[MS-WKST-Diff]:

Workstation Service Remote Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

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

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
10/22/2006	0.01	New	Version 0.01 release
1/19/2007	1.0	Major	Version 1.0 release
3/2/2007	1.1	Minor	Version 1.1 release
4/3/2007	1.2	Minor	Version 1.2 release
5/11/2007	1.3	Minor	Version 1.3 release
6/1/2007	1.3.1	Editorial	Changed language and formatting in the technical content.
7/3/2007	2.0	Major	Updated and revised the technical content.
7/20/2007	2.1	Minor	Revised technical and editorial content based on feedback.
8/10/2007	3.0	Major	Updated and revised the technical content.
9/28/2007	3.1	Minor	Revised technical and editorial content based on feedback.
10/23/2007	3.2	Minor	Made technical and editorial changes based on feedback.
11/30/2007	3.3	Minor	Made technical and editorial changes based on feedback.
1/25/2008	3.4	Minor	Clarified the meaning of the technical content.
3/14/2008	4.0	Major	Updated and revised the technical content.
5/16/2008	5.0	Major	Updated and revised the technical content.
6/20/2008	5.1	Minor	Clarified the meaning of the technical content.
7/25/2008	5.2	Minor	Clarified the meaning of the technical content.
8/29/2008	6.0	Major	Updated and revised the technical content.
10/24/2008	7.0	Major	Updated and revised the technical content.
12/5/2008	8.0	Major	Updated and revised the technical content.
1/16/2009	9.0	Major	Updated and revised the technical content.
2/27/2009	9.1	Minor	Clarified the meaning of the technical content.
4/10/2009	9.2	Minor	Clarified the meaning of the technical content.
5/22/2009	10.0	Major	Updated and revised the technical content.
7/2/2009	10.1	Minor	Clarified the meaning of the technical content.
8/14/2009	11.0	Major	Updated and revised the technical content.
9/25/2009	12.0	Major	Updated and revised the technical content.
11/6/2009	13.0	Major	Updated and revised the technical content.
12/18/2009	14.0	Major	Updated and revised the technical content.
1/29/2010	15.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
3/12/2010	16.0	Major	Updated and revised the technical content.
4/23/2010	17.0	Major	Updated and revised the technical content.
6/4/2010	17.1	Minor	Clarified the meaning of the technical content.
7/16/2010	17.2	Minor	Clarified the meaning of the technical content.
8/27/2010	17.3	Minor	Clarified the meaning of the technical content.
10/8/2010	18.0	Major	Updated and revised the technical content.
11/19/2010	18.1	Minor	Clarified the meaning of the technical content.
1/7/2011	18.2	Minor	Clarified the meaning of the technical content.
2/11/2011	19.0	Major	Updated and revised the technical content.
3/25/2011	20.0	Major	Updated and revised the technical content.
5/6/2011	21.0	Major	Updated and revised the technical content.
6/17/2011	21.1	Minor	Clarified the meaning of the technical content.
9/23/2011	21.1	None	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	22.0	Major	Updated and revised the technical content.
3/30/2012	23.0	Major	Updated and revised the technical content.
7/12/2012	23.1	Minor	Clarified the meaning of the technical content.
10/25/2012	24.0	Major	Updated and revised the technical content.
1/31/2013	24.0	None	No changes to the meaning, language, or formatting of the technical content.
8/8/2013	25.0	Major	Updated and revised the technical content.
11/14/2013	25.0	None	No changes to the meaning, language, or formatting of the technical content.
2/13/2014	25.0	None	No changes to the meaning, language, or formatting of the technical content.
5/15/2014	25.0	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	26.0	Major	Significantly changed the technical content.
10/16/2015	26.0	None	No changes to the meaning, language, or formatting of the technical content.
7/14/2016	26.1	Minor	Clarified the meaning of the technical content.
6/1/2017	26.1	None	No changes to the meaning, language, or formatting of the technical content.
9/15/2017	27.0	Major	Significantly changed the technical content.
12/1/2017	27.0	None	No changes to the meaning, language, or formatting of the

Date	Revision History	Revision Class	Comments
			technical content.
9/12/2018	28.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction 8
1.1 (Updated Section) Glossary 8
1.2 References 14
1.2.1 (Updated Section) Normative References 14
1.2.2 Informative References 15
1.3 Overview 16
1.4 (Updated Section) Relationship to Other Protocols 16
1.5 Prerequisites/Preconditions 19
1.6 Applicability Statement 19
1.7 Versioning and Capability Negotiation 19
1.8 Vendor-Extensible Fields 19
1.9 Standards Assignments 19

2 Messages 20
2.1 Transport 20
2.2 Message Syntax 20
2.2.1 Constants 20
2.2.1.1 JOIN_MAX_PASSWORD_LENGTH 20
2.2.1.2 JOIN_OBFUSCATOR_LENGTH 20
2.2.1.3 MAX_PREFERRED_LENGTH 20
2.2.2 Data Types 21
2.2.2.1 WKSSVC_IDENTIFY_HANDLE 21
2.2.2.2 WKSSVC_IMPERSONATE_HANDLE 21
2.2.2.3 handle_t 21
2.2.3 Enumerations 21
2.2.3.1 NETSETUP_JOIN_STATUS 21
2.2.3.2 NETSETUP_NAME_TYPE 22
2.2.3.3 NET_COMPUTER_NAME_TYPE 22
2.2.4 Unions 23
2.2.4.1 WKSTA_INFO 23
2.2.4.2 USE_INFO 23
2.2.5 Structures 24
2.2.5.1 WKSTA_INFO_100 24
2.2.5.2 WKSTA_INFO_101 25
2.2.5.3 WKSTA_INFO_102 25
2.2.5.4 WKSTA_INFO_502 26
2.2.5.5 WKSTA_INFO_1013 28
2.2.5.6 WKSTA_INFO_1018 28
2.2.5.7 WKSTA_INFO_1046 28
2.2.5.8 WKSTA_TRANSPORT_INFO_0 29
2.2.5.9 WKSTA_USER_INFO_0 29
2.2.5.10 WKSTA_USER_INFO_1 29
2.2.5.11 STAT_WORKSTATION_0 30
2.2.5.12 WKSTA_USER_INFO_0_CONTAINER 32
2.2.5.13 WKSTA_USER_INFO_1_CONTAINER 33
2.2.5.14 WKSTA_USER_ENUM_STRUCT 33
2.2.5.15 WKSTA_TRANSPORT_INFO_0_CONTAINER 33
2.2.5.16 WKSTA_TRANSPORT_ENUM_STRUCT 34
2.2.5.17 JOINPR_USER_PASSWORD 34
2.2.5.18 JOINPR_ENCRYPTED_USER_PASSWORD 35
2.2.5.18.1 Password Encoding 35
2.2.5.18.2 Initializing JOINPR_USER_PASSWORD 37
2.2.5.18.3 Encryption and Decryption 37
2.2.5.18.4 Password Decoding 38
2.2.5.19 UNICODE_STRING 38

2.2.5.20 NET_COMPUTER_NAME_ARRAY 39
2.2.5.21 USE_INFO_0 39
2.2.5.22 (Updated Section) USE_INFO_1 39
2.2.5.23 USE_INFO_2 41
2.2.5.24 USE_INFO_3 41
2.2.5.25 USE_INFO_0_CONTAINER 41
2.2.5.26 USE_INFO_1_CONTAINER 42
2.2.5.27 USE_INFO_2_CONTAINER 42
2.2.5.28 USE_ENUM_STRUCT 42
2.3 Directory Service Schema Elements 43

3 Protocol Details 44
3.1 wkssvc Client Details 44
3.1.1 Abstract Data Model 44
3.1.2 Timers 44
3.1.3 Initialization 44
3.1.4 Message Processing Events and Sequencing Rules 44
3.1.5 Timer Events 44
3.1.6 Other Local Events 44
3.2 wkssvc Server Details 45
3.2.1 Abstract Data Model 45
3.2.1.1 Access Control Abstract Data Model 45
3.2.1.2 Computer Name Abstract Data Model 46
3.2.1.3 OtherDomains Name Abstract Data Model 47
3.2.1.4 Transport Information Abstract Data Model 47
3.2.1.5 Mapped Abstract Data Model Elements 48
3.2.1.6 Domain Membership Abstract Data Model 48
3.2.1.6.1 Interaction with the [MS-LSAD] Data Model 49
3.2.1.7 UseEntry Information 49
3.2.1.8 Connection Information Abstract Data Model 49
3.2.2 Timers 50
3.2.3 Initialization 50
3.2.4 Message Processing Events and Sequencing Rules 51
3.2.4.1 NetrWkstaGetInfo (Opnum 0) 53
3.2.4.2 NetrWkstaSetInfo (Opnum 1) 55
3.2.4.3 NetrWkstaUserEnum (Opnum 2) 60
3.2.4.4 NetrWkstaTransportEnum (Opnum 5) 61
3.2.4.5 (Updated Section) NetrWkstaTransportAdd (Opnum 6) 63
3.2.4.6 NetrWkstaTransportDel (Opnum 7) 64
3.2.4.7 NetrUseAdd (Opnum 8) 66
3.2.4.8 NetrUseGetInfo (Opnum 9) 68
3.2.4.9 (Updated Section) NetrUseDel (Opnum 10) 71
3.2.4.10 NetrUseEnum (Opnum 11) 73
3.2.4.11 NetrWorkstationStatisticsGet (Opnum 13) 75
3.2.4.12 NetrGetJoinInformation (Opnum 20) 76
3.2.4.13 NetrJoinDomain2 (Opnum 22) 78
3.2.4.13.1 Common Message Processing 81
3.2.4.13.2 State Changes Required for Domain Join 82
3.2.4.13.3 Domain Join Specific Message Processing 83
3.2.4.13.4 Workgroup Join Specific Message Processing 87
3.2.4.14 NetrUnjoinDomain2 (Opnum 23) 87
3.2.4.15 NetrRenameMachineInDomain2 (Opnum 24) 90
3.2.4.16 NetrValidateName2 (Opnum 25) 95
3.2.4.17 NetrGetJoinableOUs2 (Opnum 26) 98
3.2.4.18 NetrAddAlternateComputerName (Opnum 27) 101
3.2.4.19 NetrRemoveAlternateComputerName (Opnum 28) 108
3.2.4.20 NetrSetPrimaryComputerName (Opnum 29) 114
3.2.4.21 NetrEnumerateComputerNames (Opnum 30) 122

3.2.4.22 Common Message Processing 125
3.2.4.22.1 Query Computer Account DN for the Local Machine....125
3.2.4.22.2 LDAP Bind 125
3.2.4.22.3 LDAP Unbind....126
3.2.4.22.4 Computer Account Update over SAMR....127
3.2.4.22.5 Update Display Name Using SAMR....128
3.2.4.22.6 StartImpersonatingClient 129
3.2.4.22.7 StopImpersonatingClient....130
3.2.5 Timer Events....130
3.2.6 Other Local Events....130
3.2.6.1 WkstaQueryOtherDomains Event 130
3.2.6.2 WkstaAddOtherDomains Event 130
3.2.6.3 Administrator Requests Redirection to Be Paused....130
3.2.6.4 Administrator Requests Redirection to Be Resumed....130
4 Protocol Examples....131
4.1 NetrWkstaGetInfo Example 131
4.2 NetrWkstaUserEnum Example....131
4.3 NetrJoinDomain2 Example....132
5 Security....135
5.1 Security Considerations for Implementers 135
5.2 Entropy Sources 135
6 Appendix A: Full IDL....136
7 (Updated Section) Appendix B: Product Behavior....144
8 Change Tracking....155
9 Index....156

1 Introduction

The Workstation Service Remote Protocol is used to perform tasks on a computer remotely on a network, including:

- Configuring properties and behavior of a Server Message Block network redirector (SMB network redirector).
- Managing domain membership and computer names.
- Gathering information, such as the number of enabled transport protocols and the number of currently logged-on users.

This protocol is based on the Remote Procedure Call (RPC) protocol [C706] [MS-RPCE].

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 (Updated Section) Glossary

This document uses the following terms:

account domain: A domain, identified by a security identifier (SID), that is the SID namespace for which a given machine is authoritative. The account domain is the same as the primary domain for a domain controller (DC) and is its default domain. For a machine that is joined to a domain, the account domain is the SID namespace defined by the local Security Accounts Manager [MS-SAMR].

Active Directory: ~~A~~ The Windows implementation of a general-purpose ~~network~~ directory service~~. Active Directory also refers to the Windows implementation of a directory service.~~, which uses LDAP as its primary access protocol. Active Directory stores information about a variety of objects in the network~~. User~~ such as user accounts, computer accounts, groups, and all related credential information used by ~~the Windows implementation of~~ Kerberos ~~are stored in Active Directory.~~[MS-KILE]. Active Directory is either deployed as Active Directory Domain Services (AD DS) or Active Directory Lightweight Directory Services (AD LDS~~). [MS-ADTS] describes both forms. For more information, see [MS-AUTHSOD] section 1.1.1.5.2, Lightweight Directory Access Protocol (LDAP) versions 2 and 3, Kerberos, and DNS~~), which are both described in [MS-ADOD]: Active Directory Protocols Overview.

active user: A user that is currently authenticated on a computer.

administrator: A user who has complete and unrestricted access to the computer or domain.

anonymous session: A session created for an anonymous user.

ASCII: The American Standard Code for Information Interchange (ASCII) is an 8-bit character-encoding scheme based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII refers to a single 8-bit ASCII character or an array of 8-bit ASCII characters with the high bit of each character set to zero.

authentication: The act of proving an identity to a server while providing key material that binds the identity to subsequent communications.

browser server: An entity that maintains or could be elected to maintain information about other servers and domains.

