved for local use. Opnum: 4
Opnum5NotUsedOnWire	Reserved for local use. Opnum: 5
Opnum6NotUsedOnWire	Reserved for local use. Opnum: 6

Method	Description
Opnum7NotUsedOnWire	Reserved for local use. Opnum: 7
NetrConnectionEnum	Lists all connections made to a shared resource on the server or all connections established from a particular computer. Opnum: 8
NetrFileEnum	Returns information about some or all open files on a server, depending on the parameters that are specified. Opnum: 9
NetrFileGetInfo	Retrieves information about a particular opening of a server resource. Opnum: 10
NetrFileClose	Forces an open resource instance (for example, file, device, or named pipe) on the server to close. Opnum: 11
NetrSessionEnum	Provides information about sessions that are established on a server. Opnum: 12
NetrSessionDel	Ends a network session between a server and a client. Opnum: 13
NetrShareAdd	Shares a server resource. Opnum: 14
NetrShareEnum	Retrieves information about each shared resource on a server. Opnum: 15
NetrShareGetInfo	Retrieves information about a particular shared resource on the server. Opnum: 16
NetrShareSetInfo	Sets the parameters of a shared resource. Opnum: 17
NetrShareDel	Deletes a share name from a server's list of shared resources, which disconnects all connections to the shared resource. Opnum: 18
NetrShareDelSticky	Deletes a sticky share name from a server's list of shared resources, which disconnects all connections to the shared resource. Opnum: 19
NetrShareCheck	Checks whether a server is sharing a device. Opnum: 20
NetrServerGetInfo	Retrieves current configuration information for the specified server. Opnum: 21
NetrServerSetInfo	Sets a server's operating parameters.

Method	Description
	Opnum: 22
NetrServerDiskEnum	Retrieves a list of disk drives on a server. Opnum: 23
NetrServerStatisticsGet	Retrieves operating statistics for a LAN Manager server service. Opnum: 24
NetrServerTransportAdd	Binds the server to the transport protocol. Opnum: 25
NetrServerTransportEnum	Supplies information about transport protocols that the server manages. Opnum: 26
NetrServerTransportDel	Unbinds (disconnects) the transport protocol from the server. Opnum: 27
NetrRemoteTOD	Returns the time of day information from a specified server. Opnum: 28
Opnum29NotUsedOnWire	Reserved for local use. Opnum: 29
NetprPathType	Checks a path name to determine its type. Opnum: 30
NetprPathCanonicalize	Converts a path name to an implementation-specific canonical format. Opnum: 31
NetprPathCompare	Performs an implementation-specific comparison of two paths. Opnum: 32
NetprNameValidate	Performs implementation-specific checks to ensure that the specified name is a valid name for the specified type. Opnum: 33
NetprNameCanonicalize	Converts a name into an implementation-specific canonical format for the specified type. Opnum: 34
NetprNameCompare	Performs an implementation-specific comparison of two names of a specific name type. Opnum: 35
NetrShareEnumSticky	Retrieves information about each sticky shared resource on a server. Opnum: 36
NetrShareDelStart	Performs the initial phase of a two-phase share delete. Opnum: 37
NetrShareDelCommit	Performs the final phase of a two-phase share delete.

Method	Description
	Opnum: 38
NetrpGetFileSecurity	Returns a copy of the security descriptor protecting a file or directory. Opnum: 39
NetrpSetFileSecurity	Sets the security of a file or directory. Opnum: 40
NetrServerTransportAddEx	Binds the specified server to the transport protocol. This extended method allows the caller to specify information levels 1, 2, and 3 beyond what the **NetrServerTransportAdd** (section 3.1.4.22) method allows. Opnum: 41
Opnum42NotUsedOnWire	Reserved for local use. Opnum: 42
NetrDfsGetVersion	Checks if the server is a DFS server, and if so, returns an implementation-specific DFS version. Opnum: 43
NetrDfsCreateLocalPartition	Marks a share as being a DFS share. Opnum: 44
NetrDfsDeleteLocalPartition	Deletes a DFS share (prefix) on the server. Opnum: 45
NetrDfsSetLocalVolumeState	Sets a local DFS share online or offline. Opnum: 46
Opnum47NotUsedOnWire	Reserved for local use. Opnum: 47
NetrDfsCreateExitPoint	Creates a DFS link on the server. Opnum: 48
NetrDfsDeleteExitPoint	Deletes a DFS link on the server. Opnum: 49
NetrDfsModifyPrefix	Changes the path that corresponds to a DFS link on the server. Opnum: 50
NetrDfsFixLocalVolume	Adds knowledge of a new DFS share on the server. Opnum: 51
NetrDfsManagerReportSiteInfo	Gets Active Directory site information. Opnum: 52
NetrServerTransportDelEx	Unbinds (disconnects) the transport protocol from the server. Opnum: 53

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<124> because it does not affect interoperability.

An implementation MAY<125> choose to support the methods whose names begin with NetrDfs.

The methods MUST NOT throw an exception.

The server MUST ignore the *ServerName* parameter when processing any message.<126>

3.1.4.1 NetrConnectionEnum (Opnum 8)

The **NetrConnectionEnum** method lists all the connections made to a shared resource on the server or all connections established from a particular computer.

```
NET_API_STATUS NetrConnectionEnum(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string, unique] WCHAR* Qualifier,
  [in, out] LPCONNECT_ENUM_STRUCT InfoStruct,
  [in] DWORD PreferedMaximumLength,
  [out] DWORD* TotalEntries,
  [in, out, unique] DWORD* ResumeHandle
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Qualifier: A pointer to a null-terminated Unicode UTF-16 string that specifies a share name or computer name for the connections of interest to the client.

InfoStruct: A pointer to a structure, in the format of a **CONNECT_ENUM_STRUCT (section 2.2.4.5)**. The **CONNECT_ENUM_STRUCT** structure has a Level field that specifies the type of structure to return. Level MUST be one of the following values.

Value	Meaning
InfoStruct 0	InfoStruct contains a **CONNECT_INFO_0_CONTAINER (section 2.2.4.3)** type structure.
InfoStruct 1	InfoStruct contains a **CONNECT_INFO_1_CONTAINER (section 2.2.4.4)** type structure.

PreferedMaximumLength: Specifies the preferred maximum length, in bytes, of the returned data. If the value that is specified is MAX_PREFERRED_LENGTH (section 2.2.2.2), the method MUST allocate the amount of memory that is required for the data.

TotalEntries: The total number of entries that could have been enumerated if the buffer was big enough to hold all the entries.

ResumeHandle: A pointer to a value that contains a handle used to continue an existing connection search. The handle MUST be zero on the first call and left unchanged for subsequent calls. If ResumeHandle is NULL, then no resume handle MUST be stored. If this parameter is not NULL and the method returns ERROR_MORE_DATA, this parameter receives an implementation-specific nonzero value that can be passed in subsequent calls to this method to continue with the enumeration.

If this parameter is NULL or points to 0x00000000, the enumeration starts from the beginning of the list of the currently active connections.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
0x000000EA	The client request succeeded. More entries are available. Not all entries could be returned in the buffer size that is specified by *PreferredMaximumLength*.
0x0000084B	The client request succeeded. More entries are available. The buffer size that is specified by *PreferredMaximumLength* was too small to fit even a single entry.
Other values	The client request failed.

In response to a **NetrConnectionEnum** message, the server MUST return information about active connections on the server that are of interest to the client, or return an error code. If there is more than one user using a connection, it is possible to get more than one structure for the same connection that has a different user name.

The *Qualifier* parameter specifies a share name or computer name for connections of interest to the client. If it is the name of a share on the server, then the server MUST return all connections made to that share only. Share names MUST NOT begin with \\.

If it is a computer name (MUST begin with \\), the server MUST return all connections made from that computer to the server.

If this parameter is NULL or if it is a NULL string, the server MUST fail the call. The server MAY<127> also fail the call if the length of the Qualifier string that includes the terminating null character is greater than some server-defined length.

The *InfoStruct* parameter has a member Level. The valid values of Level are 0 and 1. If the Level parameter is not equal to one of the valid values, the server MUST fail the call.

If the Level parameter is 0, the server MUST return the information about connections by filling the **CONNECT_INFO_0_CONTAINER** structure in the ConnectInfo field of the *InfoStruct* parameter. The **CONNECT_INFO_0_CONTAINER** structure contains an array of CONNECT_INFO_0 structures.

If the Level parameter is 1, the server MUST return the information about connections by filling the **CONNECT_INFO_1_CONTAINER** structure in the ConnectInfo field of the *InfoStruct* parameter. If the Level parameter is not equal to one of the valid values, the server SHOULD fail the call and return an appropriate error code. The **CONNECT_INFO_1_CONTAINER** structure contains an array of CONNECT_INFO_1 structures.

The *PreferredMaximumLength* parameter specifies the maximum number of bytes that the server can allocate for the ConnectInfo buffer. If this parameter is equal to MAX_PREFERRED_LENGTH, the server MUST allocate as much memory as required to return all of the requested data. If the *PreferredMaximumLength* is insufficient to hold all of the entries, the server MUST store as many entries as will fit in the ConnectInfo buffer and return ERROR_MORE_DATA.

If the server returns NERR_Success or ERROR_MORE_DATA, it MUST set the TotalEntries parameter to equal the total number of entries that could have been enumerated from the current resume position.

If the *PreferredMaximumLength* is insufficient to hold all of the entries and if the client has specified a ResumeHandle, the server MUST set ResumeHandle to some implementation specific value that allows the server to continue with this enumeration on a subsequent call to this method with the same value for ResumeHandle.

The following rules specify processing of the ResumeHandle parameter:

- If the ResumeHandle parameter is either NULL or points to 0x00000000, the enumeration MUST start from the beginning of the list of the currently active connections.
- If the ResumeHandle parameter points to a nonzero value, the server MUST continue enumeration based on the value of ResumeHandle. The server is not required to maintain any state between calls to the **NetrConnectionEnum** method.<128>
- If the client specified a ResumeHandle and if the server returns ERROR_MORE_DATA (0x000000EA), the server MUST set ResumeHandle to some implementation-specific value that allows the server to continue with this enumeration on a subsequent call to this method, with the same value for ResumeHandle.

The server SHOULD<129> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<130> fail the call.

3.1.4.2 NetrFileEnum (Opnum 9)

The **NetrFileEnum** method MUST return information about some or all open files on a server, depending on the parameters specified, or return an error code.

```
NET_API_STATUS NetrFileEnum(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string, unique] WCHAR* BasePath,
  [in, string, unique] WCHAR* UserName,
  [in, out] PFILE_ENUM_STRUCT InfoStruct,
  [in] DWORD PreferedMaximumLength,
  [out] DWORD* TotalEntries,
  [in, out, unique] DWORD* ResumeHandle
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

BasePath: A pointer to a null-terminated Unicode UTF-16 string that specifies a path component.

UserName: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of a user.

InfoStruct: A pointer to a structure, in the format of a **FILE_ENUM_STRUCT**. The **FILE_ENUM_STRUCT** structure has a Level field that specifies the type of structure to return. Level MUST be one of the following values.

Value	Meaning
InfoStruct	InfoStruct contains a **FILE_INFO_2_CONTAINER** type structure.

Value	Meaning
2	
InfoStruct 3	InfoStruct contains a **FILE_INFO_3_CONTAINER** type structure.

PreferedMaximumLength: Specifies the preferred maximum length of returned data, in bytes. If the value that is specified is MAX_PREFERRED_LENGTH, the method MUST allocate the amount of memory required for the data.

TotalEntries: Total number of entries that could have been enumerated if the buffer was big enough to hold all of the entries.

ResumeHandle: A pointer to a value that contains a handle used to continue an existing connection search. The handle MUST be zero on the first call and left unchanged for subsequent calls. If ResumeHandle is NULL, then no resume handle MUST be stored. If this parameter is not NULL and the method returns ERROR_MORE_DATA, this parameter receives an implementation-specific nonzero value that can be passed in subsequent calls to this method to continue with the enumeration.

If this parameter is NULL or points to 0x00000000, the enumeration starts from the beginning of the list of the currently active connections.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
0x000000EA	The client request succeeded. More entries are available. Not all entries could be returned in the buffer size that is specified by *PreferredMaximumLength*.
0x0000084B	The client request succeeded. More entries are available. The buffer size that is specified by *PreferredMaximumLength* was too small to fit even a single entry.
Other values	The client request failed.

In response to a **NetrFileEnum** message, the server MUST return information about some or all open files on a server, depending on the parameters that are specified, or return an error code. This information MUST be obtained from the SMB server using an implementation-specific method.

The BasePath parameter specifies a qualifier for the returned information. If this parameter is not NULL, the server MUST enumerate only resources that have basepath as a prefix. (A prefix is the path component up to a backslash.) The server MAY<131> fail the call if the BasePath string including the terminating null character is greater than some server-defined length.

The UserName parameter MUST specify the name of a user. If this parameter is specified, the server MUST limit the files returned to those that were opened by a session whose user name matches UserName. The server MAY<132> fail the call if the length of the UserName string including the trailing null character is greater than some server-defined length.

The *InfoStruct* parameter has a member Level. The valid values of Level are 2 and 3. If the Level parameter is not equal to one of the valid values, the server MUST fail the call.

If the Level parameter is 2, the server MUST return the information about open files by filling the **FILE_INFO_2_CONTAINER** structure in the FileInfo field of the *InfoStruct* parameter. **FILE_INFO_2_CONTAINER** structure contains an array of FILE_INFO_2 structures.

If the Level parameter is 3, the server MUST return the information about open files by filling the **FILE_INFO_3_CONTAINER** structure in the FileInfo field of the *InfoStruct* parameter. The **FILE_INFO_3_CONTAINER** structure contains an array of **FILE_INFO_3_CONTAINER**. If the Level parameter is not equal to one of the valid values, the server SHOULD fail the call.

The *PreferredMaximumLength* parameter specifies the maximum number of bytes that the server can allocate for the FileInfo buffer, so that the client can limit the size of the response to a value that it is prepared to handle.

If this parameter is equal to MAX_PREFERRED_LENGTH (section 2.2.2.2), the server MUST allocate as much memory as required to return all of the requested data.

If the *PreferredMaximumLength* is insufficient to hold all of the entries, the server MUST store as many entries as will fit in the FileInfo buffer and return ERROR_MORE_DATA.

If the server returns NERR_Success or ERROR_MORE_DATA, it MUST set *TotalEntries* parameter equal to the total number of entries that could have been enumerated from the current resume position.

If the *PreferredMaximumLength* is insufficient to hold all of the entries and if the client has specified a *ResumeHandle*, the server MUST set *ResumeHandle* to some implementation specific value that allows the server to continue with this enumeration on a subsequent call to this method with the same value for *ResumeHandle*.

If the *ResumeHandle* parameter is nonzero, then the server MUST continue enumeration based on the value of ResumeHandle. The server is not required to maintain any state between calls to NetrFileEnum method.

If the *ResumeHandle* parameter is equal to 0 or NULL, then the enumeration MUST start from the beginning of the list of currently open files.<133>

The server SHOULD<134> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<135> fail the call.

3.1.4.3 NetrFileGetInfo (Opnum 10)

The **NetrFileGetInfo** method MUST retrieve information about a particular opening of a server resource, or return an error code.

```
NET_API_STATUS NetrFileGetInfo(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] DWORD FileId,
  [in] DWORD Level,
  [out, switch_is(Level)] LPFILE_INFO InfoStruct
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

FileId: Specifies the file identifier of the open resource to return information for. The value of this parameter MUST have been returned in a previous **NetrFileEnum** call.

Note The *FileId* parameter returned in a previous **NetrFileEnum** call is not guaranteed to be valid. Therefore, the **NetrFileGetInfo** method is not guaranteed to succeed based on the validity of the *FileId* parameter.

Level: Specifies the information level of the data. This parameter MUST be one of the following values.

Value	Meaning
InfoStruct 2	**FILE_INFO_2**
InfoStruct 3	**FILE_INFO_3**

InfoStruct: This parameter is of type LPFILE_INFO, which is defined in section **FILE_INFO**. Its contents are determined by the value of the *Level* parameter, as shown in the previous parameter table.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrFileGetInfo** message, the server MUST return information about a particular opening of a server resource (file, device, or named pipe), or return an error code.

The *FileId* parameter specifies the file identifier of the open resource to return information for. The value of this parameter MUST have been returned in a previous **NetrFileEnum** message response by the server.

The *Level* parameter can be either 2 or 3. If the value of the *Level* parameter is anything else, the server MUST fail the call. The value of the *Level* parameter determines the format of the *InfoStruct* parameter.

If the value of the *Level* parameter is 2, the server MUST return information about the file whose file identifier is *FileId* by filling the **FILE_INFO_2** structure in the FileInfo2 of the *InfoStruct* parameter.

If the value of the *Level* parameter is 3, the server MUST return information about the file whose file identifier is *FileId* by filling the **FILE_INFO_3** structure in the FileInfo3 field of the *InfoStruct* parameter. If the *Level* parameter is not equal to one of the valid values, the server SHOULD fail the call and return an implementation-specific error code.

If there is no open resource whose file identifier is the same as *FileId*, the server MUST fail the call with an implementation-specific error code.

The server SHOULD<136> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<137> fail the call.

3.1.4.4 NetrFileClose (Opnum 11)

The server receives the **NetrFileClose** method in an RPC_REQUEST packet. In response, the server MUST force an open resource instance (for example, file, device, or named pipe) on the server to close. This message can be used when an error prevents closure by any other means.

```
NET_API_STATUS NetrFileClose(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] DWORD FileId
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

FileId: Specifies the file identifier of the open file, device, or pipe to close.

Note *FileId* parameter returned in a previous **NetrFileEnum** call is not guaranteed to be valid. Therefore, the **NetrFileClose** method is not guaranteed to succeed based on the validity of the *FileId* parameter.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

This message can be used when an error prevents closure by any other means.

The *FileId* parameter specifies the file identifier of the open resource instance to close. The value of the FileId parameter MUST correspond to a FileId returned in a previous **NetrFileEnum** message response by the server.

If there is no open resource whose file identifier is the same as *FileId*, the server MUST fail the call with an implementation-specific error code.

The server SHOULD<138> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<139> fail the call.

3.1.4.5 NetrSessionEnum (Opnum 12)

The **NetrSessionEnum** method MUST return information about sessions that are established on a server, or return an error code.

```
NET_API_STATUS NetrSessionEnum(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string, unique] WCHAR* ClientName,
  [in, string, unique] WCHAR* UserName,
  [in, out] PSESSION_ENUM_STRUCT InfoStruct,
  [in] DWORD PreferedMaximumLength,
  [out] DWORD* TotalEntries,
  [in, out, unique] DWORD* ResumeHandle
```

```
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

ClientName: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the computer session for which information is to be returned. This string MUST either be NULL or it MUST begin with \\.

UserName: A pointer to a null-terminated Unicode UTF-16 string that specifies the user name for which information is to be returned.

InfoStruct: A pointer to a structure, in the format of a **SESSION_ENUM_STRUCT**. The **SESSION_ENUM_STRUCT** structure has a **Level** field that specifies the type of structure to return. **Level** MUST be one of the following values.

Value	Meaning
SESSION_INFO_0_CONTAINER 0	*InfoStruct* contains a **SESSION_INFO_0_CONTAINER** type structure.
SESSION_INFO_1_CONTAINER 1	*InfoStruct* contains a **SESSION_INFO_1_CONTAINER** type structure.
SESSION_INFO_2_CONTAINER 2	*InfoStruct* contains a **SESSION_INFO_2_CONTAINER** type structure.
SESSION_INFO_10_CONTAINER 10	*InfoStruct* contains a **SESSION_INFO_10_CONTAINER** type structure.
SESSION_INFO_502_CONTAINER 502	*InfoStruct* contains a **SESSION_INFO_502_CONTAINER** type structure.

PreferedMaximumLength: Specifies the preferred maximum length, in bytes, of the returned data. If the value that is specified is MAX_PREFERRED_LENGTH, the method MUST allocate the amount of memory that is required for the data.

TotalEntries: The total number of entries that could have been enumerated if the buffer was big enough to hold all of the entries.

ResumeHandle: A pointer to a value that contains a handle used to continue an existing connection search. The handle MUST be zero on the first call and remain unchanged for subsequent calls. If the *ResumeHandle* parameter is NULL, then no resume handle MUST be stored. If this parameter is not NULL and the method returns ERROR_MORE_DATA, this parameter receives an implementation-specific nonzero value that can be passed in subsequent calls to this method to continue with the enumeration.

If this parameter is NULL or points to 0x00000000, the enumeration starts from the beginning of the list of the currently active connections.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
0x000000EA	The client request succeeded. More entries are available. Not all entries could be returned in the buffer size that is specified by *PreferredMaximumLength*.
0x0000084B	The client request succeeded. More entries are available. The buffer size that is specified by *PreferredMaximumLength* was too small to fit even a single entry.
Other values	The client request failed.

In response to **NetrSessionEnum** message, the server MUST return information about sessions that are established on the server, or return an error code.

The *ClientName* parameter specifies a qualifier for the returned information. If a *ClientName* is specified (that is, it is neither NULL nor an empty string), the client computer name MUST match the *ClientName* for the session to be returned.

If a *ClientName* is specified, it MUST start with "\\"; otherwise, the server MUST fail the call with an implementation-specific error code. If a *ClientName* is specified and it contains more than 1024 characters including the terminating NULL character, the server MUST fail the call with an implementation-specific error code.

The *UserName* parameter specifies a qualifier for the returned information. If a *UserName* is specified (that is, it is neither NULL nor an empty string), the user name MUST match the *UserName* parameter for the session to be returned. If a *UserName* parameter is specified and the length of the *UserName* string including the terminating NULL character is greater than 1024, the server MUST fail the call with an implementation-specific error code.

The server MUST return only those sessions that match all specified qualifiers. If no entries matching the qualifiers (ClientName/UserName) are found when a qualifier was specified, the server MUST fail the call with an implementation-specific error code.

The *InfoStruct* parameter has a member Level. The valid values of level are 0, 1, 2, 10 and 502. If the *Level* parameter is not equal to one of the valid values, the server MUST fail the call with an implementation-specific error code.

If the *Level* parameter is 0, the server MUST return the information about sessions by filling the **SESSION_INFO_0_CONTAINER** structure in the SessionInfo field of the *InfoStruct* parameter. The **SESSION_INFO_0_CONTAINER** structure contains an array of **SESSION_INFO_0** structures.

If the *Level* parameter is 1, the server MUST return the information about sessions by filling the **SESSION_INFO_1_CONTAINER** structure in the SessionInfo field of the *InfoStruct* parameter. The **SESSION_INFO_1_CONTAINER** structure contains an array of **SESSION_INFO_1** structures.

If the *Level* parameter is 2, the server MUST return the information about sessions by filling the **SESSION_INFO_2_CONTAINER** structure in the SessionInfo field of the *InfoStruct* parameter. The **SESSION_INFO_2_CONTAINER** structure contains an array of **SESSION_INFO_2** structures.

If the *Level* parameter is 10, the server MUST return the information about sessions by filling the **SESSION_INFO_10_CONTAINER** structure in the SessionInfo field of the *InfoStruct* parameter.

The **SESSION_INFO_10_CONTAINER** structure contains an array of **SESSION_INFO_10** structures.

If the *Level* parameter is 502, the server MUST return the information about sessions by filling the **SESSION_INFO_502_CONTAINER** structure in the SessionInfo field of the *InfoStruct* parameter. The **SESSION_INFO_502_CONTAINER** structure contains an array of **SESSION_INFO_502** structures.

The *PreferredMaximumLength* parameter specifies the maximum number of bytes that the server can allocate for the SessionInfo buffer. If this parameter equals MAX_PREFERRED_LENGTH, the server MUST allocate as much memory as required to return all the requested data. If *PreferredMaximumLength* is insufficient to hold all the entries, the server MUST store as many entries as will fit in the SessionInfo buffer and return ERROR_MORE_DATA.

If the server returns NERR_Success or ERROR_MORE_DATA, it MUST set the *TotalEntries* parameter to equal the total number of entries that could have been enumerated from the current resume position.

If the *PreferredMaximumLength* is insufficient to hold all the entries and if the client has specified a *ResumeHandle*, the server MUST set *ResumeHandle* to some implementation-specific value that allows the server to continue with this enumeration on a subsequent call to this method with the same value for *ResumeHandle*.

The following rules specify processing of the *ResumeHandle* parameter:

- If the *ResumeHandle* parameter is either NULL or points to 0x00000000, the enumeration MUST start from the beginning of the list of the currently existing connection search.
- If the *ResumeHandle* parameter points to a nonzero value, the server MUST continue the enumeration based on the value of *ResumeHandle*. The server is not required to maintain any state between calls to **NetrSessionEnum** method.<140>
- If the client specified a *ResumeHandle* and if the server returns ERROR_MORE_DATA (0x000000EA), the server MUST set *ResumeHandle* to some implementation-specific value that allows the server to continue with this enumeration on a subsequent call to this method, with the same value for *ResumeHandle*.

The server SHOULD<141> enforce appropriate security measures to be sure that the caller has required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<142> fail the call.

3.1.4.6 NetrSessionDel (Opnum 13)

The **NetrSessionDel** method MUST end one or more network sessions between a server and a client.

```
NET_API_STATUS NetrSessionDel(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string, unique] WCHAR* ClientName,
  [in, string, unique] WCHAR* UserName
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

ClientName: A pointer to a null-terminated UTF-16 Unicode string that specifies the computer name of the client whose sessions are to be disconnected. This string MUST either be NULL or it MUST begin with \\.

UserName: A pointer to a null-terminated Unicode UTF-16 string that specifies the user name whose sessions are to be terminated.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrSessionDel** message, the server ends network sessions between the server and a workstation.

The *ClientName* parameter specifies the computer name of the client to disconnect. If a *ClientName* is specified, it MUST start with "\\"; otherwise, the server MUST fail the call with an implementation-specific error code. If a *ClientName* is specified and it contains more than 1024 characters including the terminating NULL character, the server MUST fail the call with an implementation-specific error code.

The *UserName* parameter specifies the name of the user whose session is to be terminated. If a *UserName* is specified and if the length of the *UserName* string including the terminating null character is greater than 1024, the server MUST fail the call with an implementation-specific error code.

If both *ClientName* and *UserName* are unspecified (either NULL or empty string), the server MUST fail the call with an implementation-specific error code.