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

cleartext: In cryptography, cleartext is the form of a message (or data) that is transferred or stored without cryptographic protection.

client: A computer on which the remote procedure call (RPC) client is executing.

client side: The initiating end of the protocol.

computer name: The DNS or NetBIOS name.

directory service (DS): A service that stores and organizes information about a computer network's users and network shares, and that allows network administrators to manage users' access to the shares. See also Active Directory.

distinguished name (DN): In the Active Directory directory service, the unique identifier of an object in Active Directory, as described in [MS-ADTS] and [RFC2251].

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

domain: A set of users and computers sharing a common namespace and management infrastructure. At least one computer member of the set must act as a domain controller (DC) and host a member list that identifies all members of the domain, as well as optionally hosting the Active Directory service. The domain controller provides authentication of members, creating a unit of trust for its members. Each domain has an identifier that is shared among its members. For more information, see [MS-AUTHSOD] section 1.1.1.5 and [MS-ADTS].

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

domain name: A domain name or a NetBIOS name that identifies a domain.

Domain Name System (DNS): A hierarchical, distributed database that contains mappings of domain names to various types of data, such as IP addresses. DNS enables the location of computers and services by user-friendly names, and it also enables the discovery of other information stored in the database.

domain object: A unit of data storage in a domain that is maintained and made available to domain members by a domain controller (DC).

domain prefix: A security identifier (SID) of a domain without the relative identifier (RID) portion. The domain prefix refers to the issuing authority SID. For example, the domain prefix of S-1-5-21-397955417-626881126-188441444-1010 is S-1-5-21-397955417-626881126-188441444.

endpoint: A network-specific address of a remote procedure call (RPC) server process for remote procedure calls. The actual name and type of the endpoint depends on the RPC protocol sequence that is being used. For example, for RPC over TCP (RPC Protocol Sequence ncacn_ip_tcp), an endpoint might be TCP port 1025. For RPC over Server Message Block (RPC Protocol Sequence ncacn_np), an endpoint might be the name of a named pipe. For more information, see [C706].

fully qualified domain name (FQDN): In Active Directory, a fully qualified domain name (FQDN) that identifies a domain.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the GUID. See also universally unique identifier (UUID).

Group Policy: A mechanism that allows the implementer to specify managed configurations for users and computers in an Active Directory service environment.

handle: Any token that can be used to identify and access an object such as a device, file, or a window.

Interface Definition Language (IDL): The International Standards Organization (ISO) standard language for specifying the interface for remote procedure calls. For more information, see [C706] section 4.

Internet host name: The name of a host as defined in [RFC1123] section 2.1, with the extensions described in [MS-HNDS].

Lightweight Directory Access Protocol (LDAP): The primary access protocol for Active Directory. Lightweight Directory Access Protocol (LDAP) is an industry-standard protocol, established by the Internet Engineering Task Force (IETF), which allows users to query and update information in a directory service (DS), as described in [MS-ADTS]. The Lightweight Directory Access Protocol can be either version 2 [RFC1777] or version 3 [RFC3377].

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

machine account: An account that is associated with individual client or server machines in an Active Directory domain.

Microsoft Interface Definition Language (MIDL): The Microsoft implementation and extension of the OSF-DCE Interface Definition Language (IDL). MIDL can also mean the Interface Definition Language (IDL) compiler provided by Microsoft. For more information, see [MS-RPCE].

named pipe: A named, one-way, or duplex pipe for communication between a pipe server and one or more pipe clients.

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

NetBIOS: A particular network transport that is part of the LAN Manager protocol suite. NetBIOS uses a broadcast communication style that was applicable to early segmented local area networks. A protocol family including name resolution, datagram, and connection services. For more information, see [RFC1001] and [RFC1002].

NetBIOS name: A 16-byte address that is used to identify a NetBIOS resource on the network. For more information, see [RFC1001] and [RFC1002].

Netlogon: The Netlogon Remote Protocol, as specified in [MS-NRPC].

Network Data Representation (NDR): A specification that defines a mapping from Interface Definition Language (IDL) data types onto octet streams. NDR also refers to the runtime environment that implements the mapping facilities (for example, data provided to NDR). For more information, see [MS-RPCE] and [C706] section 14.

network redirector: A software component on a connected computer that handles requests for remote files and printer operations.

NT hash: An MD4- or MD5-based cryptographic hash of a clear text password. For more information, see [MS-NLMP] section 3.3.1 (NTOWFv1, NTLM v1 Authentication), for a normative definition.

opnum: An operation number or numeric identifier that is used to identify a specific remote procedure call (RPC) method or a method in an interface. For more information, see [C706] section 12.5.2.12 or [MS-RPCE].

organizational unit (OU): An Active Directory object contained within a domain, into which users, groups, computers, and other organizational units can be placed. An organizational unit provides a facility to classify and differentiate objects in a directory structure such as LDAP.

original equipment manufacturer (OEM) character: An 8-bit encoding used in MS-DOS and Windows operating systems to associate a sequence of bits with specific characters. The ASCII character set maps the letters, numerals, and specified punctuation and control characters to the numbers from 0 to 127. The term "code page" is used to refer to extensions of the ASCII character set that map specified characters and symbols to the numbers from 128 to 255. These code pages are referred to as OEM character sets. For more information, see [MSCHARSET].

original equipment manufacturer (OEM) character set: A character encoding used where the mappings between characters is dependent upon the code page configured on the machine, typically by the manufacturer.

plaintext: In cryptography, ordinary readable text before it is encrypted into ciphertext, or after it has been decrypted.

pseudo-random number generator (PRNG): An algorithm that generates values (numbers, bits, and so on) that give the appearance of being random from the point of view of any known test. If initialized with a true random value (called its "seed"), the output of a cryptographically strong PRNG will have the same resistance to guessing as a true random source.

read-only domain controller (RODC): A domain controller (DC) that does not accept originating updates. Additionally, an RODC does not perform outbound replication. An RODC cannot be the primary domain controller (PDC) for its domain.

registry: A local system-defined database in which applications and system components store and retrieve configuration data. It is a hierarchical data store with lightly typed elements that are logically stored in tree format. Applications use the registry API to retrieve, modify, or delete registry data. The data stored in the registry varies according to the version of the operating system.

remote procedure call (RPC): A communication protocol used primarily between client and server. The term has three definitions that are often used interchangeably: a runtime environment providing for communication facilities between computers (the RPC runtime); a set of request-and-response message exchanges between computers (the RPC exchange); and the single message from an RPC exchange (the RPC message). For more information, see [C706].

routable protocol: A communications protocol that allows packets to be forwarded from one network to another.

RPC protocol sequence: A character string that represents a valid combination of a remote procedure call (RPC) protocol, a network layer protocol, and a transport layer protocol, as described in [C706] and [MS-RPCE].

RPC transport: The underlying network services used by the remote procedure call (RPC) runtime for communications between network nodes. For more information, see [C706] section 2.

salt: An additional random quantity, specified as input to an encryption function that is used to increase the strength of the encryption.

schema: The set of attributes and object classes that govern the creation and update of objects.

security context: An abstract data structure that contains authorization information for a particular security principal in the form of a Token/Authorization Context (see [MS-DTYP] section 2.5.2). A server uses the authorization information in a security context to check access to requested resources. A security context also contains a key identifier that associates mutually established cryptographic keys, along with other information needed to perform secure communication with another security principal.

security descriptor: A data structure containing the security information associated with a securable object. A security descriptor identifies an object's owner by its security identifier (SID). If access control is configured for the object, its security descriptor contains a discretionary access control list (DACL) with SIDs for the security principals who are allowed or denied access. Applications use this structure to set and query an object's security status. The security descriptor is used to guard access to an object as well as to control which type of auditing takes place when the object is accessed. The security descriptor format is specified in [MS-DTYP] section 2.4.6; a string representation of security descriptors, called SDDL, is specified in [MS-DTYP] section 2.5.1.

security identifier (SID): An identifier for security principals that is used to identify an account or a group. Conceptually, the SID is composed of an account authority portion (typically a domain) and a smaller integer representing an identity relative to the account authority, termed the relative identifier (RID). The SID format is specified in [MS-DTYP] section 2.4.2; a string representation of SIDs is specified in [MS-DTYP] section 2.4.2 and [MS-AZOD] section 1.1.1.2.

security principal: An identity that can be used to regulate access to resources, as specified in [MS-AUTHSOD] section 1.1.1.1. A security principal can be a user, a computer, or a group that represents a set of users.

server: A replicating machine that sends replicated files to a partner (client). The term "server" refers to the machine acting in response to requests from partners that want to receive replicated files.

Server Message Block (SMB): A protocol that is used to request file and print services from server systems over a network. The SMB protocol extends the CIFS protocol with additional security, file, and disk management support. For more information, see [CIFS] and [MS-SMB].

server side: The receiving end of the protocol.

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

shared secret: A piece of data that is known only to the security principal and an authenticating authority; for example, a user and a domain controller. It is used to prove the principal's identity. A password is a common example of a shared secret. Also called a "secret key".

SMB connection: A transport connection between a Server Message Block (SMB) client and an SMB server. The SMB connection is assumed to provide reliable in-order message delivery semantics. An SMB connection can be established over any available SMB transport that is supported by both the SMB client and the SMB server, as specified in [MS-CIFS].

SMB session: An authenticated user connection established between an SMB client and an SMB server over an SMB connection. There can be multiple active SMB sessions over a single SMB connection. The Uid field in the SMB packet header distinguishes the various sessions.

standard user: A user that does not have administrative rights defined in its token and is a member of the users group. Users are prevented from making accidental or intentional system-wide changes but can perform normal daily computer tasks.

Stock Keeping Unit (SKU): A unique code that refers to a particular manufactured object or source of revenue. A SKU can refer to a retail product (software in a box that is sold through a channel), a subscription program (such as MSDN), or an online service (such as MSN).

Transmission Control Protocol (TCP): A protocol used with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. TCP handles keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The Unicode standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

universally unique identifier (UUID): A 128-bit value. UUIDs can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects in cross-process communication such as client and server interfaces, manager entry-point vectors, and RPC objects. UUIDs are highly likely to be unique. UUIDs are also known as globally unique identifiers (GUIDs) and these terms are used interchangeably in the Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the UUID. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the UUID.

user name: A unique name that identifies a specific user account. The user name of an account is unique among the other group names and user names within its own domain or workgroup.

UTF-16: A standard for encoding Unicode characters, defined in the Unicode standard, in which the most commonly used characters are defined as double-byte characters. Unless specified otherwise, this term refers to the UTF-16 encoding form specified in [UNICODE5.0.0/2007] section 3.9.

UTF-8: A byte-oriented standard for encoding Unicode characters, defined in the Unicode standard. Unless specified otherwise, this term refers to the UTF-8 encoding form specified in [UNICODE5.0.0/2007] section 3.9.

well-known endpoint: A preassigned, network-specific, stable address for a particular client/server instance. For more information, see [C706].

Windows Time Service (W32Time): A service that supports time synchronization against network and hardware time sources. For more information, see [WTSREF] and [MS-SNTP].

writable domain controller (writable DC): Synonymous with domain controller (DC), as distinct from an RODC.

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

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 (Updated Section) 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.

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

[FIPS186-2] FIPS PUBS, "Digital Signature Standard (DSS)", FIPS PUB 186-2, January 2000, http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2-change1.pdf

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

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

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

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

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

[MS-BRWSA] Microsoft Corporation, "Common Internet File System (CIFS) Browser Auxiliary Protocol".

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

[MS-CIFS] Microsoft Corporation, "Common Internet File System (CIFS) Protocol".

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

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

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

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

[MS-LSAT] Microsoft Corporation, "Local Security Authority (Translation Methods) Remote Protocol".

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

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

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

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Protocol Versions 2 and 3".

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

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

[NIS] Sun Microsystems, Inc., "System Administration Guide: Naming and Directory Services (DNS, NIS, and LDAP)", ~~http~~https://docs.~~sun~~oracle.com/~~app/docs/doc~~cd/E19253-01/816-4556/

[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

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

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992, http://www.ietf.org/rfc/rfc1321.txt

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

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

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

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

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

[RFC3629] Yergeau, F., "UTF-8, A Transformation Format of ISO 10646", STD 63, RFC 3629, November 2003, http://www.ietf.org/rfc/rfc3629.txt

[RFC4086] Eastlake III, D., Schiller, J., and Crokcer, S., "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005, http://www.ietf.org/rfc/rfc4086.txt

[SCHNEIER] Schneier, B., "Applied Cryptography, Second Edition", John Wiley and Sons, 1996, ISBN: 0471117099, http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471117099.html

[WTSREF] Microsoft Corporation, "Windows Time Service Technical Reference", March 2003, http://technet2.microsoft.com/WindowsServer/en/Library/a0fcd250-e5f7-41b3-b0e8-240f8236e2101033.mspx

1.2.2 Informative References

[FIPS140] FIPS PUBS, "Security Requirements for Cryptographic Modules", FIPS PUB 140, December 2002, http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[IEEE802.1X] Institute of Electrical and Electronics Engineers, "IEEE Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control", December 2004, http://ieeexplore.ieee.org/iel5/9828/30983/01438730.pdf

[MS-ADOD] Microsoft Corporation, "Active Directory Protocols Overview".

[MS-CERSOD] Microsoft Corporation, "Certificate Services Protocols Overview".