The server MUST delete the sessions for which the client computer name matches the *ClientName* (if it is specified) and the user name matches the *UserName* (if it is specified).

The server SHOULD<143> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<144> fail the call.

3.1.4.7 NetrShareAdd (Opnum 14)

The **NetrShareAdd** method shares a server resource.

```
NET_API_STATUS NetrShareAdd(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] DWORD Level,
  [in, switch_is(Level)] LPSHARE_INFO InfoStruct,
  [in, out, unique] DWORD* ParmErr
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Level: Specifies the information level of the data. This parameter MUST be one of the following values.

Name	Value
LPSHARE_INFO_2	2
LPSHARE_INFO_502_I	502

InfoStruct: A pointer to the LPSHARE_INFO union as specified in **SHARE_INFO (section 2.2.3.6)**. The contents of InfoStruct depends on the *Level* parameter, as shown in the previous table.

ParmErr: A pointer to a value that receives the index of the first member of the share information structure that caused the ERROR_INVALID_PARAMETER error, if it occurs.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrShareAdd** message, the server MUST share a server resource or return an error code. A shared resource is a local resource on a server (for example, a disk directory, print device, or named pipe) that can be accessed by users and applications on the network.

The *Level* parameter determines the type of structure that the client has used to specify information about the new share. The value of the Level parameter MUST be 2 or 502. If the *Level* parameter is not equal to one of the valid values, the server MUST fail the call and return an implementation-specific error code.

If the *Level* parameter is 2, InfoStruct contains a **SHARE_INFO_2** structure as specified in section 2.2.4.24. If the Level parameter is 502, InfoStruct contains a **SHARE_INFO_502_I** structure as defined in **SHARE_INFO_502_I**.

The name of the share to be added is specified in the shi*_netname member of the **SHARE_INFO** structure. If the specified share name is an empty string, is a non-empty string of length greater than 80 characters, or is "pipe" or "mailslot", the server SHOULD fail the call with an implementation-specific error code.

The server MUST validate all information that is provided in the **SHARE_INFO** (section 2.2.3.6) structure in an implementation-specific<145> manner, and if any member of the **SHARE_INFO** structure is found to be invalid, the server MUST fail the call.

If the *ParmErr* parameter is not NULL and the server finds a member of the **SHARE_INFO** structure to be invalid, the server MUST set *ParmErr* to a value that denotes the index of the member that was found to have an invalid value. The mapping between the values to set and the corresponding member is listed in section SHARE_INFO Parameter Error Codes.

The server SHOULD<146> enforce the appropriate security measures to be sure that the caller has the required permissions to execute this call. If the caller does not have the required credentials, the server SHOULD<147> fail the call.

3.1.4.8 NetrShareEnum (Opnum 15)

The **NetrShareEnum** method retrieves information about each shared resource on a server.

```
NET_API_STATUS NetrShareEnum(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, out] LPSHARE_ENUM_STRUCT InfoStruct,
  [in] DWORD PreferedMaximumLength,
  [out] DWORD* TotalEntries,
  [in, out, unique] DWORD* ResumeHandle
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

InfoStruct: A pointer to a structure, in the format of a **SHARE_ENUM_STRUCT**, as specified in section 2.2.4.36. The **SHARE_ENUM_STRUCT** structure has a **Level** field that specifies the type of structure to return in the member ShareInfo. The **Level** field MUST be one of the following values.

Value	Meaning
0	ShareInfo is of type **SHARE_INFO_0_CONTAINER (section 2.2.4.31)**.
1	ShareInfo is of type **SHARE_INFO_1_CONTAINER (section 2.2.4.32)**.
2	ShareInfo is of type **SHARE_INFO_2_CONTAINER (section 2.2.4.33)**.
501	ShareInfo is of type **SHARE_INFO_501_CONTAINER (section 2.2.4.34)**.
502	ShareInfo is of type **SHARE_INFO_502_CONTAINER (section 2.2.4.35)**.

PreferedMaximumLength: Specifies the preferred maximum length, in bytes, of the returned data. If the specified value is MAX_PREFERRED_LENGTH, the method MUST allocate the amount of memory that is required for the data.

TotalEntries: The total number of entries that could have been enumerated if the buffer was big enough to hold all the entries.

ResumeHandle: A pointer to a value that contains a handle that is used to continue an existing connection search. The handle MUST be zero on the first call and remain unchanged for subsequent calls. If the *ResumeHandle* parameter is NULL, no resume handle MUST be stored. If this parameter is not NULL and the method returns ERROR_MORE_DATA, this parameter receives an implementation-specific nonzero value that can be passed in subsequent calls to this method to continue with the enumeration.

If this parameter is NULL or points to 0x00000000, the enumeration starts from the beginning of the list of the currently active connections.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
0x000000EA	The client request succeeded. More entries are available. Not all entries could be returned in the buffer size that is specified by *PreferredMaximumLength*.
0x0000084B	The client request succeeded. More entries are available. The buffer size that is specified by *PreferredMaximumLength* was too small to fit even a single entry.
Other values	The client request failed.

In response to a **NetrShareEnum** message, the server MUST return information about each shared resource on a server, or return an error code.

The *InfoStruct* parameter has a *Level* member. The valid values of level are 0, 1, 2, 501, and 502. If the *Level* parameter is not equal to one of the valid values, the server MUST fail the call with an implementation-specific error code.

If the *Level* parameter is 0, the server MUST return the information about shares by filling the **SHARE_INFO_0_CONTAINER** structure in the **ShareInfo** field of the *InfoStruct* parameter. The **SHARE_INFO_0_CONTAINER** structure contains an array of **SHARE_INFO_0** structures.

If the *Level* parameter is 1, the server MUST return the information about shares by filling the **SHARE_INFO_1_CONTAINER** structure in the **ShareInfo** field of the *InfoStruct* parameter. The **SHARE_INFO_1_CONTAINER** structure contains an array of **SHARE_INFO_1** structures.

If the *Level* parameter is 2, the server MUST return the information about shares by filling the **SHARE_INFO_2_CONTAINER** structure in the **ShareInfo** field of the *InfoStruct* parameter. The **SHARE_INFO_2_CONTAINER** structure contains an array of **SHARE_INFO_2** structures.

If the Level parameter is 501, the server MUST return the information about shares by filling the **SHARE_INFO_501_CONTAINER** structure in the ShareInfo field of the *InfoStruct* parameter. The **SHARE_INFO_501_CONTAINER** structure contains an array of **SHARE_INFO_501** structures.

If the *Level* parameter is 502, the server MUST return the information about shares by filling the **SHARE_INFO_502_CONTAINER** structure in the ShareInfo field of the *InfoStruct* parameter. The **SHARE_INFO_502_CONTAINER** structure contains an array of **SHARE_INFO_502_I** structures.

The *PreferredMaximumLength* parameter specifies the maximum number of bytes that the server can allocate for the ShareInfo buffer. If this parameter is equal to MAX_PREFERRED_LENGTH (section 2.2.2.2), the server MUST allocate as much memory as required to return all the requested data. If the *PreferredMaximumLength* is insufficient to hold all the entries, the server MUST store as many entries as will fit in the ShareInfo buffer and return ERROR_MORE_DATA.

If the server returns NERR_Success or ERROR_MORE_DATA, it MUST set the *TotalEntries* parameter to equal the total number of entries that could have been enumerated from the current resume position.

If the *PreferredMaximumLength* is insufficient to hold all the entries and if the client has specified a *ResumeHandle*, the server MUST set *ResumeHandle* to some implementation-specific value that allows the server to continue with this enumeration on a subsequent call to this method with the same value for *ResumeHandle*.

The following rules specify processing of the *ResumeHandle* parameter:

- If the *ResumeHandle* parameter is either NULL or points to 0x00000000, the enumeration MUST start from the beginning of the list of the currently active shares.
- If the *ResumeHandle* parameter points to a nonzero value, the server MUST continue the enumeration based on the value of *ResumeHandle*. The server is not required to maintain any state between calls to **NetrShareEnum** method.<148>
- If the client specified a *ResumeHandle* and if the server returns ERROR_MORE_DATA (0x000000EA), the server MUST set *ResumeHandle* to some implementation-specific value that allows the server to continue with this enumeration on a subsequent call to this method, using the same value for ResumeHandle.

The server SHOULD<149> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<150> fail the call.

3.1.4.9 NetrShareEnumSticky (Opnum 36)

The **NetrShareEnumSticky** method retrieves information about each sticky shared resource on a server.

```
NET_API_STATUS NetrShareEnumSticky(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, out] LPSHARE_ENUM_STRUCT InfoStruct,
  [in] DWORD PreferedMaximumLength,
  [out] DWORD* TotalEntries,
  [in, out, unique] DWORD* ResumeHandle
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

InfoStruct: A pointer to a structure, in the format of a **SHARE_ENUM_STRUCT**. The **SHARE_ENUM_STRUCT** structure has a **Level** field that specifies the type of structure to return in the **ShareInfo** member. The **Level** field MUST be one of the following values.

Value	Meaning
0	ShareInfo is of type **SHARE_INFO_0_CONTAINER**.
1	ShareInfo is of type **SHARE_INFO_1_CONTAINER**.
2	ShareInfo is of type **SHARE_INFO_2_CONTAINER**.
502	ShareInfo is of type **SHARE_INFO_502_CONTAINER**.

PreferedMaximumLength: Specifies the preferred maximum length, in bytes, of the returned data. If the specified value is MAX_PREFERRED_LENGTH, the method allocates the amount of memory that is required for the data.

TotalEntries: The total number of entries that could have been enumerated if the buffer was big enough to hold all the entries.

ResumeHandle: A pointer to a value that contains a handle that is used to continue an existing connection search. The handle MUST be zero on the first call and remain unchanged for subsequent calls. If the *ResumeHandle* parameter is NULL, no resume handle MUST be stored. If this parameter is not NULL and the method returns ERROR_MORE_DATA, this parameter receives an implementation-specific nonzero value that can be passed in subsequent calls to this method to continue with the enumeration.

If this parameter is NULL or points to 0x00000000, the enumeration starts from the beginning of the list of the currently active connections.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
0x000000EA	The client request succeeded. More entries are available. Not all entries could be returned in the buffer size that is specified by *PreferredMaximumLength*.
0x0000084B	The client request succeeded. More entries are available. The buffer size that is specified by *PreferredMaximumLength* was too small to fit even a single entry.
Other values	The client request failed.

In response to a **NetrShareEnumSticky** message, the server MUST enumerate all the sticky shares, or return an error code. If the server is restarted, any shares that are created before the restart that are not sticky MUST be forgotten. Information about sticky shares MUST be stored in a persistent store,<151> and the shares MUST be restored (that is, recreated on the server) after the server is restarted.

NetrShareEnumSticky MUST NOT support Level 501, and MUST enumerate only sticky shares. Other than this difference, the server MUST process this message in exactly the same manner as the NetrShareEnum message above.

3.1.4.10 NetrShareGetInfo (Opnum 16)

The **NetrShareGetInfo** method retrieves information about a particular shared resource on the server.

```
NET_API_STATUS NetrShareGetInfo(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* NetName,
  [in] DWORD Level,
  [out, switch_is(Level)] LPSHARE_INFO InfoStruct
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

NetName: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the share to return information for.

Level: Specifies the information level of the data. This parameter MUST be one of the following values.

Value	Meaning
0	**LPSHARE_INFO_0**
1	**LPSHARE_INFO_1**
2	**LPSHARE_INFO_2**
501	**LPSHARE_INFO_501**
502	**LPSHARE_INFO_502_I**
1005	**LPSHARE_INFO_1005**

InfoStruct: This parameter is of type **LPSHARE_INFO** union, as specified in section 2.2.3.6. Its contents are determined by the value of the *Level* parameter, as shown in the previous table.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrShareGetInfo** message, the server MUST return information about a particular shared resource on a server, or return an error code.

The *NetName* parameter specifies the name of the share for which to return information. This MUST be a non-empty null-terminated Unicode UTF-16 string; otherwise, the server MUST fail the call with an implementation-specific error code.

The value of the *Level* parameter can be 0, 1, 2, 501, 502, or 1005. If the value of the *Level* parameter is anything else, the server MUST fail the call with an implementation-specific error code. The value of the *Level* parameter determines the format of the *InfoStruct* parameter.

If the value of the *Level* parameter is 0, the server MUST return information about the share whose name matches *NetName* by filling the SHARE_INFO_0 structure in the **ShareInfo0** member of the *InfoStruct* parameter.

If the value of the *Level* parameter is 1, the server MUST return information about the share whose name matches *NetName* by filling the **SHARE_INFO_1** structure in the ShareInfo1 member of the *InfoStruct* parameter.

If the value of the *Level* parameter is 2, the server MUST return information about the share whose name matches *NetName* by filling the **SHARE_INFO_2** structure in the **ShareInfo2** member of the *InfoStruct* parameter.

If the value of the *Level* parameter is 501, the server MUST return information about the share whose name matches *NetName* by filling the **SHARE_INFO_501** structure in the **ShareInfo501** member of the *InfoStruct* parameter.

If the value of the *Level* parameter is 502, the server MUST return information about the share whose name matches *NetName* by filling the **SHARE_INFO_502_I** structure in the **ShareInfo502** member of the *InfoStruct* parameter.

If the value of the *Level* parameter is 1005, the server MUST return information about the share whose name matches *NetName* by filling the **SHARE_INFO_1005** structure in the **ShareInfo1005** member of the *InfoStruct* parameter.

If there is no active share whose name matches *NetName*, the server MUST fail the call with an implementation-specific error code.

The server SHOULD<152> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<153> fail the call.

3.1.4.11 NetrShareSetInfo (Opnum 17)

The **NetrShareSetInfo** method sets the parameters of a shared resource.

```
NET_API_STATUS NetrShareSetInfo(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* NetName,
  [in] DWORD Level,
  [in, switch_is(Level)] LPSHARE_INFO ShareInfo,
  [in, out, unique] DWORD* ParmErr
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

NetName: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the share to set information for.

Level: Specifies the information level of the data. This parameter MUST be one of the following values.

Value	Meaning
1	**LPSHARE_INFO_1**
2	**LPSHARE_INFO_2**
502	**SHARE_INFO_502_I**
1004	**LPSHARE_INFO_1004**
1005	**LPSHARE_INFO_1005**
1006	**LPSHARE_INFO_1006**
1501	**LPSHARE_INFO_1501_I**

ShareInfo: This parameter is of type **LPSHARE_INFO** union, as specified in section 2.2.3.6. Its contents are determined by the value of the *Level* parameter, as shown in the previous table. This parameter MUST NOT contain a null value.

ParmErr: A pointer to a value that receives the index of the first member of the share information structure that caused the ERROR_INVALID_PARAMETER error, if it occurs.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrShareSetInfo** message, the server MUST set the parameters of a shared resource or return an error code.

The *NetName* parameter specifies the name of the share to set information for. This MUST be a non-empty null-terminated Unicode UTF-16 string; otherwise, the server MUST fail the call with an implementation-specific error code.

The value of the *Level* parameter can be 1, 2, 502, 1004, 1005, 1006, or 1501. If the value of the *Level* parameter is anything else, the server MUST fail the call with an implementation-specific error code. The value of the *Level* parameter determines the format of the *InfoStruct* parameter.

On receiving the **NetrShareSetInfo** method, the server MUST update the settings that correspond to the *InfoStruct* parameter for the share whose name matches *NetName*. The format for the *InfoStruct* parameter is defined in SHARE_INFO (section 2.2.3.6).

If the *Level* parameter is equal to 1, all the settings that are defined by the **SHARE_INFO_1** structure MUST be updated.

If the *Level* parameter is equal to 2, all the settings that are defined by the **SHARE_INFO_2** structure MUST be updated.

If the *Level* parameter is equal to 502, all the settings that are defined by the **SHARE_INFO_502_I** structure MUST be updated.

If the *Level* parameter is equal to 1004, all the settings that are defined by the **SHARE_INFO_1004** structure MUST be updated.

If the *Level* parameter is equal to 1005, all the settings that are defined by the **SHARE_INFO_1005** structure MUST be updated. Only disk shares can be affected by this Level.<154>

If the *Level* parameter is equal to 1006, all the settings that are defined by the **SHARE_INFO_1006** structure MUST be updated.

If the *Level* parameter is equal to 1501, all the settings that are defined by the **SHARE_INFO_1501_I** structure MUST be updated.

If there is no active share whose name matches *NetName*, the server MUST fail the call with an implementation-specific error code.

The server MUST validate all information that is provided in the **SHARE_INFO** structure in an implementation-specific manner, and if any member of the **SHARE_INFO** structure is found to be invalid, the server MUST fail the call with an implementation-specific<155> error code.

If the *ParmErr* parameter is not NULL and the server finds a member of the **SHARE_INFO** structure to be invalid, the server MUST set *ParmErr* to a value that denotes the index of the member that

was found to have invalid value. The mapping between the values to set and the corresponding member MUST be as specified in SHARE_INFO Parameter Error Codes.

The server SHOULD<156> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<157> fail the call.

3.1.4.12 NetrShareDel (Opnum 18)

The **NetrShareDel** method deletes a share name from a server's list of shared resources, which disconnects all connections to the shared resource. If the share is sticky, all information about the share is also deleted from permanent storage (registry).

```
NET_API_STATUS NetrShareDel(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* NetName,
  [in] DWORD Reserved
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

NetName: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the share to delete.

Reserved: The server MUST ignore this parameter.<158>

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrShareDel** message, the server MUST delete a share name from the server's list of shared resources, which disconnects all connections to the shared resource, or MUST return an error code.

The *NetName* parameter specifies the name of the share to delete. This MUST be a non-empty null-terminated Unicode UTF-16 string; otherwise, the server MUST fail the call with an implementation-specific error code.

If no share matching *NetName* exists, the server fails the call with an implementation-specific error code.

The server SHOULD<159> enforce appropriate security measures to ensure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<160> fail the call.

3.1.4.13 NetrShareDelSticky (Opnum 19)

The **NetrShareDelSticky** method deletes a sticky share name from a server's list of shared resources, which disconnects all connections to the shared resource. If the share that is specified by NetName is non-sticky, the behavior is not specified.

```
NET_API_STATUS NetrShareDelSticky(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* NetName,
  [in] DWORD Reserved
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

NetName: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the share to delete.

Reserved: The server MUST ignore this parameter.<161>

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrShareDelSticky** message, the server MUST delete a sticky share from the server's list of shared resources or MUST return an error code. The primary use of this method is to delete a sticky share whose root directory has been deleted (thus preventing actual recreation of the share) but whose entry still exists in the registry. This method can also be used to remove the persistence of a share without deleting the current incarnation of the share.

The *NetName* parameter specifies the name of the share to delete. This MUST be a non-empty null-terminated Unicode UTF-16 string; otherwise, the server MUST fail the call with an implementation-specific error code.

If a share that matches the *NetName* does not exist, the server MUST fail the call with an implementation-specific error code.

The server SHOULD<162> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<163> fail the call.

3.1.4.14 NetrShareDelStart (Opnum 37)

The **NetrShareDelStart** method performs the initial phase of a two-phase share delete.

```
NET_API_STATUS NetrShareDelStart(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* NetName,
  [in] DWORD Reserved,
  [out] PSHARE_DEL_HANDLE ContextHandle
```

```
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

NetName: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the share to delete.

Reserved: Reserved, MUST be zero.

ContextHandle: A handle for the second phase of the two-phase share delete, in the form of a **PSHARE_DEL_HANDLE (section 2.2.1.3)**.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrShareDelStart** message, the server MUST store a reference to the share in its list of shares that are pending deletion; and returns to the client an RPC context handle that the client can use to actually perform the delete by calling the **NetrShareDelCommit** method; or MUST return an error code.

This two-phase deletion MUST be used to delete IPC$, which is the share that is used for named pipes. Deleting IPC$ results in the closing of the pipe on which the RPC is being executed. Thus, the client never receives the response to the RPC. The two-phase delete offers a positive response in phase 1, and then an expected error in phase 2.

The *NetName* parameter specifies the name of the share to delete. This MUST be a non-empty null-terminated Unicode UTF-16 string; otherwise, the server MUST fail the call with an implementation-specific error code.

If no share matching *NetName* exists, the server MUST fail the call with an implementation-specific error code.

The server MUST maintain a list of shares that are marked for deletion while it waits for the corresponding commit. The share MUST remain available up until the corresponding commit has been processed.

The server MUST return an RPC context handle in the *ContextHandle* parameter (which MUST NOT be NULL), which the client is expected to use to actually delete the share by calling the NetrShareDelCommit method.

The server SHOULD<164> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<165> fail the call.

3.1.4.15 NetrShareDelCommit (Opnum 38)

The **NetrShareDelCommit** method performs the final phase of a two-phase share delete.

```
NET_API_STATUS NetrShareDelCommit(
  [in, out] PSHARE_DEL_HANDLE ContextHandle
);
```

ContextHandle: A handle returned by the first phase of a two-phase share delete.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed, unless the share being deleted is IPC$. If the share being deleted is IPC$, the return value is not meaningful.

The **NetrShareDelCommit** message is the continuation of the **NetrShareDelStart** message and MUST cause the share to be actually deleted, which disconnects all connections to the share, or MUST return an error code.

This method can be used to delete the IPC$ share as well as other shares. When the share is not IPC$, only a return value of 0 indicates success.

This two-phase deletion MUST be used to delete IPC$, which is the share that is used for named pipes. Deleting IPC$ results in the closing of the pipe on which the RPC is being executed. Thus, the client never receives the response to the RPC. The two-phase delete offers a positive response in phase 1, and then an expected error in phase 2.

The *ContextHandle* parameter specifies an RPC context handle that MUST have been returned by a previous NetrShareDelStart call. This handle provides the server with the information it needs to identify the share that needs to be deleted. If this parameter is NULL or if it points to a NULL pointer, the server MUST fail the call with an implementation-specific error code.

The server does not enforce any security measures when processing this call.

3.1.4.16 NetrShareCheck (Opnum 20)

The **NetrShareCheck** method checks whether a server is sharing a device.

```
NET_API_STATUS NetrShareCheck(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* Device,
  [out] DWORD* Type
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Device: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the device to check for shared access.

Type: A pointer to a DWORD that receives the type of the shared device. This parameter is set only if the method returns successfully. This parameter MUST be STYPE_DISKTREE, as

specified in the Share Type values in section 2.2.2.4. All other Share Type values are not supported.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrShareCheck** message, the server MUST check whether it is sharing a device, and return a response to the client.

The *Device* parameter specifies the name of the shared device to check for. The server MUST enumerate the active shared devices, and if it finds a match to the *Device* parameter, the server MUST return the type of the matching device in the *Type* parameter. It can be one of the values that are listed in Share Types.

If no match is found, the server MUST fail the call with an implementation-specific error code.

No special group membership is required to successfully execute this message.

3.1.4.17 NetrServerGetInfo (Opnum 21)

The **NetrServerGetInfo** method retrieves current configuration information for the specified server.

```
NET_API_STATUS NetrServerGetInfo(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] DWORD Level,
  [out, switch_is(Level)] LPSERVER_INFO InfoStruct
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Level: Specifies the information level of the data. The value of the *Level* parameter determines the contents of the *InfoStruct* parameter. This parameter MUST be one of the following values.

Value	Meaning
100	**LPSERVER_INFO_100**
101	**LPSERVER_INFO_101**
102	**LPSERVER_INFO_102**
502	**LPSERVER_INFO_502**
503	**LPSERVER_INFO_503**

InfoStruct: This is a structure of type LPSERVER_INFO, as specified in section 2.2.3.7. The content of the *InfoStruct* parameter is determined by the *Level* parameter, as the previous table shows.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to the *NetrServerGetInfo* message, the server MUST return the current configuration information for the server or MUST return an error code.

The value of the *Level* parameter can be 100, 101, 102, 502, or 503. If the value of the *Level* parameter is anything else, the server MUST fail the call with an implementation-specific error code.

The value of the *Level* parameter determines the format of the *InfoStruct* parameter.

If the value of the *Level* parameter is 100, the server MUST return its information by filling the **SERVER_INFO_100** structure in the ServerInfo100 member of the *InfoStruct* parameter.

If the value of the *Level* parameter is 101, the server MUST return its information by filling the **SERVER_INFO_101** structure in the ServerInfo101 member of the *InfoStruct* parameter.

If the value of the *Level* parameter is 102, the server MUST return its information by filling the **SERVER_INFO_102** structure in the ServerInfo102 member of the *InfoStruct* parameter.

If the value of the *Level* parameter is 502, the server MUST return its information by filling the **SERVER_INFO_502** structure in the ServerInfo502 member of the *InfoStruct* parameter.

If the value of the *Level* parameter is 503, the server MUST return its information by filling the **SERVER_INFO_503** structure in the ServerInfo503 member of the *InfoStruct* parameter.

The server SHOULD<166> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<167> fail the call.

3.1.4.18 NetrServerSetInfo (Opnum 22)

The **NetrServerSetInfo** method sets a server's operating parameters; it can set them individually or collectively. The information is stored in a way that allows it to remain in effect after the system is reinitialized.<168>

```
NET_API_STATUS NetrServerSetInfo(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] DWORD Level,
  [in, switch_is(Level)] LPSERVER_INFO ServerInfo,
  [in, out, unique] DWORD* ParmErr
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Level: Specifies the information level of the data. The value of the *Level* parameter determines the contents of the *ServerInfo* parameter. This parameter MUST be one of the values in the following table.

Value	Meaning
101	LPSERVER_INFO_101
102	LPSERVER_INFO_102
502	LPSERVER_INFO_502
503	LPSERVER_INFO_503
599	LPSERVER_INFO_599
1005	LPSERVER_INFO_1005
1107	LPSERVER_INFO_1107
1010	LPSERVER_INFO_1010
1016	LPSERVER_INFO_1016
1017	LPSERVER_INFO_1017
1018	LPSERVER_INFO_1018
1501	LPSERVER_INFO_1501
1502	LPSERVER_INFO_1502
1503	LPSERVER_INFO_1503
1506	LPSERVER_INFO_1506
1510	LPSERVER_INFO_1510
1511	LPSERVER_INFO_1511
1512	LPSERVER_INFO_1512
1513	LPSERVER_INFO_1513
1514	LPSERVER_INFO_1514
1515	LPSERVER_INFO_1515
1516	LPSERVER_INFO_1516
1518	LPSERVER_INFO_1518
1523	LPSERVER_INFO_1523
1528	LPSERVER_INFO_1528
1529	LPSERVER_INFO_1529
1530	LPSERVER_INFO_1530
1533	LPSERVER_INFO_1533
1534	LPSERVER_INFO_1534

Value	Meaning
1535	**LPSERVER_INFO_1535**
1536	**LPSERVER_INFO_1536**
1538	**LPSERVER_INFO_1538**
1539	**LPSERVER_INFO_1539**
1540	**LPSERVER_INFO_1540**
1541	**LPSERVER_INFO_1541**
1542	**LPSERVER_INFO_1542**
1543	**LPSERVER_INFO_1543**
1544	**LPSERVER_INFO_1544**
1545	**LPSERVER_INFO_1545**
1546	**LPSERVER_INFO_1546**
1547	**LPSERVER_INFO_1547**
1548	**LPSERVER_INFO_1548**
1549	**LPSERVER_INFO_1549**
1550	**LPSERVER_INFO_1550**
1552	**LPSERVER_INFO_1552**
1553	**LPSERVER_INFO_1553**
1554	**LPSERVER_INFO_1554**
1555	**LPSERVER_INFO_1555**
1556	**LPSERVER_INFO_1556**

ServerInfo: This is a structure of type LPSERVER_INFO, as specified in section 2.2.3.7. The content of the *ServerInfo* parameter is determined by the *Level* parameter, as the previous table shows.

ParmErr: A pointer to a value that receives the index of the first member of the server information structure that caused the ERROR_INVALID_PARAMETER error, if it occurs.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a NetrServerSetInfo message, the server MUST set its operating parameters or return an error code. The server can set its operating parameters individually or collectively. The information is stored in a way that allows it to remain in effect after the system is reinitialized.

The value of the *Level* parameter can be 101, 102, 502, 503, 599, or a value greater than 1000. If the value of the *Level* parameter is anything else, the server MUST fail the call with an implementation-specific error code.

After receiving the *NetrServerSetInfo* method, the server MUST update the server's setting that corresponds to the *ServerInfo* parameter. The format for the *ServerInfo* parameter is as specified in section **SERVER_INFO**.

If the *Level* parameter is greater than 1000, the server MUST use the value (Level-1000) to specify the index of a single parameter to set, where the index MUST be one of the values specified in section 2.2.2.12. The *ServerInfo* parameter then contains a structure with a single member that MUST specify the new value of the parameter. The server MUST update the value of the local parameter with the specified new value (if the server allows that value to be modified) or fail the call with error code ERROR_INVALID_LEVEL.

If the *Level* parameter is equal to 101, the server MUST update all the settings that are defined by the **SERVER_INFO_101** structure.

If the *Level* parameter is equal to 102, the server MUST update all the settings that are defined by the **SERVER_INFO_102** structure.

If the *Level* parameter is equal to 502, the server MUST update all the settings that are defined by the **SERVER_INFO_502** structure.

If the *Level* parameter is equal to 503, the server MUST update all the settings that are defined by the **SERVER_INFO_503** structure.

If the *Level* parameter is equal to 599, the server MUST update all the settings that are defined by the **SERVER_INFO_599** structure.

If the *Level* parameter is equal to 1005, the server MUST update all the settings that are defined by the **SERVER_INFO_1005** structure.

If the *Level* parameter is equal to 1107, the server MUST update all the settings that are defined by the **SERVER_INFO_1107** structure.

If the *Level* parameter is equal to 1016, the server MUST update all the settings that are defined by the **SERVER_INFO_1016** structure.

If the *Level* parameter is equal to 1017, the server MUST update all the settings that are defined by the **SERVER_INFO_1017** structure.

If the *Level* parameter is equal to 1018, the server MUST update all the settings that are defined by the **SERVER_INFO_1018** structure.

If the *Level* parameter is equal to 1501, the server MUST update all the settings that are defined by the **SERVER_INFO_1501** structure.

If the *Level* parameter is equal to 1502, the server MUST update all the settings that are defined by the **SERVER_INFO_1502** structure.

If the *Level* parameter is equal to 1503, the server MUST update all the settings that are defined by the **SERVER_INFO_1503** structure.

If the *Level* parameter is equal to 1506, the server MUST update all the settings that are defined by the **SERVER_INFO_1506** structure.

If the *Level* parameter is equal to 1510, the server MUST update all the settings that are defined by the **SERVER_INFO_1510** structure.

If the *Level* parameter is equal to 1511, the server MUST update all the settings that are defined by the **SERVER_INFO_1511** structure.

If the *Level* parameter is equal to 1512, the server MUST update all the settings that are defined by the **SERVER_INFO_1512** structure.

If the *Level* parameter is equal to 1513, the server MUST update all the settings that are defined by the **SERVER_INFO_1513** structure.

If the *Level* parameter is equal to 1514, the server MUST update all the settings that are defined by the **SERVER_INFO_1514** structure.

If the *Level* parameter is equal to 1515, the server MUST update all the settings that are defined by the **SERVER_INFO_1515** structure.

If the *Level* parameter is equal to 1516, the server MUST update all the settings that are defined by the **SERVER_INFO_1516** structure.

If the *Level* parameter is equal to 1518, the server MUST update all the settings that are defined by the **SERVER_INFO_1518** structure.

If the *Level* parameter is equal to 1523, the server MUST update all the settings that are defined by the **SERVER_INFO_1523** structure.

If the *Level* parameter is equal to 1528, the server MUST update all the settings that are defined by the **SERVER_INFO_1528** structure.

If the *Level* parameter is equal to 1529, the server MUST update all the settings that are defined by the **SERVER_INFO_1529** structure.

If the *Level* parameter is equal to 1530, the server MUST update all the settings that are defined by the **SERVER_INFO_1530** structure.

If the *Level* parameter is equal to 1533, the server MUST update all the settings that are defined by the **SERVER_INFO_1533** structure.

If the *Level* parameter is equal to 1534, the server MUST update all the settings that are defined by the **SERVER_INFO_1534** structure.

If the *Level* parameter is equal to 1535, the server MUST update all the settings that are defined by the **SERVER_INFO_1535** structure.

If the *Level* parameter is equal to 1536, the server MUST update all the settings that are defined by the **SERVER_INFO_1536** structure.

If the *Level* parameter is equal to 1538, the server MUST update all the settings that are defined by the **SERVER_INFO_1538** structure.

If the *Level* parameter is equal to 1539, the server MUST update all the settings that are defined by the **SERVER_INFO_1539** structure.

If the *Level* parameter is equal to 1540, the server MUST update all the settings that are defined by the **SERVER_INFO_1540** structure.

If the *Level* parameter is equal to 1541, the server MUST update all the settings that are defined by the **SERVER_INFO_1541** structure.

If the *Level* parameter is equal to 1542, the server MUST update all the settings that are defined by the **SERVER_INFO_1542** structure.

If the *Level* parameter is equal to 1543, the server MUST update all the settings that are defined by the **SERVER_INFO_1543** structure.

If the *Level* parameter is equal to 1544, the server MUST update all the settings that are defined by the **SERVER_INFO_1544** structure.

If the *Level* parameter is equal to 1545, the server MUST update all the settings that are defined by the **SERVER_INFO_1545** structure.

If the *Level* parameter is equal to 1546, the server MUST update all the settings that are defined by the **SERVER_INFO_1546** structure.

If the *Level* parameter is equal to 1547, the server MUST update all the settings that are defined by the **SERVER_INFO_1547** structure.

If the *Level* parameter is equal to 1548, the server MUST update all the settings that are defined by the **SERVER_INFO_1548** structure.

If the *Level* parameter is equal to 1549, the server MUST update all the settings that are defined by the **SERVER_INFO_1549** structure.

If the *Level* parameter is equal to 1550, the server MUST update all the settings that are defined by the **SERVER_INFO_1550** structure.

If the *Level* parameter is equal to 1552, the server MUST update all the settings that are defined by the **SERVER_INFO_1552** structure.

If the *Level* parameter is equal to 1553, the server MUST update all the settings that are defined by the **SERVER_INFO_1553** structure.

If the *Level* parameter is equal to 1554, the server MUST update all the settings that are defined by the **SERVER_INFO_1554** structure.

If the *Level* parameter is equal to 1555, the server MUST update all the settings that are defined by the **SERVER_INFO_1555** structure.

If the *Level* parameter is equal to 1556, the server MUST update all the settings that are defined by the **SERVER_INFO_1556** structure.

The server MUST validate each member of the structure that is passed in the *ServerInfo* parameter. The validation involves making sure each member of the ServerInfo structure has a valid value as specified in the definition of the corresponding SERVER_INFO structure. If any member of the structure is not valid and the *ParamErr* parameter is not NULL, the server MUST set *ParamErr* to a value based on the first member of the structure that is not valid. The mapping between the values to set and the corresponding member is listed in section 2.2.2.12.<169>

The server SHOULD<170> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<171> fail the call.

3.1.4.19 NetrServerDiskEnum (Opnum 23)

The **NetrServerDiskEnum** method retrieves a list of disk drives on a server. The method returns an array of three-character strings (a drive letter, a colon, and a terminating null character).

```
NET_API_STATUS NetrServerDiskEnum(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] DWORD Level,
  [in, out] DISK_ENUM_CONTAINER* DiskInfoStruct,
  [in] DWORD PreferredMaximumLength,
  [out] DWORD* TotalEntries,
  [in, out, unique] DWORD* ResumeHandle
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Level: This parameter MUST be 0.

DiskInfoStruct: A pointer to structure of type DISK_ENUM_CONTAINER, as specified in section **DISK_ENUM_CONTAINER (section 2.2.4.89)**.<172>

PreferredMaximumLength: The server MUST ignore this parameter.

TotalEntries: Total number of entries that could have been enumerated if the buffer was big enough to hold all of the entries.

ResumeHandle: The server MUST ignore this parameter, and it MUST be zero.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	Client request succeeded.
Other values	Client request failed.

Clients use the NetrServerDiskEnum message to retrieve a list of disk drives on a server. The method MUST return an array of three-character strings (a drive letter, a colon, and a terminating null character), or return an error code.

The *Level* parameter MUST be 0; otherwise, the server MUST fail the call with an implementation-specific error code.

The *DiskInfoStruct* parameter contains an **EntriesRead** member, where the server MUST return the number of disk drive entries that the server enumerated in the other Buffer member of *DiskInfoStruct*. The server returns the enumerated disk drives in the Buffer member of *DiskInfoStruct* in the format of the DISK_INFO structure. The server MUST allocate as much memory as required to return all enumerated disk drives in the Buffer member of the *InfoStruct* parameter. In cases where the RPC has allocated a buffer because the client had specified a non-NULL value for the *Buffer* parameter, the server MUST free the buffer allocated by the RPC.

The server MUST ignore the *PreferredMaximumLength* parameter.

The server MUST set the *TotalEntries* parameter equal to the total number of entries that could have been enumerated from the current resume position.

The server MUST ignore the *ResumeHandle* parameter.

The server SHOULD<173> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<174> fail the call.

3.1.4.20 NetrServerStatisticsGet (Opnum 24)

The **NetrServerStatisticsGet** method retrieves the operating statistics for a LAN Manager server service.

```
NET_API_STATUS NetrServerStatisticsGet(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string, unique] WCHAR* Service,
  [in] DWORD Level,
  [in] DWORD Options,
  [out] LPSTAT_SERVER_0* InfoStruct
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Service: A pointer to a null-terminated Unicode UTF-16 string that SHOULD be set to the value "LanmanServer" and ignored upon receipt.

Level: Specifies the information level of the data. This SHOULD be 0 and be ignored on receipt.

Options: Reserved; MUST be 0.

InfoStruct: A pointer to the buffer in the format of **STAT_SERVER_0** that receives the data, as specified in section 2.2.4.37.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to NetrServerStatisticsGet message, the server MUST return the operating statistics for the LAN Manager server service, or return an error code.

The *Service* parameter is ignored and MUST be a null-terminated Unicode UTF-16 string with the contents "LanmanServer".

If the *Level* parameter is not equal to 0, the server MUST fail the call with an implementation-specific error code.

If the *Options* parameter is not equal to 0, the server MUST fail the call with an implementation-specific error code.

The server MUST return the statistics in the *InfoStruct* parameter in the format **STAT_SERVER_0** structure. The **STAT_SERVER_0** structure is defined in section 2.2.4.37. The server MUST return the statistics for an implementation-specific<175> file server protocol that is used on that server.

The server SHOULD<176> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<177> fail the call.

3.1.4.21 NetrRemoteTOD (Opnum 28)

The **NetrRemoteTOD** method returns the time of day information from a specified server.

```
NET_API_STATUS NetrRemoteTOD(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [out] LPTIME_OF_DAY_INFO* BufferPtr
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

BufferPtr: A pointer to structure of type **TIME_OF_DAY_INFO** where the information is returned.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	Client request succeeded.
Other values	Client request failed.

In response to a **NetrRemoteTOD** message, the server MUST return the time of day information, or return an error code.

The server MUST return the time of day information on the server in the *BufferPtr* parameter in the format of the **LPTIME_OF_DAY_INFO** structure as specified in section 2.2.4.99.

The server SHOULD<178> enforce appropriate security measures to be sure that the caller has the required permissions to execute this routine. If the caller does not have the required credentials, the server SHOULD<179> fail the call.

3.1.4.22 NetrServerTransportAdd (Opnum 25)

The **NetrServerTransportAdd** method binds the server to the transport protocol.

```
NET_API_STATUS NetrServerTransportAdd(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] DWORD Level,
  [in] LPSERVER_TRANSPORT_INFO_0 Buffer
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Level: Specifies the information level of the data. This parameter MUST be zero.

Buffer: A pointer to the **SERVER_TRANSPORT_INFO_0** structure that describes the data.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	Client request succeeded.
Other values	Client request failed.

The **NetrServerTransportAdd** message MUST be processed in the same way as NetrServerTransportAddEx except that it MUST only allow level 0 (that is, *SERVER_TRANSPORT_INFO_0*). The NetrServerTransportAddEx message is specified in section 3.1.4.23.

3.1.4.23 NetrServerTransportAddEx (Opnum 41)

The **NetrServerTransportAddEx** method binds the specified server to the transport protocol. This extended method allows the caller to specify information levels 1, 2, and 3 beyond what the *NetrServerTransportAdd* method allows.

```
NET_API_STATUS NetrServerTransportAddEx(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] DWORD Level,
  [in, switch_is(Level)] LPTRANSPORT_INFO Buffer
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Level: Specifies the information level of the data. This parameter MUST be the following value.

Value	Meaning
0	The buffer is of type **SERVER_TRANSPORT_INFO_0**.
1	The buffer is of type **SERVER_TRANSPORT_INFO_1**.
2	The buffer is of type **SERVER_TRANSPORT_INFO_2**.
3	The buffer is of type **SERVER_TRANSPORT_INFO_3**.

Buffer: A pointer to the **TRANSPORT_INFO** union that describes the data. The type of data depends on the value of the *Level* parameter, as the previous table shows.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a NetrServerTransportAddEx message, the server MUST bind the server to the transport protocol, or return an error code. It effectively means that the SMB file server can now communicate with clients using the new transport protocol.

The *Level* parameter determines the type of structure that the client has used to specify information about the new transport. The value MUST be 0, 1, 2, or 3. If the *Level* parameter is not equal to one of the valid values, the server MUST fail the call with an implementation-specific error code.

If the *Level* parameter is 0, the *Buffer* parameter points to a **SERVER_TRANSPORT_INFO_0** structure.

If the *Level* parameter is 1, the *Buffer* parameter points to a **SERVER_TRANSPORT_INFO_1** structure.

If the *Level* parameter is 2, the *Buffer* parameter points to a **SERVER_TRANSPORT_INFO_2** structure.

If the *Level* parameter is 3, the *Buffer* parameter points to a **SERVER_TRANSPORT_INFO_3** structure.

The server MUST validate all information that is provided in the SERVER_TRANSPORT_INFO structure in an implementation-specific manner. If any member of the structure is found to be invalid, the server MUST fail the call with an implementation-specific<180> error code.

The server SHOULD<181> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the caller does not have the required credentials, the server SHOULD<182> fail the call.

3.1.4.24 NetrServerTransportEnum (Opnum 26)

The **NetrServerTransportEnum** method supplies information about transport protocols that the server manages.

```
NET_API_STATUS NetrServerTransportEnum(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, out] LPSERVER_XPORT_ENUM_STRUCT InfoStruct,
  [in] DWORD PreferedMaximumLength,
  [out] DWORD* TotalEntries,
  [in, out, unique] DWORD* ResumeHandle
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

InfoStruct: A pointer to a structure, in the format of a **SERVER_XPORT_ENUM_STRUCT** structure that receives the data. The **SERVER_XPORT _ENUM_STRUCT** structure has a Level field that specifies the type of the structure to return in the XportInfo member. Level MUST be one of the following values.

Value	Meaning
0	XportInfo is of type **SERVER_XPORT_INFO_0_CONTAINER**.
1	XportInfo is of type **SERVER_XPORT_INFO_1_CONTAINER**.

PreferedMaximumLength: Specifies the preferred maximum length, in bytes, of returned data. If the value that is specified is MAX_PREFERRED_LENGTH, the method MUST allocate the amount of memory that is required for the data.

TotalEntries: The total number of entries that could have been enumerated if the buffer was big enough to hold all the entries.

ResumeHandle: A pointer to a value that contains a handle that is used to continue an existing connection search. The handle MUST be zero on the first call and remain unchanged for subsequent calls. If the *ResumeHandle* parameter is NULL, no resume handle MUST be stored. If this parameter is not NULL and the method returns ERROR_MORE_DATA, this parameter receives an implementation-specific nonzero value that can be passed in subsequent calls to this method to continue with the enumeration.

If this parameter is NULL or points to 0x00000000, the enumeration starts from the beginning of the list of the currently active connections.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
0x000000EA	The client request succeeded. More entries are available. Not all entries could be returned in the buffer size that is specified by *PreferredMaximumLength*.
0x0000084B	The client request succeeded. More entries are available. The buffer size that is specified by *PreferredMaximumLength* was too small to fit even a single entry.
Other values	The client request failed.

In response to the **NetrServerTransportEnum** message, the server MUST return information about transport protocols that the server manages, or return an error code.

The *InfoStruct* parameter has a member Level. The value of *Level* MUST be 0 or 1. If the *Level* parameter is not equal to one of the valid values, the server MUST fail the call with an implementation-specific error code.

If the value of the *Level* parameter is 0, the server MUST return the information about the transport protocols that it is managing by filling the **SERVER_XPORT _INFO_0_CONTAINER** structure in the XportInfo field of the *InfoStruct* parameter.

If the *Level* parameter is 1, the server MUST return the information about the transport protocols that it is managing by filling the **SERVER_XPORT _INFO_1_CONTAINER** structure in the XportInfo field of the *InfoStruct* parameter.

The *PreferredMaximumLength* parameter specifies the maximum number of bytes that the server can allocate for the XportInfo buffer. If this parameter is equal to MAX_PREFERRED_LENGTH, the server MUST allocate as much memory as required to return all the requested data.

If the *PreferredMaximumLength* is insufficient to hold all the entries, the server MUST store as many entries as can fit in the XportInfo buffer and return ERROR_MORE_DATA.

If the server returns NERR_Success or ERROR_MORE_DATA, it MUST set the *TotalEntries* parameter equal to the total number of entries that could have been enumerated from the current resume position.

If the *PreferredMaximumLength* is insufficient to hold all the entries and if the client has specified a ResumeHandle, the server MUST set ResumeHandle to some implementation-specific value that allows the server to continue with this enumeration on a subsequent call to this method with the same value for *ResumeHandle*.

If the *ResumeHandle* parameter is nonzero, the server MUST continue enumeration based on the value of ResumeHandle. The server is not required to maintain any state between calls to the **NetrServerTransportEnum** method.

If the *ResumeHandle* parameter is equal to 0 or NULL, the enumeration always starts from the beginning of the list of currently active shares. If the value of *ResumeHandle* is nonzero, the server MAY<183> instead enumerate shares by starting from a particular point in the list of currently active shares.

The server SHOULD<184> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the caller does not have the required credentials, the server SHOULD<185> fail the call.

3.1.4.25 NetrServerTransportDel (Opnum 27)

The **NetrServerTransportDel** method unbinds (or disconnects) the transport protocol from the server. Effectively, the server can no longer communicate with clients by using the specified transport protocol (such as TCP or XNS).

```
NET_API_STATUS NetrServerTransportDel(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] DWORD Level,
  [in] LPSERVER_TRANSPORT_INFO_0 Buffer
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Level: Specifies the information level of the data. This SHOULD be zero and MUST be ignored on receipt.

Buffer: A pointer to **SERVER_TRANSPORT_INFO_0** structure that contains information about the transport.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	Client request succeeded.
Other values	Client request failed.

The **NetrServerTransportDel** message MUST be processed in the same way as **NetrServerTransportDelEx** except that it MUST only allow level 0 (that is, **SERVER_TRANSPORT_INFO_0**). The processing for this message is specified in section 3.1.4.26 Message.

3.1.4.26 NetrServerTransportDelEx (Opnum 53)

The server receives the **NetrServerTransportDelEx** method in an RPC_REQUEST packet. In response, the server unbinds (or disconnects) the transport protocol from the server. Effectively, the server can no longer communicate with clients by using the specified transport protocol (such as TCP or XNS). This extended method allows level 1 beyond what NetrServerTransportDel allows.

```
NET_API_STATUS NetrServerTransportDelEx(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] DWORD Level,
  [in, switch_is(Level)] LPTRANSPORT_INFO Buffer
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Level: Specifies the information level of the data. It MUST be one of the following values.

Value	Meaning
0	The buffer is of type **SERVER_XPORT_INFO_0_CONTAINER**.
1	The buffer is of type **SERVER_XPORT_INFO_1_CONTAINER**.

Buffer: A pointer to the TRANSPORT_INFO union that contains information about the transport. The value of the *Level* parameter determines the type of the contents of the *Buffer* parameter, as the previous table shows.

Return Values: The return value MUST be set as defined in the following table.<186>

Return code	Description
0x00000000	The client request succeeded.

In response to **NetrServerTransportDelEx** message, the server MUST unbind (or disconnect) the specified transport protocol from the server, or return an error code. This effectively means that the SMB file server can no longer communicate with clients using the specified transport protocol (such as TCP).<187>

The *Level* parameter determines the type of structure the client has used to specify information about the new transport. Valid values are 0 and 1. If the *Level* parameter is not equal to one of the valid values, the server MUST fail the call with an implementation-specific error code.

If the *Level* parameter is 0, the Buffer parameter points to a **SERVER_TRANSPORT_INFO_0** structure. If the Level parameter is 1, the Buffer parameter points to a **SERVER_TRANSPORT_INFO_1** structure.

The server MUST validate all information that is provided in the **SERVER_TRANSPORT_INFO** structure in an implementation-specific manner, and, if any member of the structure is found to be invalid, the server MUST fail the call with an implementation-specific error code.

The server SHOULD<188> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the caller does not have the required credentials, the server SHOULD<189> fail the call.

3.1.4.27 NetrpGetFileSecurity (Opnum 39)

The **NetrpGetFileSecurity** method returns to the caller a copy of the security descriptor that protects a file or directory.

```
DWORD NetrpGetFileSecurity(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string, unique] WCHAR* ShareName,
  [in, string] WCHAR* lpFileName,
  [in] SECURITY_INFORMATION RequestedInformation,
  [out] PADT_SECURITY_DESCRIPTOR* SecurityDescriptor
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

ShareName: A pointer to a null-terminated Unicode UTF-16 string that specifies the share name on which the file is found.

lpFileName: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the file or directory whose security is being retrieved. The name MUST specify the full path to the file from the ShareName.

RequestedInformation: The type of security information being requested, as specified in [MS-DTYP] section **2.4.7**.

SecurityDescriptor: A pointer to a **PADT_SECURITY_DESCRIPTOR** structure, as specified in section 2.2.4.100, where the desired information is returned.

Return Values: The return value MUST be set as specified in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrpGetFileSecurity** message, the server MUST return to the caller a copy of the security descriptor that protects a file or directory, or return an error code. The security descriptor is always returned in the self-relative format.

The *ShareName* parameter specifies a local share name on the server. The server MUST expand the share name to find the local path that is associated with it. The server MUST then combine the local path for the share name with the *lpFileName* parameter, in order to create a fully qualified path name that is local to the server. The server MUST then obtain the security descriptor with the information that the client requires, as specified in the *RequestedInformation* parameter, for the

local file that the path name obtained specifies, and return it to the client in the out parameter *SecurityDescriptor*. The security descriptor itself is stored in the Buffer member of the *SecurityDescriptor* parameter; the length of the security descriptor is stored in the Length member.

The server SHOULD<190> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the caller does not have the required credentials, the server SHOULD<191> fail the call.

3.1.4.28 NetrpSetFileSecurity (Opnum 40)

The **NetrpSetFileSecurity** method sets the security of a file or directory.

```
DWORD NetrpSetFileSecurity(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string, unique] WCHAR* ShareName,
  [in, string] WCHAR* lpFileName,
  [in] SECURITY_INFORMATION SecurityInformation,
  [in] PADT_SECURITY_DESCRIPTOR SecurityDescriptor
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

ShareName: A pointer to a null-terminated Unicode UTF-16 string that specifies the share name on which the file is found.

lpFileName: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of the file or directory whose security is being set.

SecurityInformation: The type of security information being set, as specified in [MS-DTYP] section **2.4.7**.

SecurityDescriptor: A pointer to a **PADT_SECURITY_DESCRIPTOR** structure, which provides the security descriptor to set.

Return Values: The return value MUST be set as specified in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a NetrpSetFileSecurity message, the server MUST set the security descriptor of the specified file or directory on the server, or return an error code.

The ShareName parameter specifies a local share name on the server. The server MUST expand the share name to find the local path that is associated with it. The server MUST then combine the local path for the share name with the *lpFileName* parameter to create a fully qualified path name that is local to the server.

The *SecurityDescriptor* parameter has a Buffer member that contains a security descriptor in self-relative format and a Length member that specifies the length, in bytes, of the Buffer member. The server MUST apply the descriptor in the Buffer to the local file, whose PathName was computed as

previously specified, by combining the local path that corresponds to the ShareName parameter and the *lpFileName* parameter.

The server SHOULD<192> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the caller does not have the required credentials, the server SHOULD<193> fail the call.

3.1.4.29 NetprPathType (Opnum 30)

The **NetprPathType** method checks a path name to determine its type.