[MSFT-AUTOENROLLMENT] Microsoft Corporation, "Certificate Autoenrollment in Windows Server 2003", April 2003, http://technet.microsoft.com/en-us/library/cc778954.aspx

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

[RFC819] Su, Z.S. and Postel, J., "The Domain Naming Convention for Internet User Applications", RFC 819, August 1982, http://www.ietf.org/rfc/rfc0819.txt

[WININTERNALS] Russinovich, M., and Solomon, D., "Microsoft Windows Internals, Fourth Edition", Microsoft Press, 2005, ISBN: 0735619174.

1.3 Overview

The Workstation Service Remote Protocol is designed for remotely querying and configuring certain aspects of an SMB network redirector on a remote computer. For example, an implementer can use this protocol to query the computer name or major and minor version numbers of the operating system running on a remote computer.

An implementer can also use the protocol to configure the behavior of an SMB network redirector. For example, an implementer can use this protocol to configure the following:

- The number of seconds the SMB network redirector maintains an inactive SMB connection to a remote computer's resource before closing it.
- The number of simultaneous network commands that can be sent to the SMB network redirector.
- The number of seconds the SMB network redirector waits before disconnecting an inactive SMB session.

The protocol is also designed to enumerate all the users currently logged on to a remote computer, and to enumerate the transport protocols currently enabled for use by the SMB network redirector on a remote computer. When enumerating currently logged-on users or transport protocols, the protocol does not guarantee that all logged-on users or transport protocols are enumerated. The protocol also does not guarantee that the enumerated users or transport protocols are not duplicated.

The protocol can also be used to manage domain membership and the computer names of a computer on a network. For example, this protocol can be used to configure the following:

- The primary name of a computer
- Alternate names of a computer
- The domain membership of a computer

This is an RPC-based protocol. This protocol contains no protocol-specific state that is stored across protocol messages and only operates on state accessible through other protocols and local services. Some methods manipulate the server state and the state at a domain controller (DC) during message processing. This state is not part of this protocol but is exposed by other protocols.

This is a simple request-response protocol. For every method that the server receives, it executes the method and returns a completion. The client simply returns the completion status to the caller. Each method call is independent of any previous method call.

1.4 (Updated Section) Relationship to Other Protocols

The Workstation Service Remote Protocol is dependent on the RPC and SMB protocols for its transport. This protocol uses RPC [MS-RPCE] over named pipes, as specified in section 2.1. Named pipes in turn use SMB [MS-SMB].<1><2>

The client-side protocol relationships are illustrated in the following diagram:

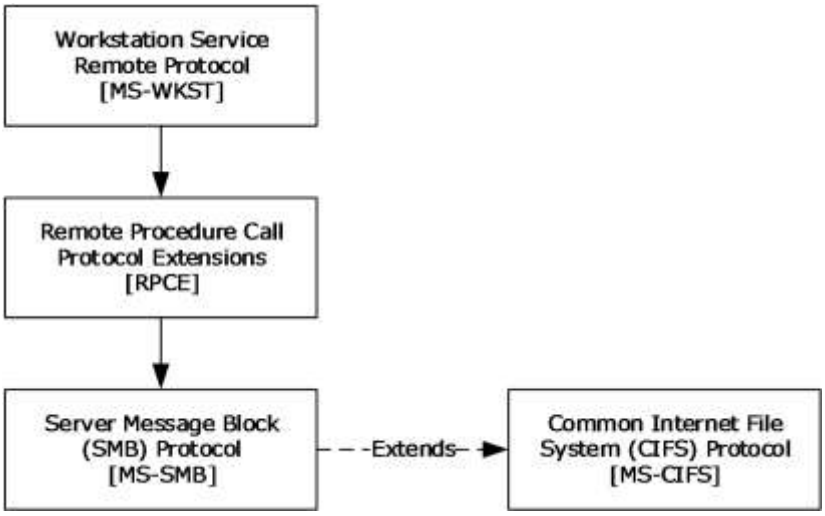

Figure 1: Client-side protocol relationships among the Workstation Service Remote Protocol and supporting protocols

The server protocol relationships are illustrated in the following diagram:

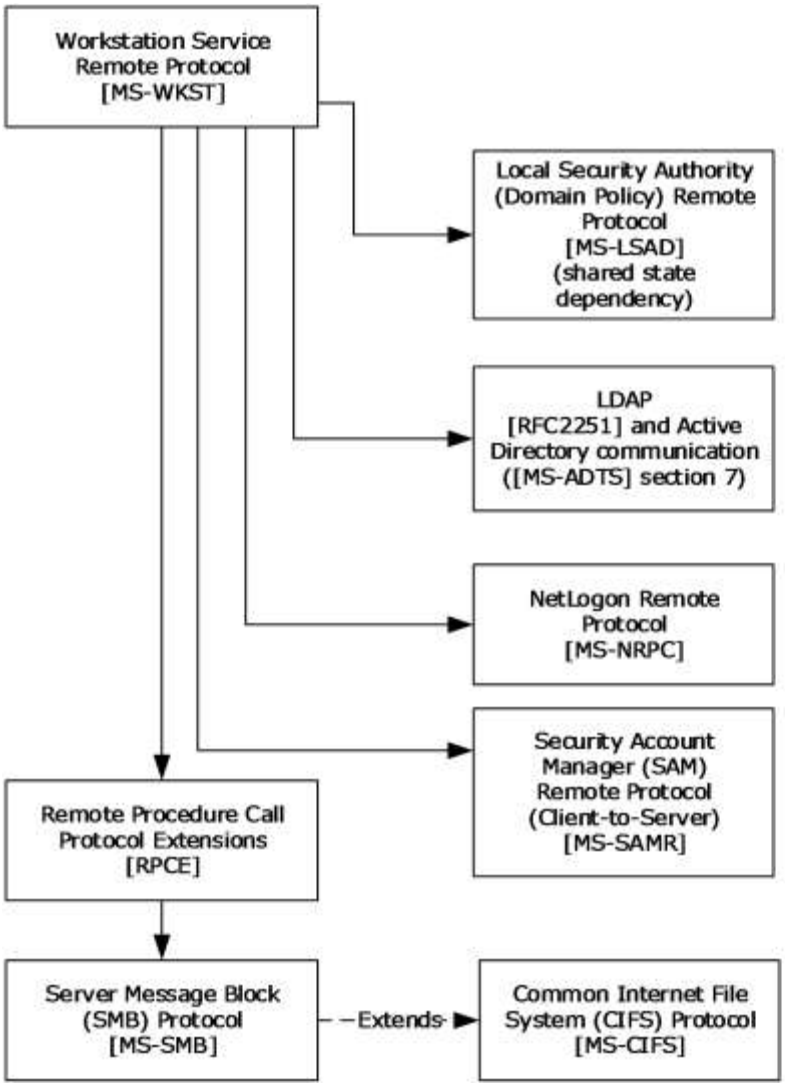

Figure 2: Server relationships among the Workstation Service Remote Protocol and supporting protocols

This protocol modifies the domain-secret ([MS-ADTS] section 6.4.1) that the Netlogon Remote Protocol [MS-NRPC] depends on. In Netlogon it is called the shared secret.

The server dependency on the Local Security Authority (Domain Policy) Remote Protocol [MS-LSAD] shown in ~~Figure 2~~the figure is a shared-state dependency resulting from [MS-WKST] depending on Access Check Algorithm Pseudocode ([MS-DTYP] section 2.5.3.2), which in turn depends on state in [MS-LSAD].

This protocol also depends on additional state that is maintained in the [MS-LSAD] protocol, as specified in section 3.2.1.6.1.

This protocol uses the Server Message Block (SMB) protocol [MS-SMB] to create SMB sessions. The implementer is required to be familiar with the SMB protocol, especially the operation of SMB during the establishment and reuse of authenticated and unauthenticated connections ([MS-SMB] section 3.2.4.2).

The server protocol invokes the domain join and unjoin tasks defined in section 3.2.4.13 and section 3.2.4.14.

The server protocol depends on LDAP [RFC2251] and [MS-ADTS] section 7 for querying and updating objects in Active Directory.

The server protocol also depends on shared ADM elements as specified in sections 3.2.1.2, 3.2.1.5, and 3.2.1.6, on read/write access to the domain-secret, as specified in [MS-ADTS] section 6.4.1, and on the data model for account representation in the domain, as defined in [MS-ADTS] section 6.4.2.

The server protocol depends on the **DsrGetDcNameEx2** method ([MS-NRPC] section 3.5.4.3.1) for DC-location functionality.

The server protocol depends on [MS-SAMR] for performing updates to the computer account (section 3.2.4.22.4).

No other protocol depends on the Workstation Service Remote Protocol.

1.5 Prerequisites/Preconditions

The Workstation Service Remote Protocol is an RPC interface and, as a result, has the prerequisites [MS-RPCE] common to RPC interfaces.

It is assumed that a Workstation Service Remote Protocol client has obtained the name of a remote computer that supports the Workstation Service Remote Protocol before this protocol is called.

The client is expected to know the names of the transport protocols that can be enabled for use by the SMB network redirector on a remote computer.

1.6 Applicability Statement

This protocol is only appropriate for querying and configuring an SMB network redirector on a remote computer or enumerating the currently logged-on users on a remote computer.

This protocol is not appropriate for enumeration of large numbers of logged-on users or transport protocols, because it provides no guarantees that those enumerations are consistent.

1.7 Versioning and Capability Negotiation

There are no versioning issues for this protocol.

1.8 Vendor-Extensible Fields

This protocol uses Win32 error codes. These values are taken from the error number space specified in [MS-ERREF]. Vendors SHOULD reuse those values with their indicated meaning.<3> Choosing any other value runs the risk of a collision in the future.

1.9 Standards Assignments

Parameter	Value	Reference
RPC Interface UUID	{6BFFD098-A112-3610-9833-46C3F87E345A}	[C706]
Pipe name	\PIPE\wkssvc	[MS-SMB]

2 Messages

2.1 Transport

The Workstation Service Remote Protocol MUST use the following RPC protocol sequence: RPC over SMB, as specified in [MS-RPCE] section 2.1.1.2.

The Workstation Service Remote Protocol MUST use the following well-known endpoint. The endpoint is the pipe name (for more information, see [PIPE]) for RPC over SMB:

\PIPE\wkssvc

The client MUST set an impersonation level for the creation of the above pipe to either IDENTIFICATION or IMPERSONATION as specified in section 2.2.2.

This is the only protocol that is supported for this endpoint.

This protocol MUST use the UUID as specified in section 1.9. The RPC version number is 1.0.

This protocol allows any user to establish a connection to the RPC server. The server 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, second bullet. The server SHOULD use this identity to perform method-specific access checks as specified in section 3.2.4.<4>

2.2 Message Syntax

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

2.2.1 Constants

2.2.1.1 JOIN_MAX_PASSWORD_LENGTH

Constant/value	Description
JOIN_MAX_PASSWORD_LENGTH 256	The size, in 16-bit characters, of the cleartext password buffer specified in a **JOINPR_USER_PASSWORD** (section 2.2.5.17) structure.

2.2.1.2 JOIN_OBFUSCATOR_LENGTH

Constant/value	Description
JOIN_OBFUSCATOR_LENGTH 8	The size, in bytes, of the unencrypted salt value in a **JOINPR_USER_PASSWORD** (section 2.2.5.17) structure.

2.2.1.3 MAX_PREFERRED_LENGTH

Constant/value	Description
MAX_PREFERRED_LENGTH	A meta value used in **NetrWkstaUserEnum** (section 3.2.4.3) and

Constant/value	Description
0xFFFFFFFF	**NetrWkstaTransportEnum** (section 3.2.4.4) method parameters to indicate that the server MUST allocate the amount of memory required to return all the requested data.

2.2.2 Data Types

2.2.2.1 WKSSVC_IDENTIFY_HANDLE

This type is declared as follows:

```
typedef [handle] wchar_t* WKSSVC_IDENTIFY_HANDLE;
```

A null-terminated Unicode string that identifies the remote computer on which to execute the method. The client MUST set the impersonation level to **SECURITY_IDENTIFICATION** ([MS-RPCE] section 2.2.1.1.10) for the RPC connection that refers to this handle.

2.2.2.2 WKSSVC_IMPERSONATE_HANDLE

This type is declared as follows:

```
typedef [handle] wchar_t* WKSSVC_IMPERSONATE_HANDLE;
```

A null-terminated Unicode string that identifies the remote computer on which to execute the method. The client MUST set the impersonation level to **SECURITY_IMPERSONATION** ([MS-RPCE] section 2.2.1.1.10) for the RPC connection that refers to this handle.

2.2.2.3 handle_t

A concrete type for an RPC binding handle ([C706] section 4.2.9.7 and [MS-DTYP] section 2.1.3). The client MUST set the impersonation level to **SECURITY_IMPERSONATION** ([MS-RPCE] section 2.2.1.1.10) for the RPC connection that refers to this handle.

2.2.3 Enumerations

2.2.3.1 NETSETUP_JOIN_STATUS

The **NETSETUP_JOIN_STATUS** enumeration contains information about the domain join status of a machine.

```
typedef  enum _NETSETUP_JOIN_STATUS
{
  NetSetupUnknownStatus = 0,
  NetSetupUnjoined,
  NetSetupWorkgroupName,
  NetSetupDomainName
} NETSETUP_JOIN_STATUS,
 *PNETSETUP_JOIN_STATUS;
```

NetSetupUnknownStatus: Domain join status of the machine is unknown.

NetSetupUnjoined: Machine is not joined to a domain or to a workgroup.

NetSetupWorkgroupName: Machine is joined to a workgroup.

NetSetupDomainName: Machine is joined to a domain.

2.2.3.2 NETSETUP_NAME_TYPE

The **NETSETUP_NAME_TYPE** enumeration specifies the types of validation that can be performed for a computer name, workgroup name, or domain_name.

```
typedef  enum _NETSETUP_NAME_TYPE
{
  NetSetupUnknown = 0,
  NetSetupMachine,
  NetSetupWorkgroup,
  NetSetupDomain,
  NetSetupNonExistentDomain,
  NetSetupDnsMachine
} NETSETUP_NAME_TYPE,
 *PNETSETUP_NAME_TYPE;
```

NetSetupUnknown: Reserved.

NetSetupMachine: Verify that the name is valid as a NetBIOS computer name and that it is not in use.

NetSetupWorkgroup: Verify that the name is valid as a workgroup name.

NetSetupDomain: Verify that the name is valid as a NetBIOS domain_name and that a domain with that name exists.

NetSetupNonExistentDomain: Verify that the name is valid as a NetBIOS domain_name and that a domain with that name does not exist.

NetSetupDnsMachine: Verify that the name is valid as a DNS computer name.

2.2.3.3 NET_COMPUTER_NAME_TYPE

The **NET_COMPUTER_NAME_TYPE** enumeration specifies the types of names that can be enumerated for a computer using the **NetrEnumerateComputerNames** (section 3.2.4.21) method.

```
typedef  enum _NET_COMPUTER_NAME_TYPE
{
  NetPrimaryComputerName = 0,
  NetAlternateComputerNames,
  NetAllComputerNames,
  NetComputerNameTypeMax
} NET_COMPUTER_NAME_TYPE,
 *PNET_COMPUTER_NAME_TYPE;
```

NetPrimaryComputerName: Query the primary name of a computer.

NetAlternateComputerNames: Query the alternate names of a computer.

NetAllComputerNames: Query all names of a computer.

NetComputerNameTypeMax: Maximum number of name types.

2.2.4 Unions

2.2.4.1 WKSTA_INFO

The **WKSTA_INFO** union contains information about a computer. This union is used by the methods **NetrWkstaGetInfo** (section 3.2.4.1) and **NetrWkstaSetInfo** (section 3.2.4.2).

```
typedef
[switch_type(unsigned long)]
union _WKSTA_INFO {
  [case(100)]
    LPWKSTA_INFO_100 WkstaInfo100;
  [case(101)]
    LPWKSTA_INFO_101 WkstaInfo101;
  [case(102)]
    LPWKSTA_INFO_102 WkstaInfo102;
  [case(502)]
    LPWKSTA_INFO_502 WkstaInfo502;
  [case(1013)]
    LPWKSTA_INFO_1013 WkstaInfo1013;
  [case(1018)]
    LPWKSTA_INFO_1018 WkstaInfo1018;
  [case(1046)]
    LPWKSTA_INFO_1046 WkstaInfo1046;
  [default]     ;
} WKSTA_INFO,
 *PWKSTA_INFO,
 *LPWKSTA_INFO;
```

WkstaInfo100: Contains information about a computer environment. For details, see section 2.2.5.1.

WkstaInfo101: Contains information about a computer environment. For details, see section 2.2.5.2.

WkstaInfo102: Contains information about a computer environment. For details, see section 2.2.5.3.

WkstaInfo502: Contains information about a computer environment. For details, see section 2.2.5.4.

WkstaInfo1013: Contains information about the state of the SMB network redirector. For details, see section 2.2.5.5

WkstaInfo1018: Contains information about the state of the SMB network redirector. For details, see section 2.2.5.6.

WkstaInfo1046: Contains information about the state of the SMB network redirector. For details, see section 2.2.5.7.

2.2.4.2 USE_INFO

The **USE_INFO** union contains information about the connection between a machine on which the workstation service is running and a shared resource. This union is used by the methods **NetrUseAdd** (section 3.2.4.7) and **NetrUseGetInfo** (section 3.2.4.8).

```
typedef
[switch_type(unsigned long)]
union _USE_INFO {
  [case(0)]
    LPUSE_INFO_0 UseInfo0;
  [case(1)]
    LPUSE_INFO_1 UseInfo1;
  [case(2)]
    LPUSE_INFO_2 UseInfo2;
```

```
  [case(3)]
    LPUSE_INFO_3 UseInfo3;
  [default]     ;
} USE_INFO,
 *PUSE_INFO,
 *LPUSE_INFO;
```

UseInfo0: Contains information about a connection. For details, see section 2.2.5.21.

UseInfo1: Contains information about a connection. For details, see section 2.2.5.22.

UseInfo2: Contains information about a connection. For details, see section 2.2.5.23.

UseInfo3: Contains information about a connection. For details, see section 2.2.5.24.

2.2.5 Structures

2.2.5.1 WKSTA_INFO_100

The **WKSTA_INFO_100** structure contains information about a computer environment, including platform-specific information, the names of the domain and the local computer, and information about the operating system.

```
typedef struct _WKSTA_INFO_100 {
  unsigned long wki100_platform_id;
  [string] wchar_t* wki100_computername;
  [string] wchar_t* wki100_langroup;
  unsigned long wki100_ver_major;
  unsigned long wki100_ver_minor;
} WKSTA_INFO_100,
 *PWKSTA_INFO_100,
 *LPWKSTA_INFO_100;
```

wki100_platform_id: Represents the type of operating system. This MUST be one of the following values.

Value	Meaning
0x0000012C	DOS. Decimal value 300.
0x00000190	OS2. Decimal value 400.
0x000001F4	Windows. Decimal value 500.
0x00000258	OSF. Decimal value 600.
0x000002BC	VMS. Decimal value 700.

wki100_computername: MUST be a null-terminated, Internet host name or NetBIOS name [RFC1001] of the local computer.

wki100_langroup: MUST be a null-terminated, fully qualified domain name (FQDN) of the domain to which the computer belongs.

wki100_ver_major: The major version number of the operating system running on the computer.

wki100_ver_minor: The minor version number of the operating system running on the computer.

2.2.5.2 WKSTA_INFO_101

The **WKSTA_INFO_101** structure contains information about a computer environment, including platform-specific information, the name of the domain and the local computer, and information about the operating system.

```
typedef struct _WKSTA_INFO_101 {
  unsigned long wki101_platform_id;
  [string] wchar_t* wki101_computername;
  [string] wchar_t* wki101_langroup;
  unsigned long wki101_ver_major;
  unsigned long wki101_ver_minor;
  [string] wchar_t* wki101_lanroot;
} WKSTA_INFO_101,
 *PWKSTA_INFO_101,
 *LPWKSTA_INFO_101;
```

wki101_platform_id: The same as *wki100_platform_id* parameter, as specified in section 2.2.5.1.

wki101_computername: MUST be a null-terminated, Internet host name or NetBIOS name [RFC1001] of the local computer.

wki101_langroup: MUST be a null-terminated, fully qualified domain name (FQDN) of the domain to which the computer belongs.

wki101_ver_major: The major version number of the operating system running on the computer.

wki101_ver_minor: The minor version number of the operating system running on the computer.

wki101_lanroot: This parameter is not used. MUST be returned as NULL by the server.

2.2.5.3 WKSTA_INFO_102

The **WKSTA_INFO_102** structure contains information about a computer environment, including platform-specific information, the name of the domain and the local computer, and information about the operating system and the number of logged-on users.

```
typedef struct _WKSTA_INFO_102 {
  unsigned long wki102_platform_id;
  [string] wchar_t* wki102_computername;
  [string] wchar_t* wki102_langroup;
  unsigned long wki102_ver_major;
  unsigned long wki102_ver_minor;
  [string] wchar_t* wki102_lanroot;
  unsigned long wki102_logged_on_users;
} WKSTA_INFO_102,
 *PWKSTA_INFO_102,
 *LPWKSTA_INFO_102;
```

wki102_platform_id: Represents the type of operating system. The values are the same as those for the *wki100_platform_id* parameter, as specified in section 2.2.5.1.

wki102_computername: MUST be a null-terminated, Internet host name or NetBIOS name [RFC1001] of the local computer.

wki102_langroup: MUST be a null-terminated, fully qualified domain name (FQDN) of the domain to which the computer belongs.

wki102_ver_major: The major version number of the operating system running on the computer.

wki102_ver_minor: The minor version number of the operating system running on the computer.

wki102_lanroot: This parameter is not used. MUST be returned as NULL by the server.

wki102_logged_on_users: The number of users who are currently active on the computer.

2.2.5.4 WKSTA_INFO_502

The **WKSTA_INFO_502** structure contains information about a computer environment.

```
typedef struct _WKSTA_INFO_502 {
  unsigned long wki502_char_wait;
  unsigned long wki502_collection_time;
  unsigned long wki502_maximum_collection_count;
  unsigned long wki502_keep_conn;
  unsigned long wki502_max_cmds;
  unsigned long wki502_sess_timeout;
  unsigned long wki502_siz_char_buf;
  unsigned long wki502_max_threads;
  unsigned long wki502_lock_quota;
  unsigned long wki502_lock_increment;
  unsigned long wki502_lock_maximum;
  unsigned long wki502_pipe_increment;
  unsigned long wki502_pipe_maximum;
  unsigned long wki502_cache_file_timeout;
  unsigned long wki502_dormant_file_limit;
  unsigned long wki502_read_ahead_throughput;
  unsigned long wki502_num_mailslot_buffers;
  unsigned long wki502_num_srv_announce_buffers;
  unsigned long wki502_max_illegal_datagram_events;
  unsigned long wki502_illegal_datagram_event_reset_frequency;
  int wki502_log_election_packets;
  int wki502_use_opportunistic_locking;
  int wki502_use_unlock_behind;
  int wki502_use_close_behind;
  int wki502_buf_named_pipes;
  int wki502_use_lock_read_unlock;
  int wki502_utilize_nt_caching;
  int wki502_use_raw_read;
  int wki502_use_raw_write;
  int wki502_use_write_raw_data;
  int wki502_use_encryption;
  int wki502_buf_files_deny_write;
  int wki502_buf_read_only_files;
  int wki502_force_core_create_mode;
  int wki502_use_512_byte_max_transfer;
} WKSTA_INFO_502,
 *PWKSTA_INFO_502,
 *LPWKSTA_INFO_502;
```

wki502_char_wait: Can be set to any value when sent and MUST be ignored on receipt.

wki502_collection_time: Can be set to any value when sent and MUST be ignored on receipt.

wki502_maximum_collection_count: Can be set to any value when sent and MUST be ignored on receipt.

wki502_keep_conn: The number of seconds the SMB network redirector maintains an inactive SMB connection to a remote computer's resource before closing it.

wki502_max_cmds: The number of simultaneous network commands that can be sent to the SMB network redirector.

wki502_sess_timeout: The number of seconds the server waits before disconnecting an inactive session.

wki502_siz_char_buf: Can be set to any value when sent and MUST be ignored on receipt.

wki502_max_threads: Can be set to any value when sent and MUST be ignored on receipt.

wki502_lock_quota: Can be set to any value when sent and MUST be ignored on receipt.

wki502_lock_increment: Can be set to any value when sent and MUST be ignored on receipt.

wki502_lock_maximum: Can be set to any value when sent and MUST be ignored on receipt.

wki502_pipe_increment: Can be set to any value when sent and MUST be ignored on receipt.

wki502_pipe_maximum: Can be set to any value when sent and MUST be ignored on receipt.

wki502_cache_file_timeout: Can be set to any value when sent and MUST be ignored on receipt.

wki502_dormant_file_limit: The maximum number of file or printer handles the SMB network redirector can continue to keep open, even after the application has closed the corresponding handle.

wki502_read_ahead_throughput: Can be set to any value when sent and MUST be ignored on receipt.

wki502_num_mailslot_buffers: Can be set to any value when sent and MUST be ignored on receipt.

wki502_num_srv_announce_buffers: Can be set to any value when sent and MUST be ignored on receipt.

wki502_max_illegal_datagram_events: Can be set to any value when sent and MUST be ignored on receipt.

wki502_illegal_datagram_event_reset_frequency: Can be set to any value when sent and MUST be ignored on receipt.

wki502_log_election_packets: Can be set to any value when sent and MUST be ignored on receipt.

wki502_use_opportunistic_locking: Can be set to any value when sent and MUST be ignored on receipt.

wki502_use_unlock_behind: Can be set to any value when sent and MUST be ignored on receipt.

wki502_use_close_behind: Can be set to any value when sent and MUST be ignored on receipt.

wki502_buf_named_pipes: Can be set to any value when sent and MUST be ignored on receipt.

wki502_use_lock_read_unlock: Can be set to any value when sent and MUST be ignored on receipt.

wki502_utilize_nt_caching: Can be set to any value when sent and MUST be ignored on receipt.

wki502_use_raw_read: Can be set to any value when sent and MUST be ignored on receipt.

wki502_use_raw_write: Can be set to any value when sent and MUST be ignored on receipt.

wki502_use_write_raw_data: Can be set to any value when sent and MUST be ignored on receipt.

wki502_use_encryption: Can be set to any value when sent and MUST be ignored on receipt.

wki502_buf_files_deny_write: Can be set to any value when sent and MUST be ignored on receipt.

wki502_buf_read_only_files: Can be set to any value when sent and MUST be ignored on receipt.

wki502_force_core_create_mode: Can be set to any value when sent and MUST be ignored on receipt.

wki502_use_512_byte_max_transfer: Can be set to any value when sent and MUST be ignored on receipt.

The **wki502_keep_conn**, **wki502_max_cmds**, **wki502_sess_timeout**, and **wki502_dormant_file_limit** fields are the only fields the server can use to configure the redirector. The server MUST store all the values and return back the existing values upon a client's request.