```
NET_API_STATUS NetprPathType(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* PathName,
  [out] DWORD* PathType,
  [in] DWORD Flags
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

PathName: A pointer to a null-terminated Unicode UTF-16 string that specifies the path name to check.

PathType: A path type is returned. It MUST be one of the values that are defined in section 2.2.2.9.

Flags: A bitmask that MUST contain the bitwise OR of zero or more of the following values specifying controlling flags.

Value	Meaning
0x00000001	If set, the method uses old-style path rules (128-byte paths, 8.3 components) when validating the path. This flag gets set automatically on DOS and OS/2 1.1 systems.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetprPathType** message, the server MUST parse the specified path, determining if it is a valid path and determining its path type, or return an error code. Path type values are defined in section PathTypes.

The *PathName* parameter specifies the path name whose type needs to be determined.

If the *Flag* parameter is 0x1, the server MUST use old (MS-DOS) style path name rules that state that a path name can be 128 bytes long, and the file portion of the path has an 8-bit name 3-bit extension. If the value of the *Flag* parameter is 0x0, the server MUST use the modern Windows path rules.

The server MUST verify that *PathName* is a valid path name and that *Flag* is either 0 or 0x1. If either parameter is not valid, the server MUST fail the call with an implementation-specific<194> error code.

The server then MUST determine the type of the path that is specified by *PathName* in an implementation-specific manner and return it in the PathType parameter. Valid return codes for the PathType parameter are listed in section 2.2.2.9.

If the server cannot determine the type of the path that is specified by *PathName*, it MUST fail the call with an implementation-specific<195> error code.

The server MAY<196> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<197> fail the call.

3.1.4.30 NetprPathCanonicalize (Opnum 31)

The **NetprPathCanonicalize** method converts a path name to an implementation-specific canonical format.

```
NET_API_STATUS NetprPathCanonicalize(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* PathName,
  [out, size_is(OutbufLen)] unsigned char* Outbuf,
  [in, range(0,64000)] DWORD OutbufLen,
  [in, string] WCHAR* Prefix,
  [in, out] DWORD* PathType,
  [in] DWORD Flags
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

PathName: A pointer to a null-terminated Unicode UTF-16 string that specifies the path name to canonicalize.

Outbuf: A pointer to the output Buffer where the canonicalized path name is returned.

OutbufLen: Length of output Buffer Outbuf in bytes. The value of this field MUST be within the range 0-64000, inclusive.

Prefix: A pointer to a null-terminated Unicode UTF-16 string that specifies an optional prefix to use when canonicalizing a relative pathname.

PathType: Place to store the path type. If this parameter is not zero, the server assumes that the *PathType* parameter has already been determined by a previous call to NetprPathType. This parameter MUST be one of the values that is defined in Path Types.

Flags: Reserved, MUST be zero.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetprPathCanonicalize** message, the server MUST compute the implementation-specific canonical version of the specified path name, or return an error code.

The *PathName* parameter specifies the path name that needs to be canonicalized.

The *PathType* parameter, if nonzero, MUST specify the path type of the path that is specified by the *PathName* parameter by a previous successful call to NetprPathType. Even if it is set to the correct nonzero value by the client, the server MAY change it because the canonicalized version of a name may be of a different type than the original version. If *PathType* is zero, the server MUST validate and get the type of *PathName* (as specified in **NetprPathType**) first. If this fails, the server MUST fail the call.

The *Prefix* parameter, if it is a non-empty string, specifies a path component that MUST be prefixed to *PathName* to get the full-path to canonicalize. The server MUST treat Prefix as a *PathName* and validate and get the type of Prefix in the same way as for *PathName*. If this fails, the server MUST fail the call.

The *OutBufLen* parameter specifies the length of the output buffer OutBuf provided by the client. If the length of the canonicalized path name is greater than *OutBufLen*, the server MUST fail the call with an implementation-specific error code.

The server MUST construct the path to canonicalize by combining the Prefix and *PathName* in an implementation-specific manner and then MUST canonicalize the resultant path. The canonicalization process is implementation dependent.<198>

After the canonicalization is successfully finished, the server MUST determine the path type of the canonicalized path name, as specified in **NetprPathType**, and store the result in the *PathType* parameter. Valid return codes for the PathType parameter are as specified in Path Types. If this fails, the server MUST fail the call.

The server MAY<199> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<200> fail the call.

3.1.4.31 NetprPathCompare (Opnum 32)

The **NetprPathCompare** method performs an implementation-specific comparison of two paths.

```
long NetprPathCompare(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* PathName1,
  [in, string] WCHAR* PathName2,
  [in] DWORD PathType,
  [in] DWORD Flags
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

PathName1: A pointer to a null-terminated Unicode UTF-16 string containing the first PathName to compare.

PathName2: A pointer to a null-terminated Unicode UTF-16 string containing the second PathName to compare.

PathType: Type of Pathnames, as specified in section 2.2.2.9.

Flags: Bitmask that MUST contain the bitwise 'OR' of zero or more of the following values specifying controlling flags.

Value	Meaning
0x00000001	SHOULD be set if both of the paths have already been canonicalized.

Return Values: MUST return 0 if both paths are same, -1 if first is less than the second and 1 otherwise.

In response to a **NetprPathCompare** message, the server MUST compare the two paths that are specified as parameters to see if they match, and return this result or return an error code. If the supplied names are not canonicalized, the server MUST do the canonicalization of the path names before a comparison can occur. This does not modify the input path names. The clients SHOULD call this method with canonicalized path names only, because the canonicalization operation can be expensive. If uncanonicalized path names are passed in, the caller SHOULD be aware that a nonzero result could be due to an error that occurred during canonicalization.

The *PathName1* parameter and *PathName2* parameter specify the two path names to be compared.

The *Flags* parameter MUST be either 0 or 1. If the *Flags* parameter has any other value, the server MUST fail the call. If the *Flags* parameter is 1, it implies that the specified path names are already canonicalized and the server MUST not try to canonicalize them.

Any combination of Name1 (canonicalized or not), Name2 (canonicalized or not), and *Flags* (0 or 1) is valid.

If *Flags* is set to 0, the server MUST first attempt to canonicalize both Name1 and Name2 (and MUST respond with an error if canonicalization fails) before comparing the names.

If *Flags* is set to 1, the server MUST compare the names without first attempting canonicalization. Using Flags=1 could optimize performance because it eliminates the need for the server to repeatedly canonicalize a path name if it is being compared multiple times. If the *Flags* parameter does not have a valid value, the server SHOULD fail the call with an implementation-specific error code.

If the *Flags* parameter is 1, the *PathType* parameter MUST specify the path type for the two path names. Valid values for the *PathType* parameter are as specified in section 2.2.2.9. If the *PathType* parameter does not have a valid value, the server MAY<201> fail the call.

If the *Flags* parameter is 0, the server MUST canonicalize the specified path names and obtain their *PathTypes* first, as specified in section 3.1.4.30. If this fails, the server MUST fail the call. If the *PathTypes* for the two path names thus obtained are different, the server MUST return 1.

The server then compares the canonicalized path names by using an implementation-specific<202> comparison and MUST return 0 to the caller if the paths match, -1 if *PathName1* is less than *PathName2*, and 1 if *PathName1* is greater than *PathName2*.

The server MAY<203> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<204> fail the call.

3.1.4.32 NetprNameValidate (Opnum 33)

The **NetprNameValidate** method performs implementation-specific checks to ensure that the specified name is a valid name for the specified type.

```
NET_API_STATUS NetprNameValidate(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* Name,
  [in] DWORD NameType,
  [in] DWORD Flags
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Name: A pointer to a null-terminated Unicode UTF-16 string that specifies the name to check.

NameType: The type of Name. It MUST be one of the values defined in section 2.2.2.8.

Flags: Reserved, MUST be zero.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetprNameValidate** message, the server MUST validate the specified LANMAN, as specified in [CIFS], object name for character set and length, or return an error code. For rules about what defines a valid name for a particular *NameType*, see [CIFS].

The *NameType* parameter determines what validation is done on the name that is specified by the Name parameter. Valid values for the *NameType* parameter are as specified in section 2.2.2.8. If the *NameType* parameter does not have a valid value, the server MUST fail the call with an implementation-specific error code.

The value of *NameType* identifies the minimum and maximum lengths for a particular *NameType* and the characters that are permitted in a name for that *NameType*. The server MUST validate the specified name by being sure that its length is within the minimum and maximum lengths for its type and that there are no characters in its name that are invalid for it type. If any of these checks fail, the server MUST fail the call with an implementation-specific error code.

The server MAY<205> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<206> fail the call.

3.1.4.33 NetprNameCanonicalize (Opnum 34)

The **NetprNameCanonicalize** method converts a name into an implementation-specific canonical format for the specified type.

```
NET_API_STATUS NetprNameCanonicalize(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* Name,
  [out, size_is(OutbufLen)] WCHAR* Outbuf,
  [in, range(0,64000)] DWORD OutbufLen,
  [in] DWORD NameType,
  [in] DWORD Flags
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Name: A pointer to a null-terminated Unicode UTF-16 string specifying the name to canonicalize.

Outbuf: A pointer to a null-terminated Unicode UTF-16 string that is the buffer where the canonicalized name is returned.

OutbufLen: The length of output buffer Outbuf. The value of this field MUST be within the range 0 through 64000, inclusive.

NameType: The type of Name, as specified in 2.2.2.8.

Flags: A bitmask that MUST contain the bitwise 'OR' of zero or more of the following values specifying controlling flags.

Value	Meaning
0x80000000	LM2.x compatible name canonicalization.
0x00000001	If set, the method requires the length of the output buffer to be sufficient to hold any name of the specified type. Otherwise, the buffer length only needs to be large enough to hold the canonicalized version of the input name that is specified in this invocation of the method.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetprNameCanonicalize** message, the server MUST convert a LANMAN, as specified in [CIFS], object name to the canonical form, or return an error code.

The *NameType* parameter determines what needs to be done on the name that is specified by the *Name* parameter to convert it to a canonical format. Valid values for the *NameType* parameter are

as specified in Name Types. If the *NameType* parameter does not have a valid value, the server MUST fail the call with an implementation-specific error code.

The *Flags* parameter is a bitmask that specifies certain controlling flags that affect how the server processes this message. The valid bits are 0x80000000 and 0x1. If any other bit is set, the server MUST fail the call with an implementation-specific error code.

If (*Flags* & 0x80000000) is true, it implies the server MUST perform a LAN Manager version 2.x compatible canonicalization. Some *NameTypes* have different rules on how a canonical name for those types on LAN Manager version 2.x is defined. For more information, see [CIFS].

The server MUST use the *NameType* parameter to determine the maximum length of any name for that type. If (Flags & 0x1) is true and the length of the output buffer specified by the *OutBufLen* parameter is not greater than or equal to the maximum length of any name for that type, the server MUST fail the call with an implementation-specific error code.

The server MUST validate the *Name* as specified NetprNameValidate to be sure it is a valid name of type *NameType*. If this fails, the server MUST fail the call. Next, the server converts the *Name* to an implementation-specific<207> canonical format and stores the output in the OutBuf parameter.

The server MAY<208> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<209> fail the call.

3.1.4.34 NetprNameCompare (Opnum 35)

The **NetprNameCompare** method does an implementation-defined comparison of two names of a specific name type.

```
long NetprNameCompare(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* Name1,
  [in, string] WCHAR* Name2,
  [in] DWORD NameType,
  [in] DWORD Flags
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Name1: A pointer to a null-terminated Unicode UTF-16 string containing the first Name to compare.

Name2: A pointer to a null-terminated Unicode UTF-16 string containing the second Name to compare.

NameType: The type of Names, as specified in section 2.2.2.8.

Flags: A bitmask that MUST contain the bitwise 'OR' of zero or more of the following values specifying controlling flags.

Value	Meaning
0x80000000	Enable LM2.x compatibility.

Value	Meaning
0x00000001	SHOULD be set if both of the names have already been canonicalized (by using NetprNameCanonicalize).

Return Values: MUST return 0 if both paths are the same. Other values indicate either the paths are different or there was an error processing the client request.

In response to a **NetprNameCompare** message, the server MUST compare two LANMAN, as specified in [CIFS], object names to see if they are the same, or return an error code. If the supplied names are not canonicalized, the server MUST do the canonicalization of the names. The server does not do name validation. It assumes that the two names have been validated separately.

The *Name1* parameter and *Name2* parameter specify the two names to be compared.

The *Flags* parameter is a bitmask that specifies certain controlling flags that affect how the server processes this message. The valid bits are 0x80000000 and 0x1. If any other bit is set, the server MUST fail the call with an implementation-specific error code.

If (*Flags* & 0x80000000) is true, it implies the server MUST enable LAN Manager version 2.x compatibility. This implies that the rules that are used for comparison and canonicalization (if needed) MUST be those defined for LAN Manager version 2.x. For more information, see [CIFS].

If (*Flags* & 0x1) is true, the names specified by *Name1* and *Name2* are already canonicalized, and the *NameType* parameter MUST specify the name type for the two names. Valid values for the *NameType* parameter are listed in section Name Types. If the *NameType* parameter does not have a valid value, the server MAY fail the call.

If (*Flags* & 0x1) is not true, the server MUST canonicalize the specified names and obtain their name types first as specified in NetprNameCanonicalize. If this fails, the server MUST fail the call.

The server MUST compare the canonicalized version of the names if the names were not already canonicalized or the original names otherwise and MUST return 0 if both are same, -1 if *Name1* is less than *Name2*, and 1 if *Name1* is greater than *Name2*. The comparison is implementation-specific.<210>

The server MAY<211> enforce appropriate security measures to be sure that the server enforces these security measures and caller has the required permissions to execute this call. If the caller does not have the required credentials, the server SHOULD<212> fail the call.

3.1.4.35 NetrDfsGetVersion (Opnum 43)

The **NetrDfsGetVersion** method checks if the server is a DFS server and if so returns an implementation-specific DFS version.

```
NET_API_STATUS NetrDfsGetVersion(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [out] DWORD* Version
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Version: A pointer to a DWORD where the server returns the DFS version.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrDfsGetVersion** message, the server MUST return the version number of DFS in use on the server, if the server supports DFS, or MUST return an error code. An implementation MAY<213> choose to perform no processing and return an implementation-specific error code when this method is called.

The *Version* parameter is a pointer to a DWORD. If the server supports DFS, the server MUST set this parameter to an implementation-specific<214> DFS version number that the server supports.

The server MAY<215> enforce appropriate security measures to be sure that the server enforces these security measures and caller has the required permissions to execute this call. If the caller does not have the required credentials, the server SHOULD<216> fail the call.

3.1.4.36 NetrDfsCreateLocalPartition (Opnum 44)

The **NetrDfsCreateLocalPartition** method marks a share as being a DFS share. In addition, if RelationInfo is non-NULL, it creates DFS links, as specified in [MS-DFSC], for each of the entries in RelationInfo.

```
NET_API_STATUS NetrDfsCreateLocalPartition(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* ShareName,
  [in] GUID* EntryUid,
  [in, string] WCHAR* EntryPrefix,
  [in, string] WCHAR* ShortName,
  [in] LPNET_DFS_ENTRY_ID_CONTAINER RelationInfo,
  [in] int Force
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

ShareName: A pointer to a null-terminated Unicode UTF-16 string that specifies the name of a local disk share on the server to add to DFS.

EntryUid: A pointer to a **GUID** type that specifies the GUID for this DFS share. The GUID for this share MUST NOT match a GUID for an existing local partition.<217>

EntryPrefix: A pointer to a null-terminated Unicode UTF-16 string that specifies the path of the DFS share.

ShortName: A pointer to a null-terminated Unicode UTF-16 string that specifies the shortname version (8.3 format) of EntryPrefix.

RelationInfo: A pointer to a **NET_DFS_ENTRY_ID_CONTAINER** structure, as specified in section 2.2.4.102. Specifies the DFS child links that are under the DFS share specified by "EntryPrefix".

Force: The *Force* parameter is ignored and SHOULD be set to zero.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrDfsCreateLocalPartition** message, the server MUST mark an existing SMB file share as a DFS share enabling it to be accessed using DFS, as specified in [MS-DFSC], if the server supports DFS, or return an error code.<218>

The *ShareName* parameter MUST specify the name of an existing SMB file share of type STYPE_DISKTREE (for more information, see Share Types) or the server MUST fail the call with an implementation-specific error code.

The *EntryUid* parameter specifies the GUID that the server MUST assign to the new DFS share.

This parameter MUST NOT be NULL, or the server MUST fail the call with an implementation-specific error code. If the *EntryUid* matches a GUID for an existing local partition, then the server MUST fail the call with an implementation-specific error code.

The EntryPrefix parameter specifies the path of the DFS share. This string MUST be in one of following two forms. The first form is \Dfsname\sharename, where Dfsname is the name of the storage server that hosts the root of a stand-alone DFS implementation; sharename is the name of a shared folder that is published on the DFS host server. The second form is \DomainName\DomDfsname, where DomainName is the name of the domain that hosts the DFS root; DomDfsname is the name of the root of a domain-based DFS implementation that is published in the domain's directory service.

The RelationInfo parameter specifies the DFS child links to create under the share that is specified by EntryPrefix. It has a member count that specifies the number of child links and a Buffer member that is an array of 'Count' structure of type **NET_DFS_ENTRY_ID**. A DFS child link MUST be created for each entry in the Buffer. The RelationInfo parameter MUST not be NULL, or the server MUST fail the call with an implementation-specific error code.

The ShortName parameter specifies old-style 8.3 format name for the share that is specified by EntryPrefix and MUST be interpreted by the server in an implementation-specific manner.<219>

The Force parameter is ignored and MUST be zero.

The server MAY<220> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<221> fail the call.

3.1.4.37 NetrDfsDeleteLocalPartition (Opnum 45)

The **NetrDfsDeleteLocalPartition** method deletes a DFS share (Prefix) on the server.

```
NET_API_STATUS NetrDfsDeleteLocalPartition(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] GUID* Uid,
  [in, string] WCHAR* Prefix
```

```
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Uid: Specifies the GUID of the DFS share to delete. This GUID MUST be obtained using NetrDfsGetInfo (opnum 4) as specified in section 3.2.5.1.6 of [MS-DFSNM].

Prefix: A pointer to a null-terminated Unicode UTF-16 string that contains the path to the DFS share.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrDfsDeleteLocalPartition** message, the server MUST delete a DFS share, or return an error code. An implementation MAY<222> choose to perform no processing and return an implementation-specific error code when this method is called.

The Prefix parameter specifies the path of the DFS share to delete. This string MUST be in one of following two forms. The first form is \Dfsname\sharename, where Dfsname is the name of the storage server that hosts the root of a stand-alone DFS implementation; sharename is the name of a shared folder that is published on the DFS host server. The second form is \DomainName\DomDfsname, where DomainName is the name of the domain that hosts the DFS root; DomDfsname is the name of the root of a domain-based DFS implementation published in the domain's directory service.

If the server cannot find a DFS share whose GUID matches Uid and whose path matches Prefix, it MUST fail the call with an implementation-specific error code. If a matching share is found, the server deletes the share and returns 0.

The server MAY<223> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<224> fail the call.

3.1.4.38 NetrDfsSetLocalVolumeState (Opnum 46)

The **NetrDfsSetLocalVolumeState** method sets a local DFS share online or offline.

```
NET_API_STATUS NetrDfsSetLocalVolumeState(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] GUID* Uid,
  [in, string] WCHAR* Prefix,
  [in] unsigned long State
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Uid: Specifies the GUID of the DFS share. This GUID MUST be obtained using NetrDfsGetInfo (opnum 4) as specified in section 3.2.5.1.6 of [MS-DFSNM].

Prefix: A pointer to a null-terminated Unicode UTF-16 string that specifies the path to the DFS share.

State: A DWORD specifying the new state for the DFS share. To set the share to offline, the State parameter MUST be (0x80). The State parameter MUST be set to any other value to take the share online.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a NetrDfsSetLocalVolumeState message, the server MUST set the state of a local DFS share to online or offline, or return an error code. Marking a share state offline makes the share inaccessible over DFS. An implementation MAY<225> choose to perform no processing and return an implementation-specific error code when this method is called.

The Uid parameter specifies the GUID of the share whose state needs to be set.

The Prefix parameter specifies the path of the DFS share whose state needs to be set. This MUST refer to a local DFS share, or the server MUST fail the call.

The State parameter specifies whether the share state MUST be set to online or offline. If the value of State is 0x80, the share state MUST be set to offline. For any other value, the state MUST be set to online.

The server MAY<226> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<227> fail the call.

3.1.4.39 NetrDfsCreateExitPoint (Opnum 48)

The **NetrDfsCreateExitPoint** method creates a DFS link on the server.

```
NET_API_STATUS NetrDfsCreateExitPoint(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] GUID* Uid,
  [in, string] WCHAR* Prefix,
  [in] unsigned long Type,
  [in, range(0,32)] DWORD ShortPrefixLen,
  [out, size_is(ShortPrefixLen)]
    WCHAR* ShortPrefix
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Uid: Specifies the GUID for the DFS link. This GUID MUST be obtained using NetrDfsGetInfo (opnum 4) as specified in section 3.2.5.1.6 of [MS-DFSNM].

Prefix: A pointer to a null-terminated Unicode UTF-16 string that specifies the path of the DFS link.

Type: This parameter MUST be one of the values that is specified in DFS Entry Flags.

ShortPrefixLen: Specifies the length of ShortPrefix, in Unicode UTF-16 characters.

ShortPrefix: A pointer to the name of the **DFS namespace** root or link, stored as a string of Unicode characters. This MUST be a Unicode UTF-16 UNC path string with one leading backslash instead of the usual two, and without a null termination.<228>

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrDfsCreateExitPoint** message, the server MUST create a DFS link, as specified in [MS-DFSC], or return an error code.

An implementation MAY<229> choose to perform no processing and return an implementation-specific error code when this method is called.

The Uid parameter specifies the GUID to be assigned to the new link.

The Prefix parameter specifies the path of the DFS link. The string MUST be in one of two forms. The first form is \Dfsname\sharename\path_to_link, where Dfsname is the name of the storage server that hosts the root of a stand-alone DFS implementation; sharename is the name of a shared folder that is published on the DFS host server; and path_to_link specifies the path on the physical network share. The second form is \DomainName\DomDfsname\path_to_link, where DomainName is the name of the domain that hosts the DFS root; DomDfsname is the name of the root of a domain-based DFS implementation published in the domain's directory service; and path_to_link specifies the path on the physical network share.

The Type parameter specifies the type of the new link and MUST be one of the values listed in DFS Entry Flags. If the value of this parameter is PKT_ENTRY_TYPE_MACHINE, the server MUST fail the call.

The ShortPrefixLen parameter specifies the length of the ShortPrefix parameter that specifies a short name for the new link in the 8.3 format.

The server MAY<230> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<231> fail the call.

3.1.4.40 NetrDfsDeleteExitPoint (Opnum 49)

The **NetrDfsDeleteExitPoint** method deletes a DFS link on the server.

```
NET_API_STATUS NetrDfsDeleteExitPoint(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] GUID* Uid,
  [in, string] WCHAR* Prefix,
  [in] unsigned long Type
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Uid: Specifies the GUID that corresponds to the DFS link given by Prefix. This GUID MUST be obtained using NetrDfsGetInfo (opnum 4), as specified in section 3.2.5.1.6 of [MS-DFSNM].

Prefix: A pointer to a null-terminated Unicode UTF-16 string that specifies the path of the DFS link.

Type: This parameter MUST be one of the values listed in DFS Entry Flags.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a NetrDfsDeleteExitPoint message, the server MUST delete a DFS link, as specified in [MS-DFSC], or return an error code. An implementation MAY<232> choose to perform no processing and return an implementation-specific error code when this method is called.

The *Uid* parameter specifies the GUID of the link to delete.

The *Prefix* parameter specifies the path of the DFS link. The string MUST be in one of two forms. The first form is \Dfsname\sharename\path_to_link, where Dfsname is the name of the storage server that hosts the root of a standalone DFS implementation; sharename is the name of a shared folder that is published on the DFS host server; and path_to_link specifies the path on the physical network share. The second form is \DomainName\DomDfsname\path_to_link, where DomainName is the name of the domain that hosts the DFS root; DomDfsname is the name of the root of a domain-based DFS implementation published in the domain's directory service; and path_to_link specifies the path on the physical network share.

The *Type* parameter specifies the type of the link to delete and MUST be one of the values listed in DFS Entry Flags. If the value of this parameter is PKT_ENTRY_TYPE_MACHINE, the server MUST fail the call.

If a link whose GUID, path, and type match the specified parameters is present, the server MUST delete it; otherwise, it MUST fail the call with an implementation-specific error code.

The server MAY<233> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<234> fail the call.

3.1.4.41 NetrDfsModifyPrefix (Opnum 50)

The **NetrDfsModifyPrefix** method changes the path that corresponds to a DFS link on the server.

```
NET_API_STATUS NetrDfsModifyPrefix(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in] GUID* Uid,
  [in, string] WCHAR* Prefix
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

Uid: Specifies the GUID that corresponds to the DFS link that needs to be changed. This GUID MUST be obtained by using NetrDfsGetInfo (opnum 4), as specified in [MS-DFSNM] section 3.2.5.1.6.

Prefix: A pointer to a null-terminated Unicode UTF-16 string that specifies the path of the updated DFS link.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrDfsModifyPrefix** message, the server MUST update the path for a DFS link, or return an error code. This message is typically used by domain controllers (DCs) to fix a bad prefix match. An implementation MAY<235> choose to perform no processing and return an implementation-specific error code when this method is called.

The *Uid* parameter specifies the GUID that corresponds to the DFS link that needs to be changed.

The *Prefix* parameter specifies the path of the updated DFS link. The string MUST be in one of two forms. The first form is \Dfsname\sharename\path_to_link, where Dfsname is the name of the storage server that hosts the root of a standalone DFS implementation; sharename is the name of a shared folder that is published on the DFS host server; and path_to_link specifies the path on the physical network share. The second form is \DomainName\DomDfsname\path_to_link, where DomainName is the name of the domain that hosts the DFS root; DomDfsname is the name of the root of a domain-based DFS implementation that is published in the domain's directory service; and path_to_link specifies the path on the physical network share.