2.2.5.5 WKSTA_INFO_1013

The **WKSTA_INFO_1013** structure contains information about the state of the SMB network redirector.

```
typedef struct _WKSTA_INFO_1013 {
  unsigned long wki1013_keep_conn;
} WKSTA_INFO_1013,
 *PWKSTA_INFO_1013,
 *LPWKSTA_INFO_1013;
```

wki1013_keep_conn: The number of seconds the SMB network redirector maintains an inactive SMB connection to a remote computer's resource before closing it.

2.2.5.6 WKSTA_INFO_1018

The **WKSTA_INFO_1018** structure contains information about the state of the SMB network redirector.

```
typedef struct _WKSTA_INFO_1018 {
  unsigned long wki1018_sess_timeout;
} WKSTA_INFO_1018,
 *PWKSTA_INFO_1018,
 *LPWKSTA_INFO_1018;
```

wki1018_sess_timeout: The number of seconds the server MUST wait before disconnecting an inactive session.

2.2.5.7 WKSTA_INFO_1046

The **WKSTA_INFO_1046** structure contains information about the state of the SMB network redirector.

```
typedef struct _WKSTA_INFO_1046 {
  unsigned long wki1046_dormant_file_limit;
} WKSTA_INFO_1046,
 *PWKSTA_INFO_1046,
 *LPWKSTA_INFO_1046;
```

wki1046_dormant_file_limit: The maximum number of file or printer handles the SMB network redirector can continue to keep open, even after the application has closed the corresponding handle.

2.2.5.8 WKSTA_TRANSPORT_INFO_0

The **WKSTA_TRANSPORT_INFO_0** structure contains information about the network transport protocol that the SMB network redirector uses.

```
typedef struct _WKSTA_TRANSPORT_INFO_0 {
  unsigned long wkti0_quality_of_service;
  unsigned long wkti0_number_of_vcs;
  [string] wchar_t* wkti0_transport_name;
  [string] wchar_t* wkti0_transport_address;
  unsigned long wkti0_wan_ish;
} WKSTA_TRANSPORT_INFO_0,
 *PWKSTA_TRANSPORT_INFO_0,
 *LPWKSTA_TRANSPORT_INFO_0;
```

wkti0_quality_of_service: Unused. Can be set to any value when sent and MUST be ignored on receipt.

wkti0_number_of_vcs: The current number of remote connections using this transport protocol.

wkti0_transport_name: The null-terminated, implementation-specific<5> name of the device that implements the transport protocol.

wkti0_transport_address: The null-terminated, implementation-specific<6> string that represents the address of the transport protocol.

wkti0_wan_ish: MUST specify whether the transport protocol is a routable protocol. If set to TRUE, this is a routable protocol. If set to FALSE, this is not a routable protocol.

2.2.5.9 WKSTA_USER_INFO_0

The **WKSTA_USER_INFO_0** structure contains the name of a user who is currently active on the computer.

```
typedef struct _WKSTA_USER_INFO_0 {
  [string] wchar_t* wkui0_username;
} WKSTA_USER_INFO_0,
 *PWKSTA_USER_INFO_0,
 *LPWKSTA_USER_INFO_0;
```

wkui0_username: Null-terminated name of a user<7> who is currently active on the computer. Multiple users can be currently active on a computer; this is the name of any such user.

2.2.5.10 WKSTA_USER_INFO_1

The **WKSTA_USER_INFO_1** structure contains user information as it pertains to a specific computer.

```
typedef struct _WKSTA_USER_INFO_1 {
  [string] wchar_t* wkui1_username;
  [string] wchar_t* wkui1_logon_domain;
  [string] wchar_t* wkui1_oth_domains;
  [string] wchar_t* wkui1_logon_server;
} WKSTA_USER_INFO_1,
 *PWKSTA_USER_INFO_1,
```

```
 *LPWKSTA_USER_INFO_1;
```

wkui1_username: MUST specify a null-terminated name of a user who is currently active on the computer.

wkui1_logon_domain: MUST specify a null-terminated name of the domain to which the user belongs.

wkui1_oth_domains: MUST specify null-terminated, NetBIOS names of other domains browsed by the computer, according to the **OtherDomains Name Abstract Data Model** (section 3.2.1.3).

wkui1_logon_server: MUST specify a null-terminated, NetBIOS name of the server that authenticated the user.

2.2.5.11 STAT_WORKSTATION_0

The **STAT_WORKSTATION_0** structure contains statistical information about the SMB network redirector.

```
typedef struct _STAT_WORKSTATION_0 {
  LARGE_INTEGER StatisticsStartTime;
  LARGE_INTEGER BytesReceived;
  LARGE_INTEGER SmbsReceived;
  LARGE_INTEGER PagingReadBytesRequested;
  LARGE_INTEGER NonPagingReadBytesRequested;
  LARGE_INTEGER CacheReadBytesRequested;
  LARGE_INTEGER NetworkReadBytesRequested;
  LARGE_INTEGER BytesTransmitted;
  LARGE_INTEGER SmbsTransmitted;
  LARGE_INTEGER PagingWriteBytesRequested;
  LARGE_INTEGER NonPagingWriteBytesRequested;
  LARGE_INTEGER CacheWriteBytesRequested;
  LARGE_INTEGER NetworkWriteBytesRequested;
  unsigned long InitiallyFailedOperations;
  unsigned long FailedCompletionOperations;
  unsigned long ReadOperations;
  unsigned long RandomReadOperations;
  unsigned long ReadSmbs;
  unsigned long LargeReadSmbs;
  unsigned long SmallReadSmbs;
  unsigned long WriteOperations;
  unsigned long RandomWriteOperations;
  unsigned long WriteSmbs;
  unsigned long LargeWriteSmbs;
  unsigned long SmallWriteSmbs;
  unsigned long RawReadsDenied;
  unsigned long RawWritesDenied;
  unsigned long NetworkErrors;
  unsigned long Sessions;
  unsigned long FailedSessions;
  unsigned long Reconnects;
  unsigned long CoreConnects;
  unsigned long Lanman20Connects;
  unsigned long Lanman21Connects;
  unsigned long LanmanNtConnects;
  unsigned long ServerDisconnects;
  unsigned long HungSessions;
  unsigned long UseCount;
  unsigned long FailedUseCount;
  unsigned long CurrentCommands;
} STAT_WORKSTATION_0,
 *PSTAT_WORKSTATION_0,
 *LPSTAT_WORKSTATION_0;
```

StatisticsStartTime: The time that statistics collection started. The value MUST be stored as the number of seconds elapsed since 00:00:00, January 1, 1970 GMT.

BytesReceived: The total number of bytes the SMB network redirector has received.

SmbsReceived: The total number of SMB messages that the SMB network redirector has received.

PagingReadBytesRequested: If applicable to the server, it is an implementation-specific value (section 3.2.4.11); otherwise, it MUST be set to zero.

NonPagingReadBytesRequested: If applicable to the server, it is an implementation-specific value; otherwise, it MUST be set to zero.

CacheReadBytesRequested: If applicable to the server, it is an implementation-specific value; otherwise, it MUST be set to zero.

NetworkReadBytesRequested: If applicable to the server, it is an implementation-specific value; otherwise, it MUST be set to zero.

BytesTransmitted: The total number of bytes that the SMB network redirector has transmitted.

SmbsTransmitted: The total number of SMB messages that the SMB network redirector has transmitted.

PagingWriteBytesRequested: If applicable to the server, it is an implementation-specific value; otherwise, it MUST be set to zero.

NonPagingWriteBytesRequested: If applicable to the server, it is an implementation-specific value; otherwise, it MUST be set to zero.

CacheWriteBytesRequested: If applicable to the server, it is an implementation-specific value; otherwise, it MUST be set to zero.

NetworkWriteBytesRequested: If applicable to the server, it is an implementation-specific value; otherwise, it MUST be set to zero.

InitiallyFailedOperations: The total number of network operations that have failed to start.

FailedCompletionOperations: The total number of network operations that have failed to complete.

ReadOperations: The total number of read operations that the SMB network redirector has initiated.

RandomReadOperations: If applicable to the server, it is an implementation-specific value; otherwise, it MUST be set to zero.

ReadSmbs: The total number of read requests that the SMB network redirector has sent to remote computers.

LargeReadSmbs: The total number of read requests greater than twice the size of the remote computer's negotiated buffer size that the SMB network redirector has sent to remote computers.

SmallReadSmbs: The total number of read requests that are less than one-quarter the size of the remote computer's negotiated buffer size that the SMB network redirector has sent to remote computers.

WriteOperations: The total number of write operations that the SMB network redirector has initiated.

RandomWriteOperations: If applicable to the server, it is an implementation-specific value; otherwise, it MUST be set to zero.

WriteSmbs: The total number of write requests that the SMB network redirector has sent to remote computers.

LargeWriteSmbs: The total number of write requests that are greater than twice the size of the remote computer's negotiated buffer size and that the SMB network redirector has sent to remote computers.

SmallWriteSmbs: The total number of write requests that are less than one-quarter the size of the remote computer's negotiated buffer size and that the SMB network redirector has sent to remote computers, as specified in [MS-CIFS] section 3.2.4.15.

RawReadsDenied: The total number of raw read requests made by the SMB network redirector that have been denied by the remote computer. This field MAY<8> be ignored.

RawWritesDenied: The total number of raw write requests made by the SMB network redirector that have been denied by the remote computer. This field MAY<9> be ignored.

NetworkErrors: The total number of network errors that the SMB network redirector has received.

Sessions: The total number of remote SMB sessions that the SMB network redirector has established.

FailedSessions: The number of times that the SMB network redirector has attempted to create an SMB session but failed.

Reconnects: The total number of SMB connections that have failed.

CoreConnects: The total number of SMB connections to remote computers supporting the PCNET1 dialect that have succeeded ([MS-CIFS] section 3.2.4.2.2).

Lanman20Connects: The total number of SMB connections that have succeeded to remote computers supporting the LM1.2X002 dialect.

Lanman21Connects: The total number of SMB connections that have succeeded to remote computers supporting the LANMAN2.1 dialect.

LanmanNtConnects: The total number of SMB connections that have succeeded to remote computers supporting the NTLANMAN dialect.

ServerDisconnects: The number of times that a remote computer has disconnected the SMB network redirector.

HungSessions: The total number of SMB sessions that have timed out due to lack of response from the remote computer.

UseCount: The total number of SMB connections that the SMB network redirector has established.

FailedUseCount: The total number of failed SMB connections for the SMB network redirector.

CurrentCommands: The number of current requests that the SMB network redirector has completed.

2.2.5.12 WKSTA_USER_INFO_0_CONTAINER

The **WKSTA_USER_INFO_0_CONTAINER** structure contains a value that indicates the number of entries that the **NetrWkstaUserEnum** (section 3.2.4.3) method returns, as well as a pointer to the buffer.

```
typedef struct _WKSTA_USER_INFO_0_CONTAINER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] LPWKSTA_USER_INFO_0 Buffer;
} WKSTA_USER_INFO_0_CONTAINER,
 *PWKSTA_USER_INFO_0_CONTAINER,
 *LPWKSTA_USER_INFO_0_CONTAINER;
```

EntriesRead: The number of entries that the method returned.

Buffer: The names of the user accounts logged on to the remote computer.

2.2.5.13 WKSTA_USER_INFO_1_CONTAINER

The **WKSTA_USER_INFO_1_CONTAINER** structure contains a value that indicates the number of entries that the **NetrWkstaUserEnum** (section 3.2.4.3) method returns, as well as a pointer to the buffer.

```
typedef struct _WKSTA_USER_INFO_1_CONTAINER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] LPWKSTA_USER_INFO_1 Buffer;
} WKSTA_USER_INFO_1_CONTAINER,
 *PWKSTA_USER_INFO_1_CONTAINER,
 *LPWKSTA_USER_INFO_1_CONTAINER;
```

EntriesRead: The number of entries that the method returned.

Buffer: MUST specify information about the user accounts logged on to the remote computer.

2.2.5.14 WKSTA_USER_ENUM_STRUCT

The **WKSTA_USER_ENUM_STRUCT** structure is used by the **NetrWkstaUserEnum** (section 3.2.4.3) method to encapsulate the *_WKSTA_USER_ENUM_UNION* union.

```
typedef struct _WKSTA_USER_ENUM_STRUCT {
  unsigned long Level;
  [switch_is(Level)] union _WKSTA_USER_ENUM_UNION {
    [case(0)]
      LPWKSTA_USER_INFO_0_CONTAINER Level0;
    [case(1)]
      LPWKSTA_USER_INFO_1_CONTAINER Level1;
    [default]       ;
  } WkstaUserInfo;
} WKSTA_USER_ENUM_STRUCT,
 *PWKSTA_USER_ENUM_STRUCT,
 *LPWKSTA_USER_ENUM_STRUCT;
```

Level: Specifies the information level of the data and, in turn, determines the type of structure that the method returns. MUST be one of the following values.

Value	Meaning
0x00000000	Specifies the type of **WKSTA_USER_INFO_0_CONTAINER** (see section 2.2.5.12).
0x00000001	Specifies the type of **WKSTA_USER_INFO_1_CONTAINER** (see section 2.2.5.13).

WkstaUserInfo: Contains a **WKSTA_USER_INFO_0_CONTAINER** or a **WKSTA_USER_INFO_1_CONTAINER** structure as specified by the Level member.

2.2.5.15 WKSTA_TRANSPORT_INFO_0_CONTAINER

The **WKSTA_TRANSPORT_INFO_0_CONTAINER** structure is used by the **NetrWkstaTransportEnum** (section 3.2.4.4) method. This structure holds a value that specifies the

number of entries and a pointer to the base structure type **WKSTA_TRANSPORT_INFO_0** (section 2.2.5.8) returned by the method.

```
typedef struct _WKSTA_TRANSPORT_INFO_0_CONTAINER {
  unsigned long EntriesRead;
  [size_is(EntriesRead)] LPWKSTA_TRANSPORT_INFO_0 Buffer;
} WKSTA_TRANSPORT_INFO_0_CONTAINER,
 *PWKSTA_TRANSPORT_INFO_0_CONTAINER,
 *LPWKSTA_TRANSPORT_INFO_0_CONTAINER;
```

EntriesRead: Number of entries that the method call returned.

Buffer: Pointer to the array of **WKSTA_TRANSPORT_INFO_0** structures that hold information about transport protocols.

2.2.5.16 WKSTA_TRANSPORT_ENUM_STRUCT

The **WKSTA_TRANSPORT_ENUM_STRUC**T structure is used by the **NetrWkstaTransportEnum** (section 3.2.4.4) method. The *Level* parameter in the submitted structure determines the information level of the data that the method returns.

```
typedef struct _WKSTA_TRANSPORT_ENUM_STRUCT {
  unsigned long Level;
  [switch_is(Level)] union _WKSTA_TRANSPORT_ENUM_UNION {
    [case(0)]
      LPWKSTA_TRANSPORT_INFO_0_CONTAINER Level0;
    [default]       ;
  } WkstaTransportInfo;
} WKSTA_TRANSPORT_ENUM_STRUCT,
 *PWKSTA_TRANSPORT_ENUM_STRUCT,
 *LPWKSTA_TRANSPORT_ENUM_STRUCT;
```

Level: Value that specifies the data's information level.

Note MUST be set to zero.

WkstaTransportInfo: Contains a pointer to a **WKSTA_TRANSPORT_INFO_0_CONTAINER** (section 2.2.5.15) structure.

2.2.5.17 JOINPR_USER_PASSWORD

The **JOINPR_USER_PASSWORD** structure represents a decrypted password in the **Buffer** member of a **JOINPR_ENCRYPTED_USER_PASSWORD** (section 2.2.5.18) structure.

```
typedef struct _JOINPR_USER_PASSWORD {
  unsigned char Obfuscator[JOIN_OBFUSCATOR_LENGTH];
  wchar_t Buffer[JOIN_MAX_PASSWORD_LENGTH];
  unsigned long Length;
} JOINPR_USER_PASSWORD,
 *PJOINPR_USER_PASSWORD;
```

Obfuscator: An array of unsigned characters that contains a salt, which is filled with random bytes by the caller.

Buffer: A cleartext string of no more than **JOIN_MAX_PASSWORD_LENGTH** (section 2.2.1.1) UTF-16 characters in little-endian order. The start of the string MUST be **Length** number of bytes from the end of the buffer. The unused portion of the buffer contains indeterminate values.

Length: An unsigned integer, in little-endian order, that specifies the length in bytes of the cleartext string in the **Buffer** member.

2.2.5.18 JOINPR_ENCRYPTED_USER_PASSWORD

The **JOINPR_ENCRYPTED_USER_PASSWORD** structure is the container for a password during the encoding, encryption, decryption and decoding process.

```
typedef struct _JOINPR_ENCRYPTED_USER_PASSWORD {
  unsigned char Buffer[JOIN_OBFUSCATOR_LENGTH + (JOIN_MAX_PASSWORD_LENGTH * sizeof(wchar_t))
+ sizeof(unsigned long)];
} JOINPR_ENCRYPTED_USER_PASSWORD,
 *PJOINPR_ENCRYPTED_USER_PASSWORD;
```

Buffer: An array of bytes that contains a **JOINPR_USER_PASSWORD** (section 2.2.5.17) structure.

The sections that follow specify the encoding, encryption, decryption, and decoding of a password. (Encoding and encryption are performed by the client, but their explanations are included for completeness and to facilitate the reader's understanding of server message processing.) The server decrypts and decodes a **Buffer** structure to extract the cleartext password.

The encoding, encryption, decryption, and decoding of a password requires the following steps:

1. Encoding the cleartext password, as specified in section 2.2.5.18.1.
2. Initializing **JOINPR_USER_PASSWORD** with the result of step 1, as specified in section 2.2.5.18.2.
3. Initializing **JOINPR_ENCRYPTED_USER_PASSWORD.Buffer** with the encrypted result of step 2, and subsequently decrypting **JOINPR_ENCRYPTED_USER_PASSWORD.Buffer**, as specified in section 2.2.5.18.3.
4. Decoding the result of step 3, as a **JOINPR_USER_PASSWORD** structure, to recover the cleartext password, as specified in section 2.2.5.18.4.

2.2.5.18.1 Password Encoding

The implementer MUST use the following algorithm to encode the password. However, the implementer MAY use alternate data structures as long as the resulting value is the same.

First, the cleartext password represented as a Unicode string in little-endian format is encoded using the following sequence:

PasswordLength: The number of characters in the cleartext password.

EncodedPassword: A buffer of length ((*PasswordLength* + 2) * 2) bytes.

Seed: A single byte.

The buffer *EncodedPassword* MUST be initialized such that every bit is zero.

Seed MUST be equal to a nonzero value of 8 bits chosen at random.

Copy the cleartext password into the buffer *EncodedPassword* beginning at the third byte (zero-based index of 2).

The third byte (zero-based index 2) of the buffer *EncodedPassword* is set to the bitwise XOR of the existing third byte and the bitwise OR value of *Seed* combined with 0x43.

For each subsequent byte I, beginning at index 3, it MUST be set equal to the result of *EncodedPassword*[I] combined using bitwise XOR with the result of a bitwise XOR operation of *EncodedPassword*[I-1] with the value of *Seed*. This operation MUST be completed for all subsequent bytes except the last two bytes of *EncodedPassword*.

The first byte of the buffer *EncodedPassword* MUST be equal to the value of *Seed*.

The second byte of the buffer *EncodedPassword* MUST be equal to 0.

The following is an example of the preceding algorithm:

- *PasswordLength* is the number of characters in the cleartext password.
- *EncodedPassword* is a zero-initialized buffer of ((*PasswordLength* + 2) * 2) bytes.
- The *Seed* is set to a nonzero value chosen at random, 0xAB in this example.
- Copy the cleartext password (which is a Unicode string in little-endian format) into *EncodedPassword* beginning at the third byte (zero-based index of 2). In this example, the cleartext password is "PASSWORD".

Then the buffer, *EncodedPassword*, interpreted as an array of double byte characters, or wchar_t, could be represented graphically as:

0	1	2	3	4	5	6	7	8	9
0	P	A	S	S	W	O	R	D	0

Figure 3: EncodedPassword characters

Then the buffer, *EncodedPassword*, interpreted as an array of bytes, where each element is depicted as a hexadecimal 8-bit value, could be represented graphically as:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9
0x 00	0x 00	0x 50	0x 00	0x 41	0x 00	0x 53	0x 00	0x 53	0x 00	0x 57	0x 00	0x 4F	0x 00	0x 52	0x 00	0x 44	0x 00	0x 00	0x 00

Figure 4: EncodedPassword buffer

The third byte is set as follows.

EncodedPassword[2] = *EncodedPassword*[2] XOR (*Seed* OR 0x43)

Subsequent bytes, except for the last two, are set as follows:

EncodedPassword[I] = *EncodedPassword*[I] XOR (*EncodedPassword*[I-1] XOR *Seed*)

In this way, the caller communicates the *Seed* necessary for decoding Buffer at the server during message processing.

Each iteration of the encoding algorithm applied to the encoding buffer follows.

```
00 00 50 00 41 00 53 00 53 00 57 00 4F 00 52 00 44 00 00 00
00 00 BB 00 41 00 53 00 53 00 57 00 4F 00 52 00 44 00 00 00
00 00 BB 10 41 00 53 00 53 00 57 00 4F 00 52 00 44 00 00 00
00 00 BB 10 FA 00 53 00 53 00 57 00 4F 00 52 00 44 00 00 00
00 00 BB 10 FA 51 53 00 53 00 57 00 4F 00 52 00 44 00 00 00
```

```
00 00 BB 10 FA 51 A9 00 53 00 57 00 4F 00 52 00 44 00 00 00
00 00 BB 10 FA 51 A9 02 53 00 57 00 4F 00 52 00 44 00 00 00
00 00 BB 10 FA 51 A9 02 FA 00 57 00 4F 00 52 00 44 00 00 00
00 00 BB 10 FA 51 A9 02 FA 51 57 00 4F 00 52 00 44 00 00 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 00 4F 00 52 00 44 00 00 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 4F 00 52 00 44 00 00 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 00 52 00 44 00 00 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 52 00 44 00 00 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 B0 00 44 00 00 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 B0 1B 44 00 00 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 B0 1B f4 00 00 00
00 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 B0 1B f4 5F 00 00
```

Finally set the first byte equal to the *Seed* and the second byte to 0.

```
AB 00 BB 10 FA 51 A9 02 FA 51 AD 06 E2 49 B0 1B F4 5F 00 00
```

The encoding is complete. The example buffer would look like the following:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9
0x AB	0x 00	0x BB	0x 10	0x FA	0x 51	0x A9	0x 02	0x FA	0x 51	0x AD	0x 06	0x E2	0x 49	0x B0	0x 1B	0x F4	0x 5F	0x 00	0x 00

Figure 5: EncodedPassword complete

2.2.5.18.2 Initializing JOINPR_USER_PASSWORD

An *EncodedPassword* is packed into the **JOINPR_USER_PASSWORD** (section 2.2.5.17) structure as follows:

- **JOINPR_USER_PASSWORD.Obfuscator** is initialized with **JOIN_OBFUSCATOR_LENGTH** (section 2.2.1.2) bytes of random data.
- **JOINPR_USER_PASSWORD.Buffer** is initialized with the value of *EncodedPassword*. The start of the string *EncodedPassword* MUST be **JOIN_USER_PASSWORD.Length** bytes from the end of the buffer. Any remaining bits MUST be initialized with random data.
- **JOINPR_USER_PASSWORD.Length** is initialized with the number of bytes in *EncodedPassword*.

2.2.5.18.3 Encryption and Decryption

The algorithm that encrypts the **JOINPR_USER_PASSWORD** (section 2.2.5.17) structure, beginning at **JOINPR_USER_PASSWORD.Buffer** and including **JOINPR_USER_PASSWORD.Length**, is specified by the following pseudocode. **JOINPR_USER_PASSWORD.Obfuscator** MUST NOT be encrypted, because it salts the shared secret session key used for encryption and decryption.

```
CALL MD5Init(md5context)
CALL MD5Update(md5context, user-session-key, 16)
CALL MD5Update(md5context, JOINPR_USER_PASSWORD.Obfuscator, 8)
CALL MD5Final(md5context)
CALL rc4_key(rc4key, 16, md5context.digest)
CALL rc4(rc4key, 516, encrypted-buffer)
```

The **Buffer** member of **JOINPR_ENCRYPTED_USER_PASSWORD** (section 2.2.5.18) structure is initialized with the encrypted **JOINPR_USER_PASSWORD**.

The symbolic elements of the pseudocode are defined as follows:

- **MD5Init**, **MD5Update**, and **MD5Final** are predicates/functions [RFC1321].
- *md5Context* is a variable of type **MD5_CTX** [RFC1321].
- **rc4_key** and **rc4** are functions/predicates [SCHNEIER].
- *rc4key* is a variable of type **RC4_KEYSTRUCT** [SCHNEIER].
- *encrypted-buffer* is the size of **JOINPR_USER_PASSWORD.Buffer** and **JOINPR_USER_PASSWORD.Length**, which is ((**JOIN_MAX_PASSWORD_LENGTH** (section 2.2.1.1) * sizeof(wchar_t)) + sizeof(unsigned long)) bytes.
- *user-session-key* is a 16-byte value obtained from the 16-byte SMB session key, as specified in Per SMB Session ([MS-SMB] section 3.2.1.3).

2.2.5.18.4 Password Decoding

Prior to decoding, decrypt the encrypted portion of the **JOINPR_USER_PASSWORD** (section 2.2.5.17) structure.

The implementer MUST use the following algorithm to decode the password. However, the implementer MAY use alternate data structures, so long as the result is the same.

The cleartext password represented as a Unicode string is decoded using the following sequence:

PasswordLength: The number of characters in the cleartext password.

EncodedPassword: A buffer of length **JOINPR_USER_PASSWORD.Length** bytes.

Seed: A single byte.

I: An unsigned integer used to index the bytes of the buffer *EncodedPassword*.

The buffer *EncodedPassword* MUST be initialized such that every bit is zero.

PasswordLength MUST be equal to (**JOINPR_USER_PASSWORD.Length** / 2) -1.

EncodedPassword MUST be equal to the last **JOINPR_USER_PASSWORD.Length** bytes of **JOINPR_USER_PASSWORD.Buffer**.

Seed MUST be equal to the first byte of *EncodedPassword*.

I MUST be **JOINPR_USER_PASSWORD.Length** -1.

For the initial value of *I* and all preceding values of *I* > 1, *EncodedPassword*[*I*-1] is the result of *EncodedPassword*[*I*-1] combined using a bitwise **XOR** operator with the result of a bitwise **XOR** operation of *EncodedPassword*[*I*-2] with the value of Seed.

EncodedPassword[0] is the result of 0x43 combined using a bitwise **XOR** operator with the value of *Seed*.

Then the buffer beginning at *EncodedPassword*[2], interpreted as an array of double byte characters, or wchar_t, is the cleartext password.