The server MAY<236> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<237> fail the call.

3.1.4.42 NetrDfsFixLocalVolume (Opnum 51)

The **NetrDfsFixLocalVolume** method adds knowledge of a new DFS share on the server.

```
NET_API_STATUS NetrDfsFixLocalVolume(
```

```
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, string] WCHAR* VolumeName,
  [in] unsigned long EntryType,
  [in] unsigned long ServiceType,
  [in, string] WCHAR* StgId,
  [in] GUID* EntryUid,
  [in, string] WCHAR* EntryPrefix,
  [in] LPNET_DFS_ENTRY_ID_CONTAINER RelationInfo,
  [in] unsigned long CreateDisposition
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

VolumeName: A pointer to a null-terminated Unicode UTF-16 string that specifies the target for the DFS root share. This MUST be local to the server and is in the form of an NT Path name, for example, \??\C:\DfsShare. This SHOULD not scope over a directory that is in DFS, and it SHOULD not be a child of a DFS share. If the volume name specified is invalid, the server SHOULD fail the call with an implementation-specific error code.

EntryType: This parameter MUST be one of the values listed in DFS Entry Flags. If the entry type specified is invalid, the server SHOULD fail the call with an implementation-specific error code.

ServiceType: This parameter MUST be a combination of one or more of the following values. If the service type specified is invalid, the server SHOULD fail the call with an implementation-specific error code.

Value	Meaning
DFS_SERVICE_TYPE_MASTER 0x00000001	Master service.
DFS_SERVICE_TYPE_READONLY 0x00000002	Read-only service.
DFS_SERVICE_TYPE_LOCAL 0x00000004	Local service.
DFS_SERVICE_TYPE_REFERRAL 0x00000008	Referral service.
DFS_SERVICE_TYPE_ACTIVE 0x000000010	Active service.
DFS_SERVICE_TYPE_DOWN_LEVEL 0x000000020	Downlevel service.
DFS_SERVICE_TYPE_COSTLIER 0x000000040	Costlier than previous.
DFS_SERVICE_TYPE_OFFLINE 0x000000080	Service is offline.

StgId: A pointer to a variable that specifies an ID for the local storage. The server MUST ignore the value passed in for the StgId parameter.

EntryUid: Specifies the GUID that corresponds to the DFS share. This GUID MUST be obtained using NetrDfsGetInfo (opnum 4) as specified in section 3.2.5.1.6 of [MS-DFSNM].

EntryPrefix: A pointer to a null-terminated Unicode UTF-16 string that specifies the path of the DFS share to be updated.

RelationInfo: A pointer to a **NET_DFS_ENTRY_ID_CONTAINER** structure as specified in 2.2.4.102. Specifies the DFS child links under the DFS share as specified by *EntryPrefix*.

CreateDisposition: Specifies what to do, depending on whether the share already exists. This field MUST be set to one of the following values.

Value	Meaning
FILE_SUPERSEDE 0x00000000	If the share already exists, replace it with the specified share. If it does not exist, create the specified share.
FILE_OPEN 0x00000001	If the share already exists, fail the request and do not create or open the specified share. If it does not exist, create the specified share.
FILE_CREATE 0x00000002	If the file already exists, open it instead of creating a new share. If it does not exist, fail the request and do not create a new share.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrDfsFixLocalVolume** message, the server MUST add the link name that corresponds to a specified Uid, or return an error code. This message is typically sent by a DC when it discovers that the server is completely unaware of a new DFS volume. An implementation MAY<238> choose to perform no processing and return an implementation-specific error code when this method is called.

The *VolumeName* parameter specifies the target for the DFS root share. This MUST be local to the server and is in the form of an NT path name, for example, \??\C:\DfsShare. This SHOULD not scope over a directory that is in DFS, and it SHOULD not be a child of a DFS share.

The *EntryType* parameter specifies the type of the link and MUST be one of the values listed in DFS Entry Flags.

The *ServiceType* parameter specifies the client's service type.

The *StgId* parameter specifies an implementation-specific ID for the local storage.

The *EntryUid* parameter specifies the GUID for the new link.

The *Prefix* parameter specifies the path of the updated DFS link. The string MUST be in one of two forms. The first form is \Dfsname\sharename\path_to_link, where Dfsname is the name of the storage server that hosts the root of a standalone DFS implementation; sharename is the name of a shared folder that is published on the DFS host server; and path_to_link specifies the path on the

physical network share. The second form is \DomainName\DomDfsname\path_to_link, where DomainName is the name of the domain that hosts the DFS root; DomDfsname is the name of the root of a domain-based DFS implementation that is published in the domain's directory service; and path_to_link specifies the path on the physical network share.

The parameter RelationInfo specifies the DFS child links under the DFS share specified by "EntryPrefix". If this parameter is NULL the server MUST fail the call.<239>

The parameter CreateDisposition specifies what MUST happen if a share with the path EntryPrefix already exists.

The server MAY<240> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<241> fail the call.

3.1.4.43 NetrDfsManagerReportSiteInfo (Opnum 52)

The **NetrDfsManagerReportSiteInfo** method obtains Active Directory site information.

```
NET_API_STATUS NetrDfsManagerReportSiteInfo(
  [in, string, unique] SRVSVC_HANDLE ServerName,
  [in, out, unique] LPDFS_SITELIST_INFO* ppSiteInfo
);
```

ServerName: An **SRVC_HANDLE**, as specified in section 2.2.1.1, that indicates the remote server on which the method MUST execute. The *ServerName* parameter MUST be ignored by the server when processing any message.

ppSiteInfo: A pointer to an **LPDFS_SITELIST_INFO** field, which in turn points to the location of a **DFS_SITELIST_INFO** structure in which the information is returned.

Return Values: The return value MUST be set as defined in the following table.

Return code	Description
0x00000000	The client request succeeded.
Other values	The client request failed.

In response to a **NetrDfsManagerReportSiteInfo** message, the server MUST return the list of Active Directory sites that this server manages, or return an error code. An implementation MAY<242> choose to perform no processing and return an implementation-specific error code when this method is called.

The *ppSiteInfo* parameter is a pointer to a **LPDFS_SITELIST_INFO** member, which in turn points to the location of a **DFS_SITELIST_INFO** structure in which the information is returned. That structure has a cSites member that the server SHOULD set to the number of sites returned. The information about the sites themselves MUST be returned in the Site member, which is an array of **DFS_SITENAME_INFO** structures. The sites the server returns are implementation-specific.<243>

The server MAY<244> enforce appropriate security measures to be sure that the caller has the required permissions to execute this call. If the server enforces these security measures and the caller does not have the required credentials, the server SHOULD<245> fail the call.

3.1.5 Timer Events

No protocol timer events are required on the client beyond the timers that are required in the underlying RPC transport.

3.1.6 Other Local Events

There are no local events used on the server beyond the events that are maintained in the underlying RPC transport.

3.2 Client Details

3.2.1 Abstract Data Model

No abstract data model is used.

3.2.2 Timers

No protocol timers are required beyond those internal ones that are used in RPC to implement resiliency to network outages. For more information, see [MS-RPCE].

3.2.3 Initialization

The client MUST create an RPC connection to the remote computer, as specified in section 2.1.

3.2.4 Message Processing Events and Sequencing Rules

Upon the completion of the RPC method, the client MUST return the result unmodified to the higher layer. This is a stateless protocol with the exception of the **NetrShareDelCommit** method.

No sequence of method calls is imposed on this protocol, with the following exceptions:

1. **NetrShareDelCommit** method: The first phase MUST be completed (by **NetrShareDelStart**) before the second phase is attempted.
2. **NetrFileGetInfo** method: **NetrFileEnum** MUST be called to obtain the *FileId* before the **NetrFileGetInfo** method is called.
3. **NetrFileClose** method: **NetrFileEnum** MUST be called to obtain the *FileId* before the **NetrFileClose** method is called.

When a method completes, the values that the RPC returns MUST be returned unmodified to the upper layer.

The client MUST ignore errors returned from the RPC server and notify the application invoker about the error that was received in the higher layer. Otherwise, no special message processing is required on the client beyond the processing that is required in the underlying RPC protocol.

3.2.5 Timer Events

There are no timer events.

3.2.6 Other Local Events

There are no additional local events.

4 Protocol Examples

For most methods, this is a simple request-response protocol. For every method that the server receives, except the NetrShareDelStart method and the NetrShareDelCommit method, it executes the method and returns a completion. The client simply returns the completion status to the caller.

For example, the client calls the NetrShareAdd method, and the server executes the method and returns NERR_Success.

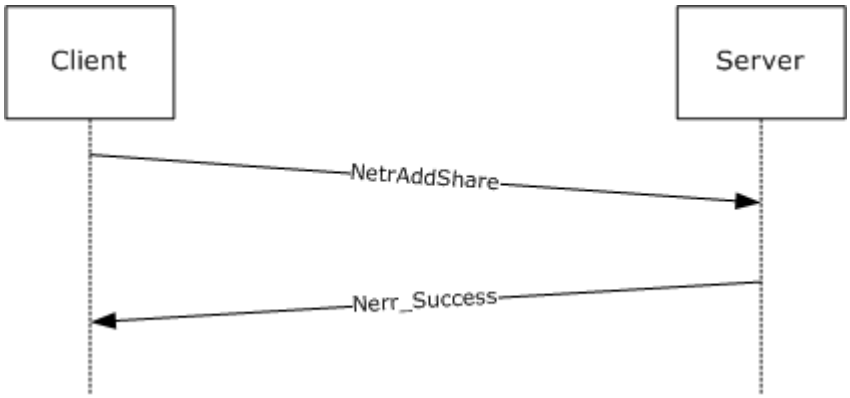

Figure 1: A simple request-response example

The following sections include two more examples of operations as used in common scenarios to illustrate the function of the Server Service Remote Protocol.

4.1 Example of ResumeHandle

The client calls the NetrFileEnum method to enumerate all open files on a server named "wingtiptoys". There are five open files on server "wingtiptoys".

The client calls the NetrFileEnum method with *ServerName* equal to "wingtiptoys", and the *Level* field of the **FILE_ENUM_STRUCT** structure that is passed in the *InfoStruct* parameter is set to 0x00000003. The client also sets the *PreferredMaximumLength* parameter to 0x00000100 and passes a non-NULL pointer in the *TotalEntries* parameter and the *ResumeHandle* parameter.

If, for example, only the information for the first two open files fits into 0x00000100 bytes, when the server receives this method, it executes the method locally and returns ERROR_MORE_DATA. The server returns the information for the first two open files in the *InfoStruct* parameter. It also sets the value of *TotalEntries* to 0x00000005 and the value of *ResumeHandle* to 0x00000120. The value of *ResumeHandle* is implementation-specific.

To continue enumerating the open files, the client calls the NetrFileEnum method with *ServerName* equal to "wingtiptoys", and the *Level* field of the **FILE_ENUM_STRUCT** structure that is passed in the *InfoStruct* parameter is set to 0x00000003. The client also sets the *PreferredMaximumLength* parameter to MAX_PREFERRED_LENGTH and passes a non-NULL pointer as *TotalEntries*. The client also passes the unchanged value of *ResumeHandle* (0x000000120).

On receiving this method, the server executes the method locally to continue enumeration based on a *ResumeHandle* value of 0x00000120 and returns ERROR_SUCCESS. The server returns the names of the next three open files in the *InfoStruct* parameter. It also sets the value of *TotalEntries* to 0x00000003. The value of *ResumeHandle* is irrelevant.

4.2 Two-Phase Share Deletion

The following figure shows the protocol message sequence for a two-phase share deletion.

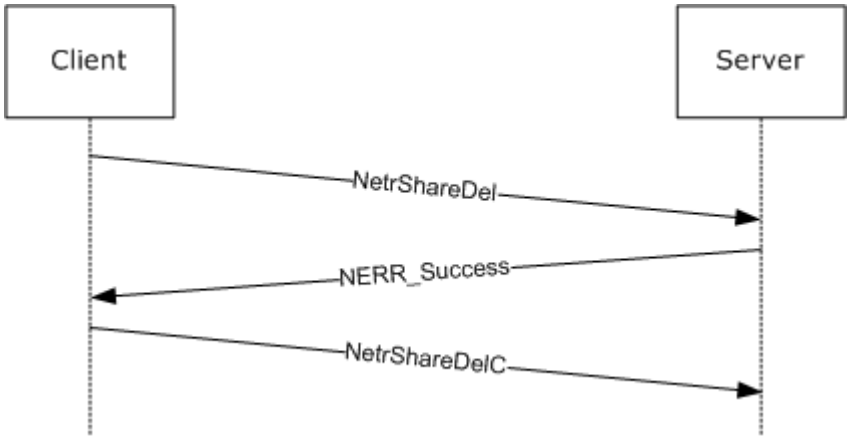

Figure 2: Two-phase share deletion

If the IPC$ share is being deleted, a two-phase delete MUST be performed because this action deletes the means of communication between the client and the server. The following is the sequence of messages for a two-phase share delete:

1. The client sends the NetrShareDelStart method to the server.
2. The server processes the first phase of the delete and returns the status NERR_Success.
3. The client sends the NetrShareDelCommit method to the server.
4. The server processes the second phase of delete. Because the communication channel between the client and the server is deleted, the client does not receive a status that indicates the successful completion of the NetrShareDelCommit method.

5 Security

The following sections specify security considerations for implementers of the Server Service Remote Protocol.

5.1 Security Considerations for Implementers

This protocol allows any user to connect to the server; therefore, any security bug in the server implementation could be exploitable. The server implementation SHOULD enforce security on each method.

5.2 Index of Security Parameters

This protocol allows any user to establish a connection to the RPC server as specified in section 2.1.

6 Appendix A: Full IDL

For ease of implementation, the full **IDL** is provided, where "ms-dtyp.idl" is the IDL as specified in [MS-DTYP] Appendix A.

```
import "ms-dtyp.idl";

[
    uuid(4B324FC8-1670-01D3-1278-5A47BF6EE188),
    version(3.0),
    ms_union,
    pointer_default(unique)
]
interface srvsvc
{
    typedef [handle, string] wchar_t * SRVSVC_HANDLE;

    typedef struct _CONNECTION_INFO_0
    {
        DWORD coni0_id;
    } CONNECTION_INFO_0,
     *PCONNECTION_INFO_0,
     *LPCONNECTION_INFO_0;

    typedef struct _CONNECT_INFO_0_CONTAINER
    {
        DWORD EntriesRead;
        [size_is(EntriesRead)] LPCONNECTION_INFO_0 Buffer;
    } CONNECT_INFO_0_CONTAINER,
     *PCONNECT_INFO_0_CONTAINER,
     *LPCONNECT_INFO_0_CONTAINER;

    typedef struct _CONNECTION_INFO_1
    {
        DWORD coni1_id;
        DWORD coni1_type;
        DWORD coni1_num_opens;
        DWORD coni1_num_users;
        DWORD coni1_time;
        [string] wchar_t * coni1_username;
        [string] wchar_t * coni1_netname;
    } CONNECTION_INFO_1,
     *PCONNECTION_INFO_1,
     *LPCONNECTION_INFO_1;

    typedef struct _CONNECT_INFO_1_CONTAINER
    {
        DWORD EntriesRead;
        [size_is(EntriesRead)] LPCONNECTION_INFO_1 Buffer;
    } CONNECT_INFO_1_CONTAINER,
     *PCONNECT_INFO_1_CONTAINER,
     *LPCONNECT_INFO_1_CONTAINER;

    typedef [switch_type(DWORD)] union _CONNECT_ENUM_UNION {
        [case(0)]
           CONNECT_INFO_0_CONTAINER* Level0;
```

```
    [case(1)]
       CONNECT_INFO_1_CONTAINER* Level1;
} CONNECT_ENUM_UNION;

typedef struct _CONNECT_ENUM_STRUCT {
    DWORD Level;
    [switch_is(Level)] CONNECT_ENUM_UNION ConnectInfo;
} CONNECT_ENUM_STRUCT,
  *PCONNECT_ENUM_STRUCT,
  *LPCONNECT_ENUM_STRUCT;

typedef struct _FILE_INFO_2
{
    DWORD fi2_id;
} FILE_INFO_2, *PFILE_INFO_2, *LPFILE_INFO_2;

typedef struct _FILE_INFO_2_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPFILE_INFO_2 Buffer;
} FILE_INFO_2_CONTAINER,
 *PFILE_INFO_2_CONTAINER,
 *LPFILE_INFO_2_CONTAINER;

typedef struct _FILE_INFO_3 {
    DWORD fi3_id;
    DWORD fi3_permissions;
    DWORD fi3_num_locks;
    [string] wchar_t * fi3_pathname;
    [string] wchar_t * fi3_username;
} FILE_INFO_3,
 *PFILE_INFO_3,
 *LPFILE_INFO_3;

typedef struct _FILE_INFO_3_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPFILE_INFO_3 Buffer;
} FILE_INFO_3_CONTAINER,
 *PFILE_INFO_3_CONTAINER,
 *LPFILE_INFO_3_CONTAINER;

typedef [switch_type(DWORD)] union _FILE_ENUM_UNION {
    [case(2)]
       FILE_INFO_2_CONTAINER* Level2;
    [case(3)]
       FILE_INFO_3_CONTAINER* Level3;
} FILE_ENUM_UNION;

typedef struct _FILE_ENUM_STRUCT {
    DWORD Level;
    [switch_is(Level)] FILE_ENUM_UNION FileInfo;
} FILE_ENUM_STRUCT,
 *PFILE_ENUM_STRUCT,
 *LPFILE_ENUM_STRUCT;

typedef [switch_type(unsigned long)] union _FILE_INFO
{
```

```
    [case(2)]
        LPFILE_INFO_2 FileInfo2;
    [case(3)]
        LPFILE_INFO_3 FileInfo3;
} FILE_INFO,
 *PFILE_INFO,
*LPFILE_INFO;

typedef struct _SESSION_INFO_0
{
    [string] wchar_t * sesi0_cname;
} SESSION_INFO_0,
 *PSESSION_INFO_0,
 *LPSESSION_INFO_0;

typedef struct _SESSION_INFO_0_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSESSION_INFO_0 Buffer;
} SESSION_INFO_0_CONTAINER,
 *PSESSION_INFO_0_CONTAINER,
 *LPSESSION_INFO_0_CONTAINER;

typedef struct _SESSION_INFO_1
{
    [string] wchar_t * sesi1_cname;
    [string] wchar_t * sesi1_username;
    DWORD sesi1_num_opens;
    DWORD sesi1_time;
    DWORD sesi1_idle_time;
    DWORD sesi1_user_flags;
} SESSION_INFO_1,
 *PSESSION_INFO_1,
 *LPSESSION_INFO_1;

typedef struct _SESSION_INFO_1_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSESSION_INFO_1 Buffer;
} SESSION_INFO_1_CONTAINER,
 *PSESSION_INFO_1_CONTAINER,
 *LPSESSION_INFO_1_CONTAINER;

typedef struct _SESSION_INFO_2
{
    [string] wchar_t * sesi2_cname;
    [string] wchar_t * sesi2_username;
    DWORD sesi2_num_opens;
    DWORD sesi2_time;
    DWORD sesi2_idle_time;
    DWORD sesi2_user_flags;
    [string] wchar_t * sesi2_cltype_name;
} SESSION_INFO_2,
 *PSESSION_INFO_2,
 *LPSESSION_INFO_2;

typedef struct _SESSION_INFO_2_CONTAINER
{
```

```
        DWORD EntriesRead;
        [size_is(EntriesRead)] LPSESSION_INFO_2 Buffer;
    } SESSION_INFO_2_CONTAINER,
     *PSESSION_INFO_2_CONTAINER,
     *LPSESSION_INFO_2_CONTAINER;

    typedef struct _SESSION_INFO_10
    {
        [string] wchar_t * sesi10_cname;
        [string] wchar_t * sesi10_username;
        DWORD sesi10_time;
        DWORD sesi10_idle_time;
    } SESSION_INFO_10,
     *PSESSION_INFO_10,
     *LPSESSION_INFO_10;

    typedef struct _SESSION_INFO_10_CONTAINER
    {
        DWORD EntriesRead;
        [size_is(EntriesRead)] LPSESSION_INFO_10 Buffer;
    } SESSION_INFO_10_CONTAINER,
     *PSESSION_INFO_10_CONTAINER,
     *LPSESSION_INFO_10_CONTAINER;

    typedef struct _SESSION_INFO_502
    {
        [string] wchar_t * sesi502_cname;
        [string] wchar_t * sesi502_username;
        DWORD sesi502_num_opens;
        DWORD sesi502_time;
        DWORD sesi502_idle_time;
        DWORD sesi502_user_flags;
        [string] wchar_t * sesi502_cltype_name;
        [string] wchar_t * sesi502_transport;
    } SESSION_INFO_502,
     *PSESSION_INFO_502,
     *LPSESSION_INFO_502;

    typedef struct _SESSION_INFO_502_CONTAINER
    {
        DWORD EntriesRead;
        [size_is(EntriesRead)] LPSESSION_INFO_502 Buffer;
    } SESSION_INFO_502_CONTAINER,
     *PSESSION_INFO_502_CONTAINER,
     *LPSESSION_INFO_502_CONTAINER;

    typedef [switch_type(DWORD)] union _SESSION_ENUM_UNION {
[case(0)]
        SESSION_INFO_0_CONTAINER* Level0;
        [case(1)]
        SESSION_INFO_1_CONTAINER* Level1;
        [case(2)]
        SESSION_INFO_2_CONTAINER* Level2;
        [case(10)]
            SESSION_INFO_10_CONTAINER* Level10;
        [case(502)]
        SESSION_INFO_502_CONTAINER* Level502;
    } SESSION_ENUM_UNION;
```

```
typedef struct _SESSION_ENUM_STRUCT {
    DWORD Level;
    [switch_is(Level)] SESSION_ENUM_UNION SessionInfo;
} SESSION_ENUM_STRUCT,
 *PSESSION_ENUM_STRUCT,
 *LPSESSION_ENUM_STRUCT;

typedef struct _SHARE_INFO_502_I
{
    [string] WCHAR * shi502_netname;
    DWORD shi502_type;
    [string] WCHAR * shi502_remark;
    DWORD shi502_permissions;
    DWORD shi502_max_uses;
    DWORD shi502_current_uses;
    [string] WCHAR * shi502_path;
    [string] WCHAR * shi502_passwd;
    DWORD shi502_reserved;
    [size_is(shi502_reserved)] unsigned char
            * shi502_security_descriptor;
} SHARE_INFO_502_I,
 *PSHARE_INFO_502_I,
 *LPSHARE_INFO_502_I;

typedef struct _SHARE_INFO_1501_I
{
    DWORD shi1501_reserved;
    [size_is(shi1501_reserved)] unsigned char
            * shi1501_security_descriptor;
} SHARE_INFO_1501_I,
 *PSHARE_INFO_1501_I,
 *LPSHARE_INFO_1501_I;

typedef struct _SHARE_INFO_0
{
    [string] wchar_t * shi0_netname;
} SHARE_INFO_0,
 *PSHARE_INFO_0,
 *LPSHARE_INFO_0;

typedef struct _SHARE_INFO_0_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSHARE_INFO_0 Buffer;
} SHARE_INFO_0_CONTAINER;

typedef struct _SHARE_INFO_1
{
    [string] wchar_t * shi1_netname;
    DWORD shi1_type;
    [string] wchar_t * shi1_remark;
} SHARE_INFO_1,
 *PSHARE_INFO_1,
 *LPSHARE_INFO_1;

typedef struct _SHARE_INFO_1_CONTAINER
{
```

```
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSHARE_INFO_1 Buffer;
} SHARE_INFO_1_CONTAINER;

typedef struct _SHARE_INFO_2
{
    [string] wchar_t * shi2_netname;
    DWORD shi2_type;
    [string] wchar_t * shi2_remark;
    DWORD shi2_permissions;
    DWORD shi2_max_uses;
    DWORD shi2_current_uses;
    [string] wchar_t * shi2_path;
    [string] wchar_t * shi2_passwd;
} SHARE_INFO_2,
 *PSHARE_INFO_2,
 *LPSHARE_INFO_2;

typedef struct _SHARE_INFO_2_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSHARE_INFO_2 Buffer;
} SHARE_INFO_2_CONTAINER,
 *PSHARE_INFO_2_CONTAINER,
 *LPSHARE_INFO_2_CONTAINER;

typedef struct _SHARE_INFO_501
{
    [string] wchar_t * shi501_netname;
    DWORD shi501_type;
    [string] wchar_t * shi501_remark;
    DWORD shi501_flags;
} SHARE_INFO_501,
 *PSHARE_INFO_501,
 *LPSHARE_INFO_501;

typedef struct _SHARE_INFO_501_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSHARE_INFO_501 Buffer;
} SHARE_INFO_501_CONTAINER, *PSHARE_INFO_501_CONTAINER,
  *LPSHARE_INFO_501_CONTAINER;

typedef struct _SHARE_INFO_502_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSHARE_INFO_502_I Buffer;
} SHARE_INFO_502_CONTAINER,
 *PSHARE_INFO_502_CONTAINER,
 *LPSHARE_INFO_502_CONTAINER;

typedef [switch_type(DWORD)] union _SHARE_ENUM_UNION {
  [case(0)]
     SHARE_INFO_0_CONTAINER* Level0;
  [case(1)]
     SHARE_INFO_1_CONTAINER* Level1;
  [case(2)]
     SHARE_INFO_2_CONTAINER* Level2;
```

```
        [case(501)]
           SHARE_INFO_501_CONTAINER* Level501;
        [case(502)]
           SHARE_INFO_502_CONTAINER* Level502;
    } SHARE_ENUM_UNION;

    typedef struct _SHARE_ENUM_STRUCT
    {
        DWORD Level;
        [switch_is(Level)] SHARE_ENUM_UNION ShareInfo;
    } SHARE_ENUM_STRUCT,
     *PSHARE_ENUM_STRUCT,
     *LPSHARE_ENUM_STRUCT;

    typedef struct _SHARE_INFO_1004
    {
        [string] wchar_t * shi1004_remark;
    } SHARE_INFO_1004,
     *PSHARE_INFO_1004,
     *LPSHARE_INFO_1004;

    typedef struct _SHARE_INFO_1006
    {
        DWORD shi1006_max_uses;
    } SHARE_INFO_1006,
     *PSHARE_INFO_1006,
     *LPSHARE_INFO_1006;

    typedef struct _SHARE_INFO_1005
    {
        DWORD shi1005_flags;
    } SHARE_INFO_1005,
     *PSHARE_INFO_1005,
     *LPSHARE_INFO_1005;