2.2.5.19 UNICODE_STRING

The **UNICODE_STRING** structure specifies a Unicode string.

```
typedef struct _UNICODE_STRING {
```

```
  unsigned short Length;
  unsigned short MaximumLength;
  [size_is(MaximumLength / 2), length_is((Length) / 2)]
    unsigned short* Buffer;
} UNICODE_STRING,
 *PUNICODE_STRING;
```

Length: The length, in bytes, of the string pointed to by the **Buffer** member, not including the terminating null character, if any. This value MUST be a multiple of 2.

MaximumLength: The total size, in bytes, of the **Buffer**. If this value is not a multiple of 2, the server MUST decrement this value by 1. This value MUST NOT be less than **Length**.

Buffer: The Unicode UTF-8 string. If the **MaximumLength** value is greater than zero, this field MUST contain a non-null character. **Buffer** can contain a terminating null character.

2.2.5.20 NET_COMPUTER_NAME_ARRAY

The **NET_COMPUTER_NAME_ARRAY** structure specifies the number of names associated with a computer and a buffer containing the names.

```
typedef struct _NET_COMPUTER_NAME_ARRAY {
  unsigned long EntryCount;
  [size_is(EntryCount)] PUNICODE_STRING ComputerNames;
} NET_COMPUTER_NAME_ARRAY,
 *PNET_COMPUTER_NAME_ARRAY;
```

EntryCount: The number of entries that the method call returns.

ComputerNames: The names as an array of **UNICODE_STRING** (section 2.2.5.19) structures that are associated with a machine.

2.2.5.21 USE_INFO_0

The **USE_INFO_0** structure contains information about the connection between a machine on which the workstation service is running and a shared resource.

```
typedef struct _USE_INFO_0 {
  [string] wchar_t* ui0_local;
  [string] wchar_t* ui0_remote;
} USE_INFO_0,
 *PUSE_INFO_0,
 *LPUSE_INFO_0;
```

ui0_local: A pointer to a string that contains the device name (for example, drive E or LPT1) being redirected to the shared resource.

ui0_remote: A pointer to a string that contains the share name of the remote resource being accessed. The string MUST be in the following form: *servername**sharename*.

2.2.5.22 (Updated Section) USE_INFO_1

The **USE_INFO_1** structure contains information about the connection between a machine on which the workstation service is running and a shared resource. The information includes connection status and connection type.

```
typedef struct _USE_INFO_1 {
  [string] wchar_t* ui1_local;
  [string] wchar_t* ui1_remote;
  [string] wchar_t* ui1_password;
  unsigned long ui1_status;
  unsigned long ui1_asg_type;
  unsigned long ui1_refcount;
  unsigned long ui1_usecount;
} USE_INFO_1,
 *PUSE_INFO_1,
 *LPUSE_INFO_1;
```

ui1_local: A pointer to a string that contains the device name (for example, drive E or LPT1) being redirected to the shared resource.

ui1_remote: A pointer to a string that contains the share name of the remote resource being accessed. The string MUST be in the following form: *servername**sharename*.

ui1_password: A pointer to a string that contains the password needed to establish a session between a machine on which the workstation service is running and a server.

ui1_status: The current status of the connection, which MUST contain one of the following values:

Value/code	Meaning
USE_OK 0x00000000	The connection is valid.
USE_PAUSED 0x00000001	Paused by local workstation.
USE_SESSLOST 0x00000002	Disconnected.
USE_NETERR 0x00000003	A network error occurred.
USE_CONN 0x00000004	The connection is being made.
USE_RECONN 0x00000005	Reconnecting.

ui1_asg_type: The type of remote resource being accessed, which MUST contain one of the following values:

Value/code	Meaning
USE_WILDCARD 0xFFFFFFFF	Matches the type of the server's shared resources. Wildcards can be used only with the ~~NetUseAdd~~NetrUseAdd function, and only when the **ui1_local** member is NULL.
USE_DISKDEV 0x00000000	Disk device.
USE_SPOOLDEV 0x00000001	Spooled printer.
USE_CHARDEV	Serial device.

Value/code	Meaning
0x00000002	
USE_IPC 0x00000003	Inter process communication (IPC).

ui1_refcount: The number of files, directories, and other processes that can be opened on the remote resource.

ui1_usecount: The number of explicit connections (with a device name) or implicit UNC connections (without the device name) that are established with the resource.

2.2.5.23 USE_INFO_2

The **USE_INFO_2** structure contains information about the connection between a machine on which the workstation service is running and a shared resource. The information includes user name and domain name.

```
typedef struct _USE_INFO_2 {
  USE_INFO_1 ui2_useinfo;
  [string] wchar_t* ui2_username;
  [string] wchar_t* ui2_domainname;
} USE_INFO_2,
 *PUSE_INFO_2,
 *LPUSE_INFO_2;
```

ui2_useinfo: A pointer to the **USE_INFO_1** (section 2.2.5.22) structure entries returned by the method.

ui2_username: A pointer to a string that contains the name of the user who initiated the connection.

ui2_domainname: A pointer to a string that contains the domain name of the remote resource.

2.2.5.24 USE_INFO_3

The **USE_INFO_3** structure contains information about the connection between a machine on which the workstation service is running and a shared resource. The information includes user name and domain name.

```
typedef struct _USE_INFO_3 {
  USE_INFO_2 ui3_ui2;
  ULONG ui3_flags;
} USE_INFO_3,
 *PUSE_INFO_3,
 *LPUSE_INFO_3;
```

ui3_ui2: A pointer to the **USE_INFO_2** (section 2.2.5.23) structure entries returned by the method.

ui3_flags: A reserved field. The client MUST set this field to zero, and the server MUST ignore it on receipt.

2.2.5.25 USE_INFO_0_CONTAINER

The **USE_INFO_0_CONTAINER** structure contains a value that indicates the number of entries that the **NetrUseEnum** (section 3.2.4.10) method returns, as well as a pointer to the buffer.

```
typedef struct _USE_INFO_0_CONTAINER {
  unsigned long EntriesRead;
  LPUSE_INFO_0 Buffer;
} USE_INFO_0_CONTAINER,
 *PUSE_INFO_0_CONTAINER,
 *LPUSE_INFO_0_CONTAINER;
```

EntriesRead: The number of entries that the method returned.

Buffer: Information about the connection between a device and a shared resource.

2.2.5.26 USE_INFO_1_CONTAINER

The **USE_INFO_1_CONTAINER** structure contains a value that indicates the number of entries that the **NetrUseEnum** (section 3.2.4.10) method returns, as well as a pointer to the buffer.

```
typedef struct _USE_INFO_1_CONTAINER {
  unsigned long EntriesRead;
  LPUSE_INFO_1 Buffer;
} USE_INFO_1_CONTAINER,
 *PUSE_INFO_1_CONTAINER,
 *LPUSE_INFO_1_CONTAINER;
```

EntriesRead: The number of entries that the method returned.

Buffer: Information about the connection between a machine on which the workstation service is running and a shared resource.

2.2.5.27 USE_INFO_2_CONTAINER

The **USE_INFO_2_CONTAINER** structure contains a value that indicates the number of entries that the **NetrUseEnum** (section 3.2.4.10) method returns, as well as a pointer to the buffer.

```
typedef struct _USE_INFO_2_CONTAINER {
  unsigned long EntriesRead;
  LPUSE_INFO_2 Buffer;
} USE_INFO_2_CONTAINER,
 *PUSE_INFO_2_CONTAINER,
 *LPUSE_INFO_2_CONTAINER;
```

EntriesRead: The number of entries that the method returned.

Buffer: Specifies information about the connection between a machine on which the workstation service is running and a shared resource.

2.2.5.28 USE_ENUM_STRUCT

The **USE_ENUM_STRUCT** structure is used by the **NetrUseEnum** (section 3.2.4.10) method to encapsulate the _USE_ENUM_UNION union.

```
typedef struct _USE_ENUM_STRUCT {
  DWORD Level;
  [switch_is(Level)] union _USE_ENUM_UNION {
    [case(0)]
      LPUSE_INFO_0_CONTAINER Level0;
    [case(1)]
      LPUSE_INFO_1_CONTAINER Level1;
```

```
    [case(2)]
      LPUSE_INFO_2_CONTAINER Level2;
    [default]       ;
  } UseInfo;
} USE_ENUM_STRUCT,
 *PUSE_ENUM_STRUCT,
 *LPUSE_ENUM_STRUCT;
```

Level: A value that specifies the information level of the data. This parameter MUST be one of the following values.

Value	Meaning
0x00000000	The **UseInfo** buffer is of type **USE_INFO_0** (section 2.2.5.21).
0x00000001	The **UseInfo** buffer is of type **USE_INFO_1** (section 2.2.5.22).
0x00000002	The **UseInfo** buffer is of type **USE_INFO_2** (section 2.2.5.23).

UseInfo: A buffer containing any one of the **USE_INFO_0**, **USE_INFO_1**, or **USE_INFO_2** structures.

2.3 Directory Service Schema Elements

The Workstation Service Remote Protocol accesses the directory service (DS) schema classes and attributes that are listed in the following table. For the syntactic specifications of the <Class> or <Class> <Attribute> pairs, refer to [MS-ADSC], [MS-ADA1], [MS-ADA2], and [MS-ADA3].

Class	Attribute
computer ([MS-ADSC] section 2.21)	**DNSHostName** ([MS-ADA1] section 2.185) **msDS-AdditionalDnsHostName** ([MS-ADA2] section 2.207) **displayName** ([MS-ADA1] section 2.175) **objectClass** ([MS-ADA3] section 2.40)
securityPrincipal ([MS-ADSC] section 2.248)	**sAMAccountName** ([MS-ADA3] section 2.222) **displayName** ([MS-ADA1] section 2.175) **objectClass** ([MS-ADA3] section 2.40)
user ([MS-ADSC] section 2.268)	**servicePrincipalName** ([MS-ADA3] section 2.253) **unicodePwd** ([MS-ADA3] section 2.332) **userAccountControl** ([MS-ADA3] section 2.342) **displayName** ([MS-ADA1] section 2.175) **objectClass** ([MS-ADA3] section 2.40)
organizationalUnit ([MS-ADSC] section 2.217)	All

3 Protocol Details

The methods comprising this RPC interface MUST all return 0x00000000 on success, and a non-zero, implementation-specific error code on failure. Unless otherwise specified in the following sections, a server implementation of this protocol can choose any non-zero Win32 error value to signify an error condition, as discussed in section 1.8. The client side of the Workstation Service Remote Protocol MUST NOT interpret returned error codes. The client side of the 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 of whether the server side of the protocol is running in a client or server version of an operating system.

3.1 wkssvc Client Details

3.1.1 Abstract Data Model

No abstract data model is required.

3.1.2 Timers

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

3.1.3 Initialization

The client MUST create an RPC connection to the remote computer, using the details specified in section 2.1.

3.1.4 Message Processing Events and Sequencing Rules

No sequence of method calls is imposed on this protocol.

When a method completes, the values that the RPC returns MUST be returned unmodified to the upper layer.

The client MUST ignore errors that the RPC server returns and notify the application invoker of the error received in the higher layer. Otherwise, no special message processing is required on the client beyond the processing required in the underlying RPC protocol.

3.1.5 Timer Events

There are no timer events.

3.1.6 Other Local Events

There are no local events.

3.2 wkssvc Server Details

3.2.1 Abstract Data Model

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

A server implementing this RPC interface contains several logical elements: an SMB network redirector, one or more network protocol transports, a list of users (and associated domain information) who are using the server, and names that identify the server on the network.

One or more network protocol transports are associated with an SMB network redirector. A transport is a protocol that is logically the layer below the redirector and provides reliable delivery of redirector messages.

Transports can be dynamically enabled and disabled from a redirector. A transport MUST be enabled for a redirector before the redirector can transmit messages through the transport. A transport has an implementation-specific name; transport names are unique on a per-computer basis.

Users are logical entities that make use of a computer. A server maintains a list of users who are currently active on it. This is referred to as the "user list". Users can be logical members of a domain; in that case, associated with each logical user is the domain of which the user is a member.

The server data model is defined as follows:

DormantFileLimit: The maximum number of file or printer handles the SMB network redirector can hold open after the application has closed its handle.

IsWorkstationPaused: A Boolean that, if set, indicates that redirection for the printer share and serial communications devices is paused.

Keep_Connection: The number of seconds the SMB connection keeps active.

Max_Commands: The number of simultaneous network commands that can be sent to the SMB network redirector.

Platform_Id: The type of operating system running on the computer.

Session_TimeOut: The number of seconds that the server waits before disconnecting an inactive session.

UseTable: A table of use entries, indexed by **UseEntry.UserToken**, as specified in section 3.2.1.7.

Ver_Major: The major version number of the operating system running on the computer.

Ver_Minor: The minor version number of the operating system running on the computer.

3.2.1.1 Access Control Abstract Data Model

Access Rights: The access rights defined by this protocol are specified by the bit settings in the following table.

Name	Value	Informative Summary
WKSTA_NETAPI_CHANGE_CONFIG	0x1	Granted to security principals that are allowed to make changes to the state of the server during message processing. For

Name	Value	Informative Summary
		example, members of the Administrators group are granted this access right.
WKSTA_NETAPI_QUERY	0x2	Granted to security principals that are allowed to query the state of the server during message processing. For example, authenticated users are granted this access right.

NetSecurityDescriptor: A security descriptor that is used for the verification of access security during message processing. If present, this security descriptor MUST NOT be changed. Its value can be expressed as follows in Security Descriptor Description Language (SDDL) ([MS-DTYP] section 2.5.1).<10>

```
O:NSG:NSD:(A;;%x3;;;SY)(A;;%x3;;;BA)(A;;%x2;;;AU)
```

The **rights** field in each **NetSecurityDescriptor** ACE string uses the values defined in **Access Rights**. The value of the **NetSecurityDescriptor** expresses the following information: this object is owned by the network service; this object has the network service as its primary group; local system and built-in administrator accounts are granted query and change rights to this object; authenticated users are granted query rights to this object.