//JMP: order differs in documentation
    typedef [switch_type(unsigned long)] union _SHARE_INFO
    {
        [case(0)]
            LPSHARE_INFO_0 ShareInfo0;
        [case(1)]
            LPSHARE_INFO_1 ShareInfo1;
        [case(2)]
            LPSHARE_INFO_2 ShareInfo2;
        [case(502)]
            LPSHARE_INFO_502_I ShareInfo502;
        [case(1004)]
            LPSHARE_INFO_1004 ShareInfo1004;
        [case(1006)]
            LPSHARE_INFO_1006 ShareInfo1006;
        [case(1501)]
            LPSHARE_INFO_1501_I ShareInfo1501;
        [case(1005)]
            LPSHARE_INFO_1005 ShareInfo1005;
        [case(501)]
            LPSHARE_INFO_501 ShareInfo501;
    } SHARE_INFO,
     *PSHARE_INFO,
```

```
 *LPSHARE_INFO;

typedef struct _SERVER_INFO_100
{
    DWORD sv100_platform_id;
    [string] wchar_t * sv100_name;
} SERVER_INFO_100,
 *PSERVER_INFO_100,
 *LPSERVER_INFO_100;

typedef struct _SERVER_INFO_101
{
    DWORD sv101_platform_id;
    [string] wchar_t * sv101_name;
    DWORD sv101_version_major;
    DWORD sv101_version_minor;
    DWORD sv101_type;
    [string] wchar_t * sv101_comment;
} SERVER_INFO_101,
 *PSERVER_INFO_101,
 *LPSERVER_INFO_101;

typedef struct _SERVER_INFO_102
{
     DWORD sv102_platform_id;
     [string] wchar_t * sv102_name;
     DWORD sv102_version_major;
     DWORD sv102_version_minor;
     DWORD sv102_type;
     [string] wchar_t * sv102_comment;
     DWORD sv102_users;
     long sv102_disc;
     int sv102_hidden;
     DWORD sv102_announce;
     DWORD sv102_anndelta;
     DWORD sv102_licenses;
     [string] wchar_t * sv102_userpath;
} SERVER_INFO_102,
 *PSERVER_INFO_102,
 *LPSERVER_INFO_102;

typedef struct _SERVER_INFO_502
{
    DWORD sv502_sessopens;
    DWORD sv502_sessvcs;
    DWORD sv502_opensearch;
    DWORD sv502_sizreqbuf;
    DWORD sv502_initworkitems;
    DWORD sv502_maxworkitems;
    DWORD sv502_rawworkitems;
    DWORD sv502_irpstacksize;
    DWORD sv502_maxrawbuflen;
    DWORD sv502_sessusers;
    DWORD sv502_sessconns;
    DWORD sv502_maxpagedmemoryusage;
    DWORD sv502_maxnonpagedmemoryusage;
    int sv502_enablesoftcompat;
    int sv502_enableforcedlogoff;
```

```
    int sv502_timesource;
    int sv502_acceptdownlevelapis;
    int sv502_lmannounce;
} SERVER_INFO_502,
 *PSERVER_INFO_502,
 *LPSERVER_INFO_502;

typedef struct _SERVER_INFO_503
{
    DWORD sv503_sessopens;
    DWORD sv503_sessvcs;
    DWORD sv503_opensearch;
    DWORD sv503_sizreqbuf;
    DWORD sv503_initworkitems;
    DWORD sv503_maxworkitems;
    DWORD sv503_rawworkitems;
    DWORD sv503_irpstacksize;
    DWORD sv503_maxrawbuflen;
    DWORD sv503_sessusers;
    DWORD sv503_sessconns;
    DWORD sv503_maxpagedmemoryusage;
    DWORD sv503_maxnonpagedmemoryusage;
    int sv503_enablesoftcompat;
    int sv503_enableforcedlogoff;
    int sv503_timesource;
    int sv503_acceptdownlevelapis;
    int sv503_lmannounce;
    [string] wchar_t * sv503_domain;
    DWORD sv503_maxcopyreadlen;
    DWORD sv503_maxcopywritelen;
    DWORD sv503_minkeepsearch;
    DWORD sv503_maxkeepsearch;
    DWORD sv503_minkeepcomplsearch;
    DWORD sv503_maxkeepcomplsearch;
    DWORD sv503_threadcountadd;
    DWORD sv503_numblockthreads;
    DWORD sv503_scavtimeout;
    DWORD sv503_minrcvqueue;
    DWORD sv503_minfreeworkitems;
    DWORD sv503_xactmemsize;
    DWORD sv503_threadpriority;
    DWORD sv503_maxmpxct;
    DWORD sv503_oplockbreakwait;
    DWORD sv503_oplockbreakresponsewait;
    int sv503_enableoplocks;
    int sv503_enableoplockforceclose;
    int sv503_enablefcbopens;
    int sv503_enableraw;
    int sv503_enablesharednetdrives;
    DWORD sv503_minfreeconnections;
    DWORD sv503_maxfreeconnections;
} SERVER_INFO_503,
 *PSERVER_INFO_503,
 *LPSERVER_INFO_503;

typedef struct _SERVER_INFO_599
{
    DWORD sv599_sessopens;
```

```
    DWORD sv599_sessvcs;
    DWORD sv599_opensearch;
    DWORD sv599_sizreqbuf;
    DWORD sv599_initworkitems;
    DWORD sv599_maxworkitems;
    DWORD sv599_rawworkitems;
    DWORD sv599_irpstacksize;
    DWORD sv599_maxrawbuflen;
    DWORD sv599_sessusers;
    DWORD sv599_sessconns;
    DWORD sv599_maxpagedmemoryusage;
    DWORD sv599_maxnonpagedmemoryusage;
    int sv599_enablesoftcompat;
    int sv599_enableforcedlogoff;
    int sv599_timesource;
    int sv599_acceptdownlevelapis;
    int sv599_lmannounce;
    [string] wchar_t * sv599_domain;
    DWORD sv599_maxcopyreadlen;
    DWORD sv599_maxcopywritelen;
    DWORD sv599_minkeepsearch;
    DWORD sv599_maxkeepsearch;
    DWORD sv599_minkeepcomplsearch;
    DWORD sv599_maxkeepcomplsearch;
    DWORD sv599_threadcountadd;
    DWORD sv599_numblockthreads;
    DWORD sv599_scavtimeout;
    DWORD sv599_minrcvqueue;
    DWORD sv599_minfreeworkitems;
    DWORD sv599_xactmemsize;
    DWORD sv599_threadpriority;
    DWORD sv599_maxmpxct;
    DWORD sv599_oplockbreakwait;
    DWORD sv599_oplockbreakresponsewait;
    int sv599_enableoplocks;
    int sv599_enableoplockforceclose;
    int sv599_enablefcbopens;
    int sv599_enableraw;
    int sv599_enablesharednetdrives;
    DWORD sv599_minfreeconnections;
    DWORD sv599_maxfreeconnections;
    DWORD sv599_initsesstable;
    DWORD sv599_initconntable;
    DWORD sv599_initfiletable;
    DWORD sv599_initsearchtable;
    DWORD sv599_alertschedule;
    DWORD sv599_errorthreshold;
    DWORD sv599_networkerrorthreshold;
    DWORD sv599_diskspacethreshold;
    DWORD sv599_reserved;
    DWORD sv599_maxlinkdelay;
    DWORD sv599_minlinkthroughput;
    DWORD sv599_linkinfovalidtime;
    DWORD sv599_scavqosinfoupdatetime;
    DWORD sv599_maxworkitemidletime;
} SERVER_INFO_599,
 *PSERVER_INFO_599,
 *LPSERVER_INFO_599;
```

```
typedef struct _SERVER_INFO_1005
{
    [string] wchar_t * sv1005_comment;
} SERVER_INFO_1005,
 *PSERVER_INFO_1005,
 *LPSERVER_INFO_1005;

typedef struct _SERVER_INFO_1107
{
    DWORD sv1107_users;
} SERVER_INFO_1107,
 *PSERVER_INFO_1107,
 *LPSERVER_INFO_1107;

typedef struct _SERVER_INFO_1010
{
    long sv1010_disc;
} SERVER_INFO_1010,
 *PSERVER_INFO_1010,
 *LPSERVER_INFO_1010;

typedef struct _SERVER_INFO_1016
{
    int sv1016_hidden;
} SERVER_INFO_1016,
 *PSERVER_INFO_1016,
 *LPSERVER_INFO_1016;

typedef struct _SERVER_INFO_1017
{
    DWORD sv1017_announce;
} SERVER_INFO_1017,
 *PSERVER_INFO_1017,
 *LPSERVER_INFO_1017;

typedef struct _SERVER_INFO_1018
{
    DWORD sv1018_anndelta;
} SERVER_INFO_1018,
 *PSERVER_INFO_1018,
 *LPSERVER_INFO_1018;

typedef struct _SERVER_INFO_1501
{
    DWORD sv1501_sessopens;
} SERVER_INFO_1501,
 *PSERVER_INFO_1501,
 *LPSERVER_INFO_1501;

typedef struct _SERVER_INFO_1502
{
    DWORD sv1502_sessvcs;
} SERVER_INFO_1502,
 *PSERVER_INFO_1502,
 *LPSERVER_INFO_1502;

typedef struct _SERVER_INFO_1503
```

```
{
    DWORD sv1503_opensearch;
} SERVER_INFO_1503, *PSERVER_INFO_1503, *LPSERVER_INFO_1503;

typedef struct _SERVER_INFO_1506
{
    DWORD sv1506_maxworkitems;
} SERVER_INFO_1506, *PSERVER_INFO_1506, *LPSERVER_INFO_1506;

typedef struct _SERVER_INFO_1510
{
    DWORD sv1510_sessusers;
} SERVER_INFO_1510, *PSERVER_INFO_1510, *LPSERVER_INFO_1510;

typedef struct _SERVER_INFO_1511
{
    DWORD sv1511_sessconns;
} SERVER_INFO_1511, *PSERVER_INFO_1511, *LPSERVER_INFO_1511;

typedef struct _SERVER_INFO_1512
{
    DWORD sv1512_maxnonpagedmemoryusage;
} SERVER_INFO_1512, *PSERVER_INFO_1512, *LPSERVER_INFO_1512;

typedef struct _SERVER_INFO_1513
{
    DWORD sv1513_maxpagedmemoryusage;
} SERVER_INFO_1513, *PSERVER_INFO_1513, *LPSERVER_INFO_1513;

typedef struct _SERVER_INFO_1514
{
    int sv1514_enablesoftcompat;
} SERVER_INFO_1514, *PSERVER_INFO_1514, *LPSERVER_INFO_1514;

typedef struct _SERVER_INFO_1515
{
    int sv1515_enableforcedlogoff;
} SERVER_INFO_1515, *PSERVER_INFO_1515, *LPSERVER_INFO_1515;

typedef struct _SERVER_INFO_1516
{
    int sv1516_timesource;
} SERVER_INFO_1516, *PSERVER_INFO_1516, *LPSERVER_INFO_1516;

typedef struct _SERVER_INFO_1518
{
    int sv1518_lmannounce;
} SERVER_INFO_1518, *PSERVER_INFO_1518, *LPSERVER_INFO_1518;

typedef struct _SERVER_INFO_1523
{
    DWORD sv1523_maxkeepsearch;
} SERVER_INFO_1523, *PSERVER_INFO_1523, *LPSERVER_INFO_1523;

typedef struct _SERVER_INFO_1528
{
    DWORD sv1528_scavtimeout;
} SERVER_INFO_1528, *PSERVER_INFO_1528, *LPSERVER_INFO_1528;
```

```
typedef struct _SERVER_INFO_1529
{
    DWORD sv1529_minrcvqueue;
} SERVER_INFO_1529, *PSERVER_INFO_1529, *LPSERVER_INFO_1529;

typedef struct _SERVER_INFO_1530
{
    DWORD sv1530_minfreeworkitems;
} SERVER_INFO_1530, *PSERVER_INFO_1530, *LPSERVER_INFO_1530;

typedef struct _SERVER_INFO_1533
{
    DWORD sv1533_maxmpxct;
} SERVER_INFO_1533, *PSERVER_INFO_1533, *LPSERVER_INFO_1533;

typedef struct _SERVER_INFO_1534
{
    DWORD sv1534_oplockbreakwait;
} SERVER_INFO_1534, *PSERVER_INFO_1534, *LPSERVER_INFO_1534;

typedef struct _SERVER_INFO_1535
{
    DWORD sv1535_oplockbreakresponsewait;
} SERVER_INFO_1535, *PSERVER_INFO_1535, *LPSERVER_INFO_1535;

typedef struct _SERVER_INFO_1536
{
    int sv1536_enableoplocks;
} SERVER_INFO_1536, *PSERVER_INFO_1536, *LPSERVER_INFO_1536;

typedef struct _SERVER_INFO_1538
{
    int sv1538_enablefcbopens;
} SERVER_INFO_1538, *PSERVER_INFO_1538, *LPSERVER_INFO_1538;

typedef struct _SERVER_INFO_1539
{
    int sv1539_enableraw;
} SERVER_INFO_1539, *PSERVER_INFO_1539, *LPSERVER_INFO_1539;

typedef struct _SERVER_INFO_1540
{
    int sv1540_enablesharednetdrives;
} SERVER_INFO_1540, *PSERVER_INFO_1540, *LPSERVER_INFO_1540;

typedef struct _SERVER_INFO_1541
{
    int sv1541_minfreeconnections;
} SERVER_INFO_1541, *PSERVER_INFO_1541, *LPSERVER_INFO_1541;

typedef struct _SERVER_INFO_1542
{
    int sv1542_maxfreeconnections;
} SERVER_INFO_1542, *PSERVER_INFO_1542, *LPSERVER_INFO_1542;

typedef struct _SERVER_INFO_1543
{
```

```
    DWORD sv1543_initsesstable;
} SERVER_INFO_1543, *PSERVER_INFO_1543, *LPSERVER_INFO_1543;

typedef struct _SERVER_INFO_1544
{
    DWORD sv1544_initconntable;
} SERVER_INFO_1544, *PSERVER_INFO_1544, *LPSERVER_INFO_1544;

typedef struct _SERVER_INFO_1545
{
    DWORD sv1545_initfiletable;
} SERVER_INFO_1545, *PSERVER_INFO_1545, *LPSERVER_INFO_1545;

typedef struct _SERVER_INFO_1546
{
    DWORD sv1546_initsearchtable;
} SERVER_INFO_1546, *PSERVER_INFO_1546, *LPSERVER_INFO_1546;

typedef struct _SERVER_INFO_1547
{
    DWORD sv1547_alertschedule;
} SERVER_INFO_1547, *PSERVER_INFO_1547, *LPSERVER_INFO_1547;

typedef struct _SERVER_INFO_1548
{
    DWORD sv1548_errorthreshold;
} SERVER_INFO_1548, *PSERVER_INFO_1548, *LPSERVER_INFO_1548;

typedef struct _SERVER_INFO_1549
{
    DWORD sv1549_networkerrorthreshold;
} SERVER_INFO_1549, *PSERVER_INFO_1549, *LPSERVER_INFO_1549;

typedef struct _SERVER_INFO_1550
{
    DWORD sv1550_diskspacethreshold;
} SERVER_INFO_1550, *PSERVER_INFO_1550, *LPSERVER_INFO_1550;

typedef struct _SERVER_INFO_1552
{
    DWORD sv1552_maxlinkdelay;
} SERVER_INFO_1552, *PSERVER_INFO_1552, *LPSERVER_INFO_1552;

typedef struct _SERVER_INFO_1553
{
    DWORD sv1553_minlinkthroughput;
} SERVER_INFO_1553, *PSERVER_INFO_1553, *LPSERVER_INFO_1553;

typedef struct _SERVER_INFO_1554
{
    DWORD sv1554_linkinfovalidtime;
} SERVER_INFO_1554, *PSERVER_INFO_1554, *LPSERVER_INFO_1554;

typedef struct _SERVER_INFO_1555
{
    DWORD sv1555_scavqosinfoupdatetime;
} SERVER_INFO_1555, *PSERVER_INFO_1555, *LPSERVER_INFO_1555;
```

```
typedef struct _SERVER_INFO_1556
{
    DWORD sv1556_maxworkitemidletime;
} SERVER_INFO_1556, *PSERVER_INFO_1556, *LPSERVER_INFO_1556;

typedef [switch_type(unsigned long)] union _SERVER_INFO
{
    [case(100)]
        LPSERVER_INFO_100 ServerInfo100;
    [case(101)]
        LPSERVER_INFO_101 ServerInfo101;
    [case(102)]
        LPSERVER_INFO_102 ServerInfo102;
    [case(502)]
        LPSERVER_INFO_502 ServerInfo502;
    [case(503)]
        LPSERVER_INFO_503 ServerInfo503;
    [case(599)]
        LPSERVER_INFO_599 ServerInfo599;
    [case(1005)]
        LPSERVER_INFO_1005 ServerInfo1005;
    [case(1107)]
        LPSERVER_INFO_1107 ServerInfo1107;
    [case(1010)]
        LPSERVER_INFO_1010 ServerInfo1010;
    [case(1016)]
        LPSERVER_INFO_1016 ServerInfo1016;
    [case(1017)]
        LPSERVER_INFO_1017 ServerInfo1017;
    [case(1018)]
        LPSERVER_INFO_1018 ServerInfo1018;
    [case(1501)]
        LPSERVER_INFO_1501 ServerInfo1501;
    [case(1502)]
        LPSERVER_INFO_1502 ServerInfo1502;
    [case(1503)]
        LPSERVER_INFO_1503 ServerInfo1503;
    [case(1506)]
        LPSERVER_INFO_1506 ServerInfo1506;
    [case(1510)]
        LPSERVER_INFO_1510 ServerInfo1510;
    [case(1511)]
        LPSERVER_INFO_1511 ServerInfo1511;
    [case(1512)]
        LPSERVER_INFO_1512 ServerInfo1512;
    [case(1513)]
        LPSERVER_INFO_1513 ServerInfo1513;
    [case(1514)]
        LPSERVER_INFO_1514 ServerInfo1514;
    [case(1515)]
        LPSERVER_INFO_1515 ServerInfo1515;
    [case(1516)]
        LPSERVER_INFO_1516 ServerInfo1516;
    [case(1518)]
        LPSERVER_INFO_1518 ServerInfo1518;
    [case(1523)]
        LPSERVER_INFO_1523 ServerInfo1523;
    [case(1528)]
```

```
            LPSERVER_INFO_1528 ServerInfo1528;
        [case(1529)]
            LPSERVER_INFO_1529 ServerInfo1529;
        [case(1530)]
            LPSERVER_INFO_1530 ServerInfo1530;
        [case(1533)]
            LPSERVER_INFO_1533 ServerInfo1533;
        [case(1534)]
            LPSERVER_INFO_1534 ServerInfo1534;
        [case(1535)]
            LPSERVER_INFO_1535 ServerInfo1535;
        [case(1536)]
            LPSERVER_INFO_1536 ServerInfo1536;
        [case(1538)]
            LPSERVER_INFO_1538 ServerInfo1538;
        [case(1539)]
            LPSERVER_INFO_1539 ServerInfo1539;
        [case(1540)]
            LPSERVER_INFO_1540 ServerInfo1540;
        [case(1541)]
            LPSERVER_INFO_1541 ServerInfo1541;
        [case(1542)]
            LPSERVER_INFO_1542 ServerInfo1542;
        [case(1543)]
            LPSERVER_INFO_1543 ServerInfo1543;
        [case(1544)]
            LPSERVER_INFO_1544 ServerInfo1544;
        [case(1545)]
            LPSERVER_INFO_1545 ServerInfo1545;
        [case(1546)]
            LPSERVER_INFO_1546 ServerInfo1546;
        [case(1547)]
            LPSERVER_INFO_1547 ServerInfo1547;
        [case(1548)]
            LPSERVER_INFO_1548 ServerInfo1548;
        [case(1549)]
            LPSERVER_INFO_1549 ServerInfo1549;
        [case(1550)]
            LPSERVER_INFO_1550 ServerInfo1550;
        [case(1552)]
            LPSERVER_INFO_1552 ServerInfo1552;
        [case(1553)]
            LPSERVER_INFO_1553 ServerInfo1553;
        [case(1554)]
            LPSERVER_INFO_1554 ServerInfo1554;
        [case(1555)]
            LPSERVER_INFO_1555 ServerInfo1555;
        [case(1556)]
            LPSERVER_INFO_1556 ServerInfo1556;
    } SERVER_INFO, *PSERVER_INFO, *LPSERVER_INFO;

    typedef struct _DISK_INFO
    {
        [string] WCHAR Disk[3];
    } DISK_INFO, *PDISK_INFO, *LPDISK_INFO;

    typedef struct _DISK_ENUM_CONTAINER
    {
```

```
    DWORD EntriesRead;
    [size_is(EntriesRead), length_is(EntriesRead)] LPDISK_INFO
         Buffer;
} DISK_ENUM_CONTAINER;

typedef struct _SERVER_TRANSPORT_INFO_0
{
    DWORD svti0_numberofvcs;
    [string] wchar_t * svti0_transportname;
    [size_is(svti0_transportaddresslength)] unsigned char
         * svti0_transportaddress;
    DWORD svti0_transportaddresslength;
    [string] wchar_t * svti0_networkaddress;
} SERVER_TRANSPORT_INFO_0, *PSERVER_TRANSPORT_INFO_0,
  *LPSERVER_TRANSPORT_INFO_0;

typedef struct _SERVER_XPORT_INFO_0_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSERVER_TRANSPORT_INFO_0 Buffer;
} SERVER_XPORT_INFO_0_CONTAINER, *PSERVER_XPORT_INFO_0_CONTAINER;

typedef struct _SERVER_TRANSPORT_INFO_1
{
    DWORD svti1_numberofvcs;
    [string] wchar_t * svti1_transportname;
    [size_is(svti1_transportaddresslength)] unsigned char
         * svti1_transportaddress;
    DWORD svti1_transportaddresslength;
    [string] wchar_t * svti1_networkaddress;
    [string] wchar_t * svti1_domain;
} SERVER_TRANSPORT_INFO_1, *PSERVER_TRANSPORT_INFO_1,
  *LPSERVER_TRANSPORT_INFO_1;

typedef struct _SERVER_XPORT_INFO_1_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSERVER_TRANSPORT_INFO_1 Buffer;
} SERVER_XPORT_INFO_1_CONTAINER, *PSERVER_XPORT_INFO_1_CONTAINER;

typedef struct _SERVER_TRANSPORT_INFO_2
{
    DWORD svti2_numberofvcs;
    [string] wchar_t * svti2_transportname;
    [size_is(svti2_transportaddresslength)] unsigned char
         * svti2_transportaddress;
    DWORD svti2_transportaddresslength;
    [string] wchar_t * svti2_networkaddress;
    [string] wchar_t * svti2_domain;
    unsigned long svti2_flags;
} SERVER_TRANSPORT_INFO_2, *PSERVER_TRANSPORT_INFO_2,
  *LPSERVER_TRANSPORT_INFO_2;

typedef struct _SERVER_XPORT_INFO_2_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSERVER_TRANSPORT_INFO_2 Buffer;
} SERVER_XPORT_INFO_2_CONTAINER, *PSERVER_XPORT_INFO_2_CONTAINER;
```

```
typedef struct _SERVER_TRANSPORT_INFO_3
{
    DWORD svti3_numberofvcs;
    [string] wchar_t * svti3_transportname;
    [size_is(svti3_transportaddresslength)] unsigned char
         * svti3_transportaddress;
    DWORD svti3_transportaddresslength;
    [string] wchar_t * svti3_networkaddress;
    [string] wchar_t * svti3_domain;
    unsigned long svti3_flags;
    DWORD svti3_passwordlength;
    unsigned char svti3_password[ 256 ];
} SERVER_TRANSPORT_INFO_3, *PSERVER_TRANSPORT_INFO_3,
  *LPSERVER_TRANSPORT_INFO_3;

typedef struct _SERVER_XPORT_INFO_3_CONTAINER
{
    DWORD EntriesRead;
    [size_is(EntriesRead)] LPSERVER_TRANSPORT_INFO_3 Buffer;
} SERVER_XPORT_INFO_3_CONTAINER, *PSERVER_XPORT_INFO_3_CONTAINER;

typedef [switch_type(unsigned long)] union _TRANSPORT_INFO
{
    [case(0)]
        SERVER_TRANSPORT_INFO_0 Transport0;
    [case(1)]
        SERVER_TRANSPORT_INFO_1 Transport1;
    [case(2)]
        SERVER_TRANSPORT_INFO_2 Transport2;
    [case(3)]
        SERVER_TRANSPORT_INFO_3 Transport3;
} TRANSPORT_INFO, *PTRANSPORT_INFO, *LPTRANSPORT_INFO;

typedef [switch_type(DWORD)] union _SERVER_XPORT_ENUM_UNION {
   [case(0)]
       PSERVER_XPORT_INFO_0_CONTAINER Level0;
   [case(1)]
       PSERVER_XPORT_INFO_1_CONTAINER Level1;
   [case(2)]
       PSERVER_XPORT_INFO_2_CONTAINER Level2;
   [case(3)]
       PSERVER_XPORT_INFO_3_CONTAINER Level3;
} SERVER_XPORT_ENUM_UNION;

typedef struct _SERVER_XPORT_ENUM_STRUCT
{
    DWORD Level;
    [switch_is(Level)] SERVER_XPORT_ENUM_UNION XportInfo;

} SERVER_XPORT_ENUM_STRUCT, *PSERVER_XPORT_ENUM_STRUCT,
  *LPSERVER_XPORT_ENUM_STRUCT;

typedef [context_handle] void *SHARE_DEL_HANDLE;
typedef SHARE_DEL_HANDLE *PSHARE_DEL_HANDLE;

typedef struct _ADT_SECURITY_DESCRIPTOR
{
```

```
    DWORD Length;
    [size_is(Length)] unsigned char * Buffer;
} ADT_SECURITY_DESCRIPTOR, *PADT_SECURITY_DESCRIPTOR;

typedef struct _STAT_SERVER_0
{
     DWORD sts0_start;
     DWORD sts0_fopens;
     DWORD sts0_devopens;
     DWORD sts0_jobsqueued;
     DWORD sts0_sopens;
     DWORD sts0_stimedout;
     DWORD sts0_serrorout;
     DWORD sts0_pwerrors;
     DWORD sts0_permerrors;
     DWORD sts0_syserrors;
     DWORD sts0_bytessent_low;
     DWORD sts0_bytessent_high;
     DWORD sts0_bytesrcvd_low;
     DWORD sts0_bytesrcvd_high;
     DWORD sts0_avresponse;
     DWORD sts0_reqbufneed;
     DWORD sts0_bigbufneed;
} STAT_SERVER_0, *PSTAT_SERVER_0, *LPSTAT_SERVER_0;

typedef struct _TIME_OF_DAY_INFO
{
    DWORD tod_elapsedt;
    DWORD tod_msecs;
    DWORD tod_hours;
    DWORD tod_mins;
    DWORD tod_secs;
    DWORD tod_hunds;
    long tod_timezone;
    DWORD tod_tinterval;
    DWORD tod_day;
    DWORD tod_month;
    DWORD tod_year;
    DWORD tod_weekday;
} TIME_OF_DAY_INFO, *PTIME_OF_DAY_INFO, *LPTIME_OF_DAY_INFO;

typedef struct _NET_DFS_ENTRY_ID
{
    GUID Uid;
    [string] WCHAR * Prefix;
} NET_DFS_ENTRY_ID, *LPNET_DFS_ENTRY_ID;

typedef struct _NET_DFS_ENTRY_ID_CONTAINER
{
    unsigned long Count;
    [size_is(Count)] LPNET_DFS_ENTRY_ID Buffer;
} NET_DFS_ENTRY_ID_CONTAINER, *LPNET_DFS_ENTRY_ID_CONTAINER;

typedef struct _DFS_SITENAME_INFO
{
    unsigned long SiteFlags;
    [string,unique] WCHAR * SiteName;
} DFS_SITENAME_INFO, *PDFS_SITENAME_INFO, *LPDFS_SITENAME_INFO;
```

```
    typedef struct _DFS_SITELIST_INFO
    {
        unsigned long cSites;
        [size_is(cSites)] DFS_SITENAME_INFO Site[];
    } DFS_SITELIST_INFO, *PDFS_SITELIST_INFO, *LPDFS_SITELIST_INFO;

// This method not used on the wire
    void Opnum0NotUsedOnWire(void);

// This method not used on the wire
    void Opnum1NotUsedOnWire(void);

// This method not used on the wire
    void Opnum2NotUsedOnWire(void);

// This method not used on the wire
    void Opnum3NotUsedOnWire(void);

// This method not used on the wire
    void Opnum4NotUsedOnWire(void);

// This method not used on the wire
    void Opnum5NotUsedOnWire(void);

// This method not used on the wire
    void Opnum6NotUsedOnWire(void);

// This method not used on the wire
    void Opnum7NotUsedOnWire(void);

    NET_API_STATUS
    NetrConnectionEnum (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in,string,unique] WCHAR * Qualifier,
        [in,out] LPCONNECT_ENUM_STRUCT InfoStruct,
        [in] DWORD PreferedMaximumLength,
        [out] DWORD * TotalEntries,
        [in,out,unique] DWORD * ResumeHandle
    );

    NET_API_STATUS
    NetrFileEnum (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in,string,unique] WCHAR * BasePath,
        [in,string,unique] WCHAR * UserName,
        [in,out] PFILE_ENUM_STRUCT InfoStruct,
        [in] DWORD PreferedMaximumLength,
        [out] DWORD * TotalEntries,
        [in,out,unique] DWORD * ResumeHandle
    );

    NET_API_STATUS
    NetrFileGetInfo (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in] DWORD FileId,
        [in] DWORD Level,
        [out, switch_is(Level)] LPFILE_INFO InfoStruct
```

```
);

NET_API_STATUS
NetrFileClose (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in] DWORD FileId
);

NET_API_STATUS
NetrSessionEnum (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string,unique] WCHAR * ClientName,
    [in,string,unique] WCHAR * UserName,
    [in,out] PSESSION_ENUM_STRUCT InfoStruct,
    [in] DWORD PreferedMaximumLength,
    [out] DWORD * TotalEntries,
    [in,out,unique] DWORD * ResumeHandle
);

NET_API_STATUS
NetrSessionDel (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string,unique] WCHAR * ClientName,
    [in,string,unique] WCHAR * UserName
);

NET_API_STATUS
NetrShareAdd (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in] DWORD Level,
    [in, switch_is(Level)] LPSHARE_INFO InfoStruct,
    [in,out,unique] DWORD * ParmErr
);

NET_API_STATUS
NetrShareEnum (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,out] LPSHARE_ENUM_STRUCT InfoStruct,
    [in] DWORD PreferedMaximumLength,
    [out] DWORD * TotalEntries,
    [in,out,unique] DWORD * ResumeHandle
);

NET_API_STATUS
NetrShareGetInfo (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string] WCHAR * NetName,
    [in] DWORD Level,
    [out, switch_is(Level)] LPSHARE_INFO InfoStruct
);

NET_API_STATUS
NetrShareSetInfo (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string] WCHAR * NetName,
    [in] DWORD Level,
    [in, switch_is(Level)] LPSHARE_INFO ShareInfo,
    [in,out,unique] DWORD * ParmErr
```

```
);

NET_API_STATUS
NetrShareDel (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string] WCHAR * NetName,
    [in] DWORD Reserved
);

NET_API_STATUS
NetrShareDelSticky (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string] WCHAR * NetName,
    [in] DWORD Reserved
);

NET_API_STATUS
NetrShareCheck (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string] WCHAR * Device,
    [out] DWORD * Type
);

NET_API_STATUS
NetrServerGetInfo (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in] DWORD Level,
    [out, switch_is(Level)] LPSERVER_INFO InfoStruct
);

NET_API_STATUS
NetrServerSetInfo (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in] DWORD Level,
    [in, switch_is(Level)] LPSERVER_INFO ServerInfo,
    [in,out,unique] DWORD * ParmErr
);

NET_API_STATUS
NetrServerDiskEnum (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in] DWORD Level,
    [in,out] DISK_ENUM_CONTAINER *DiskInfoStruct,
    [in] DWORD PreferredMaximumLength,
    [out] DWORD * TotalEntries,
    [in,out,unique] DWORD * ResumeHandle
);

NET_API_STATUS
NetrServerStatisticsGet (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string,unique] WCHAR * Service,
    [in] DWORD Level,
    [in] DWORD Options,
    [out] LPSTAT_SERVER_0 *InfoStruct
);

NET_API_STATUS
```

```
    NetrServerTransportAdd (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in] DWORD Level,
        [in] LPSERVER_TRANSPORT_INFO_0 Buffer
    );

    NET_API_STATUS
    NetrServerTransportEnum (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in,out] LPSERVER_XPORT_ENUM_STRUCT InfoStruct,
        [in] DWORD PreferedMaximumLength,
        [out] DWORD * TotalEntries,
        [in,out,unique] DWORD * ResumeHandle
    );

    NET_API_STATUS
    NetrServerTransportDel (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in] DWORD Level,
        [in] LPSERVER_TRANSPORT_INFO_0 Buffer
    );

    NET_API_STATUS
    NetrRemoteTOD (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [out] LPTIME_OF_DAY_INFO *BufferPtr
    );

// This method not used on the wire
    void Opnum29NotUsedOnWire(void);

    NET_API_STATUS
    NetprPathType(
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in,string] WCHAR * PathName,
        [out] DWORD * PathType,
        [in] DWORD Flags
    );

    NET_API_STATUS
    NetprPathCanonicalize(
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in,string] WCHAR * PathName,
        [out,size_is(OutbufLen)] unsigned char * Outbuf,
        [in,range(0, 64000)] DWORD OutbufLen,
        [in,string] WCHAR * Prefix,
        [in,out] DWORD * PathType,
        [in] DWORD Flags
    );

    long
    NetprPathCompare(
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in,string] WCHAR * PathName1,
        [in,string] WCHAR * PathName2,
        [in] DWORD PathType,
        [in] DWORD Flags
    );
```

```
NET_API_STATUS
NetprNameValidate(
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string] WCHAR * Name,
    [in] DWORD NameType,
    [in] DWORD Flags
);

NET_API_STATUS
NetprNameCanonicalize(
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string] WCHAR * Name,
    [out,size_is(OutbufLen)] WCHAR * Outbuf,
    [in,range(0, 64000)] DWORD OutbufLen,
    [in] DWORD NameType,
    [in] DWORD Flags
);

long
NetprNameCompare(
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string] WCHAR * Name1,
    [in,string] WCHAR * Name2,
    [in] DWORD NameType,
    [in] DWORD Flags
);

NET_API_STATUS
NetrShareEnumSticky (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,out] LPSHARE_ENUM_STRUCT InfoStruct,
    [in] DWORD PreferedMaximumLength,
    [out] DWORD * TotalEntries,
    [in,out,unique] DWORD * ResumeHandle
);

NET_API_STATUS
NetrShareDelStart (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string] WCHAR * NetName,
    [in] DWORD Reserved,
    [out] PSHARE_DEL_HANDLE ContextHandle
);

NET_API_STATUS
NetrShareDelCommit (
    [in, out] PSHARE_DEL_HANDLE ContextHandle
);

DWORD
NetrpGetFileSecurity (
    [in,string,unique] SRVSVC_HANDLE ServerName,
    [in,string,unique] WCHAR * ShareName,
    [in,string] WCHAR * lpFileName,
    [in] SECURITY_INFORMATION RequestedInformation,
    [out] PADT_SECURITY_DESCRIPTOR *SecurityDescriptor
);
```

```
    DWORD
    NetrpSetFileSecurity (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in,string,unique] WCHAR * ShareName,
        [in,string] WCHAR * lpFileName,
        [in] SECURITY_INFORMATION SecurityInformation,
        [in] PADT_SECURITY_DESCRIPTOR SecurityDescriptor
    );

    NET_API_STATUS
    NetrServerTransportAddEx (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in] DWORD Level,
        [in, switch_is(Level)] LPTRANSPORT_INFO Buffer
    );

// This method not used on the wire
    void Opnum42NotUsedOnWire(void);

    NET_API_STATUS
    NetrDfsGetVersion(
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [out] DWORD * Version
    );

    NET_API_STATUS
    NetrDfsCreateLocalPartition (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in,string] WCHAR * ShareName,
        [in] GUID * EntryUid,
        [in,string] WCHAR * EntryPrefix,
        [in,string] WCHAR * ShortName,
        [in] LPNET_DFS_ENTRY_ID_CONTAINER RelationInfo,
        [in] int Force
    );

    NET_API_STATUS
    NetrDfsDeleteLocalPartition (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in] GUID * Uid,
        [in,string] WCHAR * Prefix
    );

    NET_API_STATUS
    NetrDfsSetLocalVolumeState (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in] GUID * Uid,
        [in,string] WCHAR * Prefix,
        [in] unsigned long State
    );

// This method not used on the wire
    void Opnum47NotUsedOnWire(void);

    NET_API_STATUS
    NetrDfsCreateExitPoint (
        [in,string,unique] SRVSVC_HANDLE ServerName,
```

```
        [in] GUID * Uid,
        [in,string] WCHAR * Prefix,
        [in] unsigned long Type,
        [in, range(0,32) ] DWORD ShortPrefixLen,
        [out,size_is(ShortPrefixLen)] WCHAR * ShortPrefix
    );

    NET_API_STATUS
    NetrDfsDeleteExitPoint (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in] GUID * Uid,
        [in,string] WCHAR * Prefix,
        [in] unsigned long Type
    );

    NET_API_STATUS
    NetrDfsModifyPrefix (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in] GUID * Uid,
        [in,string] WCHAR * Prefix
    );

    NET_API_STATUS
    NetrDfsFixLocalVolume (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in,string] WCHAR * VolumeName,
        [in] unsigned long EntryType,
        [in] unsigned long ServiceType,
        [in,string] WCHAR * StgId,
        [in] GUID * EntryUid,
        [in,string] WCHAR * EntryPrefix,
        [in] LPNET_DFS_ENTRY_ID_CONTAINER RelationInfo,
        [in] unsigned long CreateDisposition
    );

    NET_API_STATUS
    NetrDfsManagerReportSiteInfo (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in,out,unique] LPDFS_SITELIST_INFO *ppSiteInfo
    );

    NET_API_STATUS
    NetrServerTransportDelEx (
        [in,string,unique] SRVSVC_HANDLE ServerName,
        [in] DWORD Level,
        [in, switch_is(Level)] LPTRANSPORT_INFO Buffer
    );
}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows NT
- Windows 2000
- Windows Server 2003
- Windows XP
- Windows Vista

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 1.8: Windows only uses the values in [MS-EERR].

<2> Section 2.1: Windows uses the identity of the caller to perform method specific access checks.

<3> Section 2.2.2.1: Windows-based SMB clients set this field based on the version and service pack level of the Windows operating system. A list of possible values for this field includes the following. Windows Vista sets this field to an empty string.

Windows OS Version	Meaning
Windows Server 2003 with SP1	"Windows Server 2003 3790 Service Pack 1"
Windows XP with SP2	"Windows 2002 Service Pack 2"
Windows 2000	"Windows 5.0"
Windows NT 4.0	"Windows NT 1381"
Windows 98 and Windows SE	"Windows 4.0"

<4> Section 2.2.2.1: Windows Server currently does not enforce any limits on the Sessionclient string size, and will accept any string containing 0 or more characters. The existing Windows clients limit the size to less than 256 bytes.

<5> Section 2.2.2.6: PLATFORM_ID_NT should be used for Windows NT Server, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008.

<6> Section 2.2.2.9: The following table provides example paths for different PathTypes as returned by Windows Servers.

Name	Example
ITYPE_UNC_COMPNAME	\products
ITYPE_UNC_WC	*
ITYPE_UNC	\\products\

Name	Example
ITYPE_UNC_WC_PATH	\\server* or \\server\share*
ITYPE_UNC_SYS_SEM	Not returned
ITYPE_UNC_SYS_SHMEM	Not returned
ITYPE_UNC_SYS_MSLOT	Not returned
ITYPE_UNC_SYS_PIPE	Not returned
ITYPE_UNC_SYS_QUEUE	Not returned
ITYPE_PATH_ABSND	\ or \folder or \folder\share
ITYPE_PATH_ABSD	\\.\windows\system32 or C:\windows
ITYPE_PATH_RELND	windows or windows\system32
ITYPE_PATH_RELD	Not returned
ITYPE_PATH_ABSND_WC	\folder\? \windows\system32*
ITYPE_PATH_ABSD_WC	C:\windows*\bin\ C:\windows\?\bin\
ITYPE_PATH_RELND_WC	windows*\bin\ windows\?\bin\
ITYPE_PATH_RELD_WC	Not returned
ITYPE_PATH_SYS_SEM	Not returned
ITYPE_PATH_SYS_SHMEM	Not returned
ITYPE_PATH_SYS_MSLOT	Not returned
ITYPE_PATH_SYS_PIPE	Not returned
ITYPE_PATH_SYS_COMM	Not returned
ITYPE_PATH_SYS_PRINT	Not returned
ITYPE_PATH_SYS_QUEUE	Not returned
ITYPE_PATH_SYS_SEM_M	Not returned
ITYPE_PATH_SYS_SHMEM_M	Not returned
ITYPE_PATH_SYS_MSLOT_M	Not returned
ITYPE_PATH_SYS_PIPE_M	Not returned
ITYPE_PATH_SYS_COMM_M	Not returned
ITYPE_PATH_SYS_PRINT_M	Not returned
ITYPE_PATH_SYS_QUEUE_M	Not returned
ITYPE_DEVICE_DISK	C:

Name	Example
ITYPE_DEVICE_LPT	LPT1 or LPT1: or \DEV\LPT1
ITYPE_DEVICE_COM	COM1 or COM1: or \DEV\COM1
ITYPE_DEVICE_CON	Not returned
ITYPE_DEVICE_NUL	Not returned

<7> Section 2.2.4.28: SHI1005_FLAGS_ACCESS_BASED_DIRECTORY_ENUM is only supported on servers that run Windows Server 2003 SP1.

<8> Section 2.2.4.28: SHI1005_FLAGS_ACCESS_BASED_DIRECTORY_ENUM is only supported on servers running Windows Server 2003 SP1.

<9> Section 2.2.4.30: SHARE_INFO_1501_I is only supported on Windows versions Windows 2000 or higher.

<10> Section 2.2.4.40: The server ignores this field on a **NetrServerSetInfo** operation.

<11> Section 2.2.4.40: The server ignores this field on a **NetrServerSetInfo** operation.

<12> Section 2.2.4.40: The server ignores this field on a **NetrServerSetInfo** operation and returns 3 on a **NetrServerGetInfo** operation.

<13> Section 2.2.4.40: The server ignores this field on a **NetrServerSetInfo** operation and returns 10 on a **NetrServerGetInfo** operation.

<14> Section 2.2.4.40: The server ignores this field on a **NetrServerSetInfo** operation.

<15> Section 2.2.4.40: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 1 to 0xFFFFFFFF, inclusive. The server enforces a ceiling based on the particular stock-keeping unit (SKU) running on the server by taking a minimum of specified value and the ceiling.

<16> Section 2.2.4.40: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 0 to 0xFFFFFFFF.

<17> Section 2.2.4.40: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is FALSE (0).

<18> Section 2.2.4.40: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 1 to 65535, inclusive.

<19> Section 2.2.4.40: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 0 to 65535.

<20> Section 2.2.4.40: On Windows-based systems, the server ignores this field on a **NetrServerSetInfo** operation and returns 0 on a **NetrServerGetInfo** operation.

<21> Section 2.2.4.43: The allowed range of values on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 16384, inclusive. On Windows NT 4.0, it is 1 to 2048, inclusive.

<22> Section 2.2.4.43: Only allowed value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1.

<23> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 2048, inclusive.

<24> Section 2.2.4.43: The allowed range of values for get operations on Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1024 to 65535, inclusive. On Windows NT 4.0 and Windows 2000, it is 512 to 65535, inclusive. This field is ignored by the server upon receipt for set operations.

<25> Section 2.2.4.43: The allowed range of values for get operations on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 512, inclusive. This field is ignored by the server upon receipt for set operations.

<26> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 65535, inclusive. The server enforces a ceiling based on the particular SKU that is running on the server by taking a minimum of specified value and the ceiling.

<27> Section 2.2.4.43: The allowed range of values for get operations on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 512, inclusive. This field is ignored by the server upon receipt for set operations.

<28> Section 2.2.4.43: The allowed range of values for get operations on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 11 to 50, inclusive. On Windows NT 4.0, it is 1 to 20, inclusive. This field is ignored by the server upon receipt for set operations.

<29> Section 2.2.4.43: The only allowed value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 65535. The server validates the field on receipt. The server does not store this value.

<30> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 2048, inclusive.

<31> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 2048, inclusive.

<32> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0x100000 to 0xFFFFFFFF, inclusive.

<33> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0x100000 to 0xFFFFFFFF.

<34> Section 2.2.4.43: This affects the open mode when the client does not have read/write permission to the file it is accessing. If this feature is enabled, the server uses share access (parameter to CreateFile) equal to FILE_SHARE_READ and does not mark the open as compatibility mode open; otherwise, share access is set equal to 0, and the open is marked as compatibility mode open. The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is TRUE (1).

<35> Section 2.2.4.43: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is TRUE (1).

<36> Section 2.2.4.43: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is TRUE (1). This field is ignored by the server upon receipt.

<37> Section 2.2.4.43: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is FALSE (0).

<38> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0 to 0xFFFFFFFF, inclusive. The server validates this field on receipt. The server does not store this value.

<39> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0 to 0xFFFFFFFF, inclusive. The server validates this field on receipt. The server does not store this value.

<40> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 5 to 5000, inclusive. The server validates this field on receipt. The server does not store this value.

<41> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 10 to 10000, inclusive.

<42> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 1000, inclusive. The server validates this field on receipt. The server does not store this value.

<43> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 2 to 10000, inclusive. The server validates this field on receipt. The server does not store this value.

<44> Section 2.2.4.43: Windows SMB server implementation has a timer to run periodic maintenance tasks, which include the following: a timeout for waiting opens, a timeout for oplock break requests, a check for malicious attacks, a check and a log error for a resource shortage, a cleanup for data structures that were improperly freed, a timeout for idle sessions, and an update server statistics. The sv*_scavtimeout parameter configures the frequency of this timer.

<45> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 300, inclusive.

<46> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0 to 10, inclusive.

<47> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0 to 10, inclusive.

<48> Section 2.2.4.43: The allowed range of values for get operations on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0x10000 (64 KB) to 0x1000000 (16 MB), inclusive. This field is ignored by the server upon receipt for set operations.

<49> Section 2.2.4.43: The allowed range of values for get operations on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0 to THREAD_BASE_PRIORITY_LOWRT (15), inclusive. This field is ignored by the server upon receipt for set operations.

<50> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 65535, inclusive.

<51> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 10 to 180, inclusive.

<52> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 10 to 180, inclusive.

<53> Section 2.2.4.43: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is TRUE (1).

<54> Section 2.2.4.43: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is TRUE (1).

<55> Section 2.2.4.43: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is TRUE (1).

<56> Section 2.2.4.43: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is FALSE (0).

<57> Section 2.2.4.43: The allowed range of values for Windows NT 4.0, Windows 2000, and Windows XP is 2 to 32, inclusive. The allowed range of values on Windows Server 2003, Windows Vista, and Windows Server 2008 is 2 to 1024, inclusive.

<58> Section 2.2.4.43: The allowed range of values for Windows NT 4.0, Windows 2000, and Windows XP is 2 to 100, inclusive. The allowed range of values on Windows Server 2003, Windows Vista, and Windows Server 2008 is 2 to 16384, inclusive.

<59> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 64, inclusive.

<60> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 128, inclusive.

<61> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 256, inclusive.

<62> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 2048, inclusive.

<63> Section 2.2.4.43: Windows SMB server implementation has a timer to run periodic diagnostic tasks that check the server for alert conditions. These include a check to make sure that the number of server errors and network errors have not crossed the administrator set threshold, and a check that determines the available disk space and makes sure that the disk space has not fallen below the administrator set threshold. If any of these thresholds are crossed the server notifies the alerter service and logs a failure event in service event log. The sv*_alertschedule parameter determines the frequency of this timer.

<64> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 65535, inclusive.

<65> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 65535, inclusive.

<66> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 1 to 100, inclusive.

<67> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0 to 99, inclusive.

<68> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0 to 0x100000, inclusive.

<69> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0 to 0xFFFFFFFF, inclusive.

<70> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0 to 0x100000, inclusive.

<71> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 0 to 0x100000, inclusive.

<72> Section 2.2.4.43: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is 10 to 1800, inclusive.

<73> Section 2.2.4.44: The default value on Windows is an empty string.

<74> Section 2.2.4.45: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 0 to 0xFFFFFFFF, inclusive.

<75> Section 2.2.4.46: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 0 to 0xFFFFFFFF, inclusive.

<76> Section 2.2.4.47: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is FALSE (0).

<77> Section 2.2.4.48: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 1 to 65535, inclusive.

<78> Section 2.2.4.49: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 0 to 65535, inclusive.

<79> Section 2.2.4.50: The allowed range of values on Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 1 to 16384, inclusive. On Windows NT 4.0, it is 1 to 2048, inclusive.

<80> Section 2.2.4.51: The only allowed value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 1.

<81> Section 2.2.4.52: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 1 to 2048, inclusive.

<82> Section 2.2.4.53: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 1 to 65535, inclusive.

<83> Section 2.2.4.54: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 1 to 2048, inclusive.

<84> Section 2.2.4.55: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 1 to 2048, inclusive.

<85> Section 2.2.4.56: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 0x100000 to 0xFFFFFFFF, inclusive.

<86> Section 2.2.4.57: The allowed range of values on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 is 0x100000 to 0xFFFFFFFF, inclusive.

<87> Section 2.2.4.58: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is TRUE (1).

<88> Section 2.2.4.59: The default value on Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008 is TRUE (1).

<89> Section 2.2.4.61: The default value on Windows NT 4.0 and higher is FALSE (0).

<90> Section 2.2.4.62: The allowed range of values on Windows NT 4.0 and later is 10 to 10000, inclusive.

<91> Section 2.2.4.63: The allowed range of values on Windows NT 4.0 and later is 0 to 300, inclusive.

<92> Section 2.2.4.64: The allowed range of values on Windows NT 4.0 and higher is 0 to 10, inclusive.

<93> Section 2.2.4.65: The allowed range of values on Windows NT 4.0 and higher is 0 to 10, inclusive.

<94> Section 2.2.4.66: The allowed range of values on Windows NT 4.0 and higher is 0 to 65535, inclusive.

<95> Section 2.2.4.67: The allowed range of values on Windows NT 4.0 and later is 10 to 180, inclusive.

<96> Section 2.2.4.68: The allowed range of values on Windows NT 4.0 and later is 10 to 180, inclusive.

<97> Section 2.2.4.69: The default value on Windows NT 4.0 and later is TRUE (1).

<98> Section 2.2.4.70: The default value on Windows NT 4.0 and higher is TRUE (1).

<99> Section 2.2.4.71: The default value on Windows NT 4.0 and later is TRUE (1).

<100> Section 2.2.4.72: The default value on Windows NT 4.0 and higher is FALSE (0).

<101> Section 2.2.4.73: The allowed range of values on Windows NT 4.0, Windows 2000 and Windows XP is 2 to 32, inclusive. The allowed range of values on Windows Server 2003and later is 2 to 1024, inclusive.

<102> Section 2.2.4.74: The allowed range of values on Windows NT 4.0, Windows 2000 and Windows XP is 2 to 100, inclusive. The allowed range of values on Windows Server 2003and later is 2 to 16384, inclusive.

<103> Section 2.2.4.75: The allowed range of values on Windows NT 4.0 and higher is 1 to 64, inclusive.

<104> Section 2.2.4.76: The allowed range of values on Windows NT 4.0 and higher is 1 to 128, inclusive.

<105> Section 2.2.4.77: The allowed range of values on Windows NT 4.0 and higher is 1 to 256, inclusive.

<106> Section 2.2.4.78: The allowed range of values on Windows NT 4.0 and later is 1 to 2048, inclusive.

<107> Section 2.2.4.79: The allowed range of values on Windows NT 4.0 and later is 1 to 65535, inclusive.

<108> Section 2.2.4.80: The allowed range of values on Windows NT 4.0 and later is 1 to 65535, inclusive. The particular operations whose failure causes the count of failed non-network operations to be incremented is implementation-dependent.

<109> Section 2.2.4.81: The allowed range of values on Windows NT 4.0 and later is 1 to 100, inclusive.

<110> Section 2.2.4.82: The allowed range of values on Windows NT 4.0 and later is 1 to 99, inclusive.

<111> Section 2.2.4.83: The allowed range of values on Windows NT 4.0 and later is 1 to 0x100000, inclusive.

<112> Section 2.2.4.84: The allowed range of values on Windows NT 4.0 and later is 1 to 0xFFFFFFFF, inclusive.

<113> Section 2.2.4.85: The allowed range of values on Windows NT 4.0 and later is 1 to 0x100000, inclusive.

<114> Section 2.2.4.86: The allowed range of values on Windows NT 4.0 and later is 1 to 0x100000, inclusive.

<115> Section 2.2.4.87: The allowed range of values on Windows NT 4.0 and later is 1 to 1800, inclusive.

<116> Section 2.2.4.93: Following are examples of values that this field can have for Microsoft-supported protocols:

- NETBT (NetBIOS over TCP/IP)

  On Windows 2000 and later, the format is as follows, where the value between braces is the GUID of the underlying physical interface that is generated by the operating system at installation time: \Device\NetBT_Tcpip_{2C9725F4-151A-11D3-AEEC-C3B211BD350B}

On Windows NT 4.0, the format is as follows, where DC21X41 is the name for the adapter chosen by the manufacturer: \Device\NetBT_DC21X41

- Direct hosting of SMB over TCP/IP (NetBIOS-less SMB)

  This protocol is available only on Windows 2000 and later. The format is: \Device\NetbiosSmb

- Nwlnk (the Microsoft version of the Novell IPX/SPX Protocol)

  This protocol is not installed by default. It provides the following two transports: \Device\NwlnkIpx and \Device\NwlnkNb

- NetBEUI

  This protocol is only supported on Windows 2000 and Windows NT 4.0. The value between braces is the GUID of the underlying physical port generated by the operating system at installation time. The NdisWanNbfOut/NdisWanNbfIn devices correspond to bindings between the NetBEUI transport and NDISWAN driver. The format options are:

  \Device\Nbf_{868B258E-252B-4F65-A383-18803360701F}

  \Device\Nbf_NdisWanNbfOut{77C17309-B558-4096-8A2B-2D1E9E4FC932}

  \Device\Nbf_NdisWanNbfIn{331BB986-F9B0-406C-9FA2-36425F52CC05}

<117> Section 2.2.4.93: This member is usually the NetBIOS name that the server is using, or it can represent an SMB/IPX name.

<118> Section 2.2.4.93: The server normalizes this to 16 characters by truncating the given length to this value if it is larger, or padding the transport address buffer with the blank character (0x20) until the length is 16.

<119> Section 2.2.4.93: Following are examples of values this field can have for Microsoft-supported protocols:

- NETBT (NetBIOS over TCP/IP)

  The MAC address of the n/w device, for example: 00065b5da43f

- NetBIOS over SMB

  000000000000

- Nwlnk (the Microsoft version of the Novell IPX/SPX Protocol)

  The MAC address of the n/w device, for example: 00065b5da43f

- NetBEUI

  The MAC address of the n/w device for the non NdisWan devices, for example: 00065b5da43f

For the NdisWan devices this is an index into internal connection tables of the driver. The first two characters are generated randomly using current system tick count and the next two using the current system time at installation. The last 6 characters are always 20524153 and stand for the string "RAS" including the leading blank. For example: d2e82052415

<120> Section 3.1.1: In Windows, virtual shares are implemented in DFS, which is a referral service to SMB shares, as specified in [MS-DFSC]. The DFS abstract model is specified in [MS-DFSC]. DFS is a special type of share that is relevant to the Windows client.

<121> Section 3.1.1: Windows-specific transport names are as specified in the Windows Behavior note for svti3_transportname in section 2.2.4.93.

<122> Section 3.1.1: Windows stores the list of all active shares that are identified by a share identifier in the registry, at the path HKEY_LOCAL_MACHINE|SYSTEM\CurrentControlSet\Services\lanmanserver.

<123> Section 3.1.1: The server tracks the statistics described by the **STAT_SERVER_0 (section 2.2.4.37)** structure.

<124> Section 3.1.4: Gaps in the opnum numbering sequence apply to Windows as indicated in the following table.

Opnum	Description
0	Returns ERROR_NOT_IMPLEMENTED. It is never used.
1	Returns ERROR_NOT_IMPLEMENTED. It is never used.
2	Returns ERROR_NOT_IMPLEMENTED. It is never used.
3	Returns ERROR_NOT_IMPLEMENTED. It is never used.
4	Returns ERROR_NOT_IMPLEMENTED. It is never used.
5	Returns ERROR_NOT_IMPLEMENTED. It is never used.
6	Returns ERROR_NOT_IMPLEMENTED. It is never used.
7	Returns ERROR_NOT_IMPLEMENTED. It is never used.
29	Only used locally by Windows, never remotely.
42	Only used locally by Windows, never remotely.
47	Deprecated and not defined. It is never used.

<125> Section 3.1.4: In Windows Server 2003, Windows Vista, and Windows Server 2008, messages that are discussed in section **NetrDfsGetVersion (Opnum 43) (section 3.1.4.35)** through section **NetrDfsManagerReportSiteInfo (Opnum 52) (section 3.1.4.43)** (all messages whose names begin with NetrDfs) have been deprecated. Calling them on Windows Server 2003, Windows Vista, and Windows Server 2008 returns an implementation-specific error code.

<126> Section 3.1.4: Windows implementation uses RPC protocol to retrieve the identity of the caller specified in [MS-RPCE] section 3.3.3.4.3. 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 an implementation specific error code.

<127> Section 3.1.4.1: If the *Qualifier* parameter is NULL, if it is a NULL string, or if the length of the Qualifier string, including the terminating null character, is greater than 1024, the server fails the call.

<128> Section 3.1.4.1: The SMB server identifies each connection by a connection identifier. Connection identifiers are monotonically increasing in order. The SMB server enumerates connections in increasing order of connection identifier. ResumeHandle stores the connection identifier of the last connection returned to the client. If **NetrConnectionEnum** is called with a

nonzero ResumeHandle, the server only enumerates those connections that have a connection identifier greater than the ResumeHandle.

<129> Section 3.1.4.1: The Windows implementation checks to see if the caller is a member of the Administrator, Server or Print Operator, or Power User local group.

<130> Section 3.1.4.1: If the caller is not a member of the Administrator, Server or Print Operator, or Power User local group, the method call fails with an implementation-specific error code.

<131> Section 3.1.4.2: If the BasePath parameter is not NULL and if the length of the BasePath string, including the terminating null character, is greater than 1024, the server fails the call.

<132> Section 3.1.4.2: If the UserName parameter is not NULL and if the length of the UserName string, including the terminating null character, is greater than 1024, the server fails the call.

<133> Section 3.1.4.2: The SMB server identifies each open file by a file identifier. File identifiers are monotonically increasing in order. The SMB server enumerates open files in increasing order of file identifier. *ResumeHandle* stores the file identifier of the last open file that was returned to the client. If NetrFileEnum is called with a nonzero *ResumeHandle*, the server only enumerates those connections that have a connection identifier greater than the *ResumeHandle*.

<134> Section 3.1.4.2: The Windows implementation checks to see if the caller is a member of the Administrator or Server Operator local group.

<135> Section 3.1.4.2: If the caller is not a member of the Administrator or Server Operator local group, the method call fails with an implementation-specific error code.

<136> Section 3.1.4.3: The Windows implementation checks to see if the caller is a member of the Administrator or Server Operator local group.

<137> Section 3.1.4.3: If the caller is not a member of the Administrator or Server Operator local group, the method call fails with an implementation-specific error code.

<138> Section 3.1.4.4: The Windows implementation checks to see if the caller is a member of the Administrator or Server Operator local group.

<139> Section 3.1.4.4: If the caller is not a member of the Administrator or Server Operator local group, the method call fails with an implementation-specific error code.

<140> Section 3.1.4.5: Server identifies each open session by a session identifier. Session identifiers are monotonically increasing in order. The server enumerates open files in increasing order of session identifier. *ResumeHandle* stores the session identifier of the last open session that was returned to the client. If **NetrSessionEnum** is called with a nonzero *ResumeHandle*, the server only enumerates those sessions that have a session identifier greater than the *ResumeHandle*.

<141> Section 3.1.4.5: The Windows implementation checks to see if the caller is a member of the Administrator or Server Operator local group.

<142> Section 3.1.4.5: If the caller is not a member of the Administrator or Server Operator local group, the method call fails with an implementation-specific error code.

<143> Section 3.1.4.6: The Windows implementation checks to see if the caller is a member of the Administrators or Server Operators local group.

<144> Section 3.1.4.6: If the caller is not a member of the Administrators or Server Operators local group, the method call fails with an implementation-specific error code.

<145> Section 3.1.4.7: If a member of the SHARE_INFO structure is found invalid, the server fails with an implementation-specific error code. The server performs the following validation on the structure:

- shi*_netname MUST not be NULL or an empty string and its length MUST not be greater than 80 characters.
- If Level=502 and a security descriptor is provided, it MUST be a valid security descriptor.
- If shi*_netname specifies an IPC$ or the ADMIN$ share, shi*_path MUST be NULL; otherwise, shi*_path MUST be a nonempty string that specifies a valid share path (MUST not have '.' and '..' appear as directory names).
- If shi*_netname specifies an NT path (begins with "\\?\"), shi*_type must not have a STYPE_DISKTREE flag.
- If shi*_remark is specified, its length MUST not be greater than 48.
- If shi*_ type specifies STYPE_DISKTREE flag and shi*_netname is not an ADMIN$ share, shi*_path MUST specify an absolute directory path. If the server does not support shared net drivers (determined by the SERVER_INFO field sv*_enablesharednetdrivers), the path must not be on a network drive.
- If a disk share is being added, the directory to be shared MUST exist and the caller MUST have access to it.

<146> Section 3.1.4.7: If the share being requested is a file share, the Windows implementation checks to see if the caller is a member of the Administrators, System Operators, or Power Users local group. If the share that is being requested is a printer share, the Windows implementation checks to see if the caller is a member of the Print Operators group.

<147> Section 3.1.4.7: Only members of the Administrators, System Operators, or Power Users local group can add file shares with a call to the **NetrShareAdd** method. The Print Operator can add printer shares. If this condition is not met, the server fails with an implementation-specific error code.

<148> Section 3.1.4.8: The server identifies each active share by a share identifier. Share identifiers are monotonically increasing in order. The server enumerates active shares in increasing order of share identifier. *ResumeHandle* stores the share identifier of the last active share that was returned to the client. If **NetrShareEnum** is called with a nonzero *ResumeHandle*, the server only enumerates those shares that have a share identifier greater than the *ResumeHandle*.

<149> Section 3.1.4.8: The Windows implementation checks to see if the caller is a member of the Administrator or Server Operator local group.

<150> Section 3.1.4.8: If the caller is not a member of the Administrator or Server Operator local group, the method call fails with an implementation-specific error code.

<151> Section 3.1.4.9: The server stores information about sticky shares in the Windows registry.

<152> Section 3.1.4.10: If the requested level is 2 or 502, the Windows implementation checks to see if the caller is in the Administrators, Server or Print Operators, or Power Users local group. No special group membership is required for other levels.

<153> Section 3.1.4.10: Only members of the Administrators, Server or Print Operators, or Power Users local group can successfully execute the **NetrShareGetInfo** message at level 2 or level 502.

No special group membership is required for the other levels. If this condition is not met, the server fails the call with an implementation-specific error code.

<154> Section 3.1.4.11: If Level=1005 and shi*_type do not have the flag STYPE_DISKTREE, the server fails the call by using an implementation-specific error code.

<155> Section 3.1.4.11: If a member of the **SHARE_INFO** structure is found invalid, the server fails with an implementation-specific error code. The server does the following validation on the SHARE_INFO structure:

- If shi*_type has the flag STYPE_SPECIAL then a security descriptor MUST not be specified in shi502_security_descriptor (Level = 502).
- If shi*_remark is specified its length MUST not be greater than 48.
- If Level=502 and a security descriptor is provided it MUST be a valid security descriptor

<156> Section 3.1.4.11: Windows checks to see if the caller is a member of the Administrator or Server Operator local group.

<157> Section 3.1.4.11: If the caller is not a member of the Administrator or Server Operator local group, the method call fails with an implementation-specific error code.

<158> Section 3.1.4.12: Windows-based clients set this field to an arbitrary value. The actual value does not affect server behavior because the server is required to ignore this field.

<159> Section 3.1.4.12: If the specified share is a file share, the Windows implementation checks to see if the caller is a member of the Administrators, Server Operators, or Power Users local group. If the specified share is a printer share, the Windows implementation checks to see if the caller is a member of the Print Operator group.

<160> Section 3.1.4.12: Only members of the Administrators, Server Operators, or Power Users local group can successfully delete file shares with a NetrShareDel message call. The Print Operator can delete printer shares. If the caller does not meet these requirements, the server fails the call with an implementation-specific error code.

<161> Section 3.1.4.13: Windows-based clients set this field to an arbitrary value. The actual value does not affect server behavior because the server is required to ignore this field.

<162> Section 3.1.4.13: If the specified share is a file share, the Windows implementation checks to see if the caller is a member of the Administrators, Server Operators, or Power Users local group. If the share that is specified is a printer share, the Windows implementation checks to see if the caller is a member of the Print Operator group.

<163> Section 3.1.4.13: Only members of the Administrators, Server Operators, or Power Users local group can successfully delete file shares with a **NetrShareDelSticky** message call. The Print Operator can delete printer shares. If the caller does not meet these requirements, the server fails the call with an implementation-specific error code.

<164> Section 3.1.4.14: If the specified share is a file share, the Windows implementation checks to see if the caller is a member of the Administrators, Server Operators, or Power Users local group. If the share that is specified is a printer share, the Windows implementation checks to see if the caller is a member of the Print Operator group.

<165> Section 3.1.4.14: Only members of the Administrators, Server Operators, or Power Users local group can successfully delete file shares with a NetrShareDeStart message call. The Print

Operator can delete printer shares. If the caller does not meet these requirements, the server fails the call with an implementation-specific error code.

<166> Section 3.1.4.17: If the level is 503, the Windows implementation checks to see if the caller is a member of the Administrators or Server Operators local group. If the level is 102 or 502, the Windows implementation checks to see if the caller is a member of one of the groups previously mentioned or a member of the Power Users local group.

<167> Section 3.1.4.17: If the caller is not a member of the Administrators or Server Operators local group and the level is 503, the server fails the calls with an implementation-specific error code. If the caller is not a member of one of the groups previously mentioned, not a member of the Power Users local group, and the level is 102 or 502, the server fails the calls with an implementation-specific error code.

<168> Section 3.1.4.18: This information is stored in the Windows registry.

<169> Section 3.1.4.18: If any member of the structure ServerInfo is found invalid, the server fails the call with an implementation-specific error code.

<170> Section 3.1.4.18: The Windows implementation checks to see if the client is a member of the Administrators or Server Operators local group.

<171> Section 3.1.4.18: If the client is not a member of the Administrators or Server Operators local group, the server fails the call with an implementation-specific error code.

<172> Section 3.1.4.19: Windows never passes any values in this parameter from client to server. Although defined as an [in, out] parameter, it is used only as an [out] parameter.

<173> Section 3.1.4.19: The Windows implementation checks to see if the client is a member of the Administrators or Server Operators local group.

<174> Section 3.1.4.19: If the client is not a member of the Administrators or Server Operators local group, the server fails the call with an implementation-specific error code.

<175> Section 3.1.4.20: The server returns statistics for SMB file server.

<176> Section 3.1.4.20: The Windows implementation checks to see if the client is a member of the Administrators or Server Operators local group.

<177> Section 3.1.4.20: If the client is not a member of the Administrators or Server Operators local group, the server fails the call with an implementation-specific error code.

<178> Section 3.1.4.21: No special group membership is required to successfully execute this message.

<179> Section 3.1.4.21: No special group membership is required to successfully execute this message.

<180> Section 3.1.4.23: If a member of the SERVER_TRANSPORT_INFO structure is found invalid, the server fails with an implementation-specific error code. The server performs the following validation:

- The svti*_transportname and svti*_transportaddress MUST not be NULL and svti*_transportaddresslength MUST not be zero.
- If svti*_domain is not NULL, its length MUST not be greater than 15.
- The svti*_flags can only be 0 or SVTI2_REMAP_PIPE_NAMES.

<181> Section 3.1.4.23: The Windows implementation checks to see if the client is a member of the Administrators or Server Operators local group.

<182> Section 3.1.4.23: If the client is not a member of the Administrators or Server Operators local group, the server fails the call with an implementation-specific error code.

<183> Section 3.1.4.24: The server identifies each managed transport by a transport identifier. Transport identifiers are monotonically increasing in order. The server enumerates managed transports in increasing order of transport identifier. *ResumeHandle* stores the transport identifier of the last managed transport that was returned to the client. If **NetrServerTransportEnum** is called with a nonzero *ResumeHandle*, the server only enumerates those managed transports that have a transport identifier greater than the *ResumeHandle*.

<184> Section 3.1.4.24: The Windows implementation checks to see if the client is a member of the Administrators or Server Operators local group.

<185> Section 3.1.4.24: If the client is not a member of the Administrators or Server Operators local group, the server fails the call with an implementation-specific error code.

<186> Section 3.1.4.26: Windows Vista and Windows Server 2008 return 0x00000000 even when the transport that is being deleted does not exist or has already been deleted.

<187> Section 3.1.4.26: The method NetrServerTransportDelEx is only defined on Windows XP and later.

<188> Section 3.1.4.26: The Windows implementation checks to see if the client is a member of the Administrators or Server Operators local group.

<189> Section 3.1.4.26: If the client is not a member of the Administrators or Server Operators local group, the server fails the call with an implementation-specific error code.

<190> Section 3.1.4.27: To read the owner, group, or discretionary access control list (DACL), as specified in [MS-DTYP], from the security descriptor for the specified file or directory, the DACL for the file or directory the caller MUST have READ_CONTROL access, or the caller must be the owner of the file or directory. To read the system access control list (SACL), as specified in [MS-DTYP], of a file or directory, the SE_SECURITY_NAME privilege, as specified in [MS-DTYP], must be enabled for the calling process.

<191> Section 3.1.4.27: If the caller does not meet the security measures that are specified for the Windows implementation, the server fails the call with an implementation-specific error code.

<192> Section 3.1.4.28: This message executes successfully only if the following conditions are met:

- If the owner of the object is being set, the client must have either WRITE_OWNER permission or be the owner of the object.
- If the DACL of the object is being set, the client must have either WRITE_DAC permission or be the owner of the object.
- If the SACL of the object is being set, the SE_SECURITY_NAME privilege must be enabled for the client.

<193> Section 3.1.4.28: If the server does not meet the security measures that are specified for the Windows implementation, the server fails the call with an implementation-specific error code.

<194> Section 3.1.4.29: If *PathName* is not a nonempty string or has a length greater than 260, the server fails the call with an implementation-specific error code. If the *Flag* parameter has a value other than 0 or 1, the server fails the call with an implementation-specific error code.

<195> Section 3.1.4.29: If the server cannot determine the path type for *PathName*, it fails with an implementation-specific error code.

The following are examples of paths that result in the server returning an error:

- PathName equals "\\"
- Any path ending in "\"; that is "\\server\", "\\server\<path>\", "\folder1\", "\folder1\<path>\", "C:\", "C:\<path>\"
- Invalid *PathName*
- NULL
- The number of characters in *PathName* is 0 or greater than or equal to 260

<196> Section 3.1.4.29: No security restrictions are imposed by Windows server implementations on the caller.

<197> Section 3.1.4.29: No security restrictions are imposed by Windows server implementations on the caller.

<198> Section 3.1.4.30: The Windows implementation does the following during canonicalization:

- All macros in the input file name (\., .\, \.., ..\) are removed and replaced by path components.
- Any required translations are performed on the path specification:
- Unix-style / converted to DOS-style \.
- Specific transliteration.

  Note The input case is NOT converted. The underlying file system may be case insensitive. The path is passed through, with the case exactly as presented by the caller.

- Device names (that is, namespace controlled by the server) are canonicalized by converting device names to uppercase and removing trailing colons in all but disk devices.

<199> Section 3.1.4.30: No security restrictions are imposed by Windows server implementations on the caller.

<200> Section 3.1.4.30: No security restrictions are imposed by Windows server implementations on the caller.

<201> Section 3.1.4.31: If *Flags* parameter is 1, the server ignores the *PathType* parameter.

<202> Section 3.1.4.31: The server does a string comparison on the canonicalized path names and returns the result.

<203> Section 3.1.4.31: No security restrictions are imposed by Windows server implementations on the caller.

<204> Section 3.1.4.31: No security restrictions are imposed by Windows server implementations on the caller.

<205> Section 3.1.4.32: No security restrictions are imposed by Windows server implementations on the caller.

<206> Section 3.1.4.32: No security restrictions are imposed by Windows server implementations on the caller.

<207> Section 3.1.4.33: The server simply converts the name to uppercase if (*Flags* & 0x80000000) is true except for name types that specify a password; otherwise, it just copies the input name to Outbuf. Names of type **NAMETYPE_NET** are always converted to uppercase.

<208> Section 3.1.4.33: No security restrictions are imposed by Windows server implementations on the caller.

<209> Section 3.1.4.33: No security restrictions are imposed by Windows server implementations on the caller.

<210> Section 3.1.4.34: The server does a string comparison and returns the results for all NameTypes except NAMETYPE_COMPUTER, NAMETYPE_WORKGROUP, and NAMETYPE_DOMAIN. For these, the server first converts the names to the corresponding OEM character set for the local and then does a string comparison on the resultant strings.

<211> Section 3.1.4.34: No security restrictions are imposed by Windows server implementations on the caller.

<212> Section 3.1.4.34: No security restrictions are imposed by Windows server implementations on the caller.

<213> Section 3.1.4.35: This method is not supported on Windows Server 2003, Windows Vista and Windows Server 2008 and fails with an implementation-specific error code.

<214> Section 3.1.4.35: The server always sets *Version* to zero.

<215> Section 3.1.4.35: No security restrictions are imposed by Windows server implementations on the caller.

<216> Section 3.1.4.35: No security restrictions are imposed by Windows server implementations on the caller.

<217> Section 3.1.4.36: Windows implementations use the CoCreateGuid() API to create a unique GUID. For more information on the CoCreateGuid() API, see [MSDN-CoCreateGuid].

<218> Section 3.1.4.36: This method is not supported on Windows Server 2003, Windows Vista and Windows Server 2008.

<219> Section 3.1.4.36: Both the ShortName and EntryPrefix are used to match a DFS path. If the latter does not match, but the first matches, then the server tries to use that.

<220> Section 3.1.4.36: No security restrictions are imposed by Windows server implementations on the caller.

<221> Section 3.1.4.36: No security restrictions are imposed by Windows server implementations on the caller.

<222> Section 3.1.4.37: This method is not supported on Windows Server 2003, Windows Vista and Windows Server 2008.

<223> Section 3.1.4.37: No security restrictions are imposed by Windows server implementations on the caller.

<224> Section 3.1.4.37: No security restrictions are imposed by Windows server implementations on the caller.

<225> Section 3.1.4.38: This method is not supported on Windows Server 2003, Windows Vista and Windows Server 2008.

<226> Section 3.1.4.38: No security restrictions are imposed by Windows server implementations on the caller.

<227> Section 3.1.4.38: No security restrictions are imposed by Windows server implementations on the caller.

<228> Section 3.1.4.39: Windows 2000 Server stores an 8.3 name in the ShortPrefix field. Windows Server 2003 and Windows Server 2008 use the same name for the *ShortPrefix* as for the *Prefix* field.

<229> Section 3.1.4.39: This method is not supported on Windows Server 2003 and Windows Server 2008.

<230> Section 3.1.4.39: No security restrictions are imposed by Windows server implementations on the caller.

<231> Section 3.1.4.39: No security restrictions are imposed by Windows server implementations on the caller.

<232> Section 3.1.4.40: This method is not supported on Windows Server 2003 and Windows Server 2008.

<233> Section 3.1.4.40: No security restrictions are imposed by Windows server implementations on the caller.

<234> Section 3.1.4.40: No security restrictions are imposed by Windows server implementations on the caller.

<235> Section 3.1.4.41: This method is not supported on Windows Server 2003, Windows Vista, and Windows Server 2008.

<236> Section 3.1.4.41: No security restrictions are imposed by Windows server implementations on the caller.

<237> Section 3.1.4.41: No security restrictions are imposed by Windows server implementations on the caller.

<238> Section 3.1.4.42: This method is not supported on Windows Server 2003, Windows Vista and Windows Server 2008.

<239> Section 3.1.4.42: If RelationInfo is NULL, or if its member Count is nonzero and its member Buffer is NULL, the server fails the call with an implementation-specific error code.

<240> Section 3.1.4.42: No security restrictions are imposed by Windows server implementations on the caller.

<241> Section 3.1.4.42: No security restrictions are imposed by Windows server implementations on the caller.

<242> Section 3.1.4.43: The method **NetrDfsManagerReportSiteInfo** is only defined on Windows 2000 Server and later. On other platforms, the server fails the call with an implementation-specific error code.

<243> Section 3.1.4.43: The server first tries to get the sites that are specific to the server. If that is not successful, it returns the current site and the default list. This first queries the registry key "SYSTEM\\CurrentControlSet\\Services\\DfsDriver\\CoveredSites" with the *ServerName*. If that is not successful, it queries the value "CoveredSites" for the default site list.

<244> Section 3.1.4.43: No security restrictions are imposed by Windows server implementations on the caller.

<245> Section 3.1.4.43: No security restrictions are imposed by Windows server implementations on the caller.

8 Index

M

N

T

U

V

W