Method Access Control Algorithm: During message processing, the server implementing this protocol performs access security verification on the caller's identity using the following steps:

1. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the access check algorithm fails and the server MUST return an error.
2. The server MUST retrieve the client's impersonation token as specified in [MS-RPCE] section 3.3.3.4.3.1. If this operation fails, the access check algorithm fails and the server MUST continue executing at step 4, and MUST return an error.
3. The server MUST invoke the access check algorithm as specified in [MS-DTYP] section 2.5.3.2, Access Check Algorithm Pseudocode. For this protocol, the input parameters of that algorithm are mapped as follows:
   - *SecurityDescriptor*: This is the **NetSecurityDescriptor** specified previously in this section. If the security descriptor does not exist, the client is automatically granted access.
   - *Token*: This is the token of the client, as retrieved in step 2.
   - *Access Request mask*: This is specified by each method's message processing logic and MUST be one or more of the **Access Rights** specified previously in this section.
   - *Object Tree*: This parameter MUST be NULL.
   - *PrincipalSelfSubst SID*: This parameter MUST be NULL.
4. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).
5. The server MUST return the results of the algorithm (from either step 2 or 3).

3.2.1.2 Computer Name Abstract Data Model

ComputerName (Public): The domain client MUST know the name of the computer upon which it is executing, in a form that can be resolved by the underlying network infrastructure. The **ComputerName** element is conceptually the same as the "hostname" element used in other

standards; specifically, the name by which the computer can be referenced. The **ComputerName** element contains the following elements. Either or both of these elements might be present, depending on the configuration. The value of at least one of these elements MUST NOT be NULL.

ComputerName.FQDN (Public): Refers to the canonical, fully qualified DNS name of the computer, and MUST NOT be an alias (such as a CNAME in DNS) to another name.

ComputerName.NetBIOS (Public): The NetBIOS name of the computer. The **ComputerName.NetBIOS** element is used for holding the NetBIOS name of the computer, and it MUST be a Unicode UTF-8 string [RFC3629] of 15 characters or fewer. The NetBIOS name of the computer is the same as the unqualified name of the computer—for an example, see use of "simple-name" [RFC819]—if the name fits within the NetBIOS naming constraints. If the simple name does not meet the requirements of NetBIOS host names, the transformation from simple name to NetBIOS name is implementation-specific.

A host participating in this system is not required to implement NetBIOS to interact correctly with other services in the system, but a flat, unqualified name for the computer or host MUST be used. For clarity, that is referred to as the NetBIOS name, even if the implementation does not use NetBIOS.

alternate-computer-names: A list of tuples containing:

- **NetBIOS**: An alternate NetBIOS name of the computer.
- **FQDN**: An alternate Internet host name of the computer.

The list of **alternate-computer-names** MAY<11> be empty.

3.2.1.3 OtherDomains Name Abstract Data Model

OtherDomains: Specifies a list of NetBIOS names of domains browsed by the computer. Each name MUST be at most 15 characters in length and MUST NOT contain trailing spaces or NetBIOS suffix as defined in [MS-BRWS] section 2.1.1. The names in the OtherDomains list MUST be separated by spaces.

This element is shared with the Common Internet File System (CIFS) Browser Protocol [MS-BRWS] and the Common Internet File System (CIFS) Browser Auxiliary Protocol [MS-BRWSA] through the **WkstaQueryOtherDomains** event (section 3.2.6.1).

OtherDomainsInitialization: Contains the list of NetBIOS names of domains to be browsed by the computer. This list SHOULD be empty.<12>

This element is set locally by a principal with administrator privileges.

3.2.1.4 Transport Information Abstract Data Model

The server data model for transport information is defined as follows:

TransportList: A list of transports the workstation service is active on. Each transport MUST have the following properties:

Transport. Address: The address of the transport.

Transport.Name: The name of the transport.

Transport.VC_Count: The number of clients that are communicating with the server using this protocol.

Transport.Wannish: A Boolean value that indicates whether the transport is routable.

3.2.1.5 Mapped Abstract Data Model Elements

DomainNameNetBIOS: the NetBIOS name of the domain that the machine is joined to, or the name of the workgroup; this ADM element is the same as **DomainName.NetBIOS**, as defined in section 3.2.1.6.

DomainNameFQDN: the FQDN name of the domain that the machine is joined to; this ADM element is the same as **DomainName.FQDN**, as defined in section 3.2.1.6.

ComputerNameNetBIOS: the NetBIOS name of the machine; this ADM element is the same as **ComputerName.NetBIOS**, as defined in section 3.2.1.2.

ComputerNameFQDN: the FQDN name of the machine; this ADM element is the same as **ComputerName.FQDN**, as defined in section 3.2.1.2.

3.2.1.6 Domain Membership Abstract Data Model

The server must maintain the following data in a persistent store. The following elements are shared in a read-write mode with domain client administrators. These elements must be shared in a read-only mode with other protocols on the domain client unless otherwise specified.

DomainName (Public): The server MUST know the name of the domain to which it belongs. **DomainName** contains the following elements:

- **DomainName.FQDN (Public)**: The fully qualified Domain Name System (DNS) domain name.<13> For Active Directory–style domains, this is the flat NetBIOS name of the domain. When the server is not joined to a domain, **DomainName.FQDN** is set to NULL.
- **DomainName.NetBIOS (Public)**: The NetBIOS name of the domain. When the computer is not joined to a domain, **DomainName.NetBIOS** is set to the NetBIOS name of the workgroup the server is associated with

DomainSid (Public): The server MUST preserve the security identifier (SID) of the domain to which it belongs. This SID is used later as part of the authorization process. If the server has never been joined to a domain, or was previously joined and then unjoined, this value is empty.

DomainGuid (Public): The server MUST preserve the GUID of the domain to which it belongs. If the server has never been joined to a domain, or was previously joined and then unjoined, this value is empty.

ForestNameFQDN (Public): The server MUST preserve the canonical fully qualified DNS name of the forest that contains **DomainName**. If the server has never been joined to a domain, or was previously joined and then unjoined, this value is empty.

SiteName (Public): The server can retain the site that it has determined either through administrative configuration or dynamic discovery. Preserving the site name allows the domain client to use the site in the process of finding a "near" domain controller (DC) during the location process (assuming that the site of the domain client does not shift often as might be the case, for example, for a business traveler using a laptop). Domain client implementations SHOULD incorporate site awareness and preserve the name of the site.

ClientName (Public): The domain client MUST know its own name as the domain knows it. This corresponds to the sAMAccountName attribute of the object in the directory. The **ClientName** can be populated from configuration (for example, a service or machine name), or from human interaction.

Password (Public): The domain client must know the password credentials associated with the account object for **ClientName** in the directory.

The preceding elements provide the basis for how the domain client invokes the protocols used when communicating with the DC. They must be persisted in an implementation-dependent way when the domain client is not interactive. That is, if the domain client is acting on behalf of a user, it is possible to prompt the user for this information. If the domain client is acting on behalf of a service or set of services (for example, a server), the implementation must store these values in a way that allows the domain client to retrieve them.

3.2.1.6.1 Interaction with the [MS-LSAD] Data Model

If the domain client is running the [MS-LSAD] protocol, the following ADM elements (presented in section 3.2.1.6) MUST be considered to be owned by the [MS-LSAD] protocol:

Domain interaction elements	[MS-LSAD] elements
DomainName.NetBIOS	The **Name** member of the **LSAPR_POLICY_DNS_DOMAIN_INFO** ([MS-LSAD] section 2.2.4.14) structure, which represents the **DNS Domain Information** ADM element of [MS-LSAD] section 3.1.1.1.
DomainName.FQDN	The **DnsDomainName** member of the **LSAPR_POLICY_DNS_DOMAIN_INFO** structure, which represents the **DNS Domain Information** ADM element of [MS-LSAD].
DomainSid	The **Sid** member of the LSAPR_POLICY_DNS_DOMAIN_INFO structure, which represents the **DNS Domain Information** ADM element of [MS-LSAD].
DomainGuid	The **DomainGuid** member of the LSAPR_POLICY_DNS_DOMAIN_INFO structure, which represents the **DNS Domain Information** ADM element of [MS-LSAD].
ForestNameFQDN	The **DnsForestName** member of the LSAPR_POLICY_DNS_DOMAIN_INFO structure, which represents the **DNS Domain Information** ADM element of [MS-LSAD].

Furthermore, when the domain client is running the [MS-LSAD] protocol, access to these [MS-LSAD] ADM elements MUST be implemented ([MS-LSAD] section 3.1.1.10).

3.2.1.7 UseEntry Information

UseEntry.UserToken: A token that represents the identity of the client or the user that created **UseEntry**.

UseEntry.ConnectionTable: A list of connections established between a workstation and a server on behalf of a user, each entry of which is specified in the **Connection Information Abstract Data Model** (section 3.2.1.8).

3.2.1.8 Connection Information Abstract Data Model

Connection: An array of connections established between a workstation and a server.

- **Connection.local**: The device name (for example, drive E or LPT1) being redirected to the shared resource.
- **Connection.remote**: The name of the remote share.
- **Connection.status**: The current status of the connection.
- **Connection.asgtype**: The type of remote resource being accessed.

- **Connection.refcount**: The number of files, directories, and other processes that are open on the remote resource.
- **Connection.username**: The name of the user who initiated the connection.
- **Connection.usecount**: The number of explicit connections (with a local device name) or implicit UNC connections (without a local device name) that are established with the share.
- **Connection.domain**: The domain name associated with the user name.
- **Connection.context**: The context handle associated with the connection.

3.2.2 Timers

This protocol requires no timers.

3.2.3 Initialization

Section 2.1 specifies the parameters necessary to initialize the RPC protocol.

The server SHOULD initialize the **OtherDomains** abstract data model element based on the **OtherDomainsInitialization** element (section 3.2.1.3).

The server MUST enable advertising of the workstation service by invoking [MS-SRVS] section 3.1.6.12, passing SV_TYPE_WORKSTATION as the input parameter.

The initialization of the server data model (section 3.2.3) is defined as follows:

Connection: Set to empty.

DormantFileLimit: Set to an implementation-defined value.<14>

IsWorkstationPaused: Set to FALSE.

Keep_Connection: Set to an implementation-defined value.<15>

Max_Commands: Set to an implementation-defined value.<16>

Platform_Id: Set to an implementation-defined value.<17>

Session_TimeOut: Set to an implementation-defined value.<18>

UseTable: Set to empty.

Ver_Major: Set to an implementation-defined value.<19>

Ver_Minor: Set to an implementation-defined value.<20>

The server SHOULD initialize the **TransportList** abstract data model element based on the list of network interfaces on the system. The initialization of the transport information model (section 3.2.1.4) is defined as follows:

Transport.Address: Set to an implementation-defined value.<21>

Transport.Name: Set to an implementation-defined value.<22>

Transport.VC_Count: Set to an implementation-defined value.<23>

Transport.Wannish: Set to an implementation-defined value.<24>

3.2.4 Message Processing Events and Sequencing Rules

This protocol requires the following:

- The RPC runtime MUST perform a strict network data representation (NDR) data consistency check at target level 6.0 ([MS-RPCE] section 3.1.1.5.3.3).
- The RPC runtime MUST reject a NULL unique or full pointer with a nonzero conformant value ([MS-RPCE] section 3.1.1.5.3.3.1.2).

Methods that accept any of the following types of handles as parameters MUST return an implementation specific error to the caller, if the impersonation level for the RPC connection that refers to the handle is not set to **SECURITY_IDENTIFICATION** ([MS-RPCE] section 2.2.1.1.10):

- **WKSSVC_IDENTIFY_HANDLE** (section 2.2.2.1)
- **WKSSVC_IMPERSONATE_HANDLE** (section 2.2.2.2)
- **handle_t** (section 2.2.2.3)

The server SHOULD<25> enforce security measures to verify that the caller has the required permissions to execute the methods in this protocol. Specifications for determining the identity of the caller for performing an access check are in [MS-RPCE] section 3.3.3.1.3.

The following methods make up the **wkssvc** interface.

Method	Description
NetrWkstaGetInfo (section 3.2.4.1)	Returns information about the configuration of a workstation. Opnum: 0
NetrWkstaSetInfo (section 3.2.4.2)	Configures the permanent settings for a workstation. Opnum: 1
NetrWkstaUserEnum (section 3.2.4.3)	Lists information about all users currently logged on to a workstation. Opnum: 2
Opnum3NotUsedOnWire	Reserved for local use. Opnum: 3
Opnum4NotUsedOnWire	Reserved for local use. Opnum: 4
NetrWkstaTransportEnum (section 3.2.4.4)	Returns information about the settings of the network redirector. Opnum: 5
NetrWkstaTransportAdd (section 3.2.4.5)	Binds the transport to the network redirector. Opnum: 6
NetrWkstaTransportDel (section 3.2.4.6)	Unbinds the transport from the network redirector. Opnum: 7
NetrUseAdd (section 3.2.4.7)	Establishes a connection between the workstation server and an SMB server. Opnum: 8
NetrUseGetInfo (section 3.2.4.8)	Retrieves information from a remote workstation about a connection to a shared resource on an SMB server. Opnum: 9

Method	Description
NetrUseDel (section 3.2.4.9)	Disconnects the connection between the workstation server and an SMB server. Opnum: 10
NetrUseEnum (section 3.2.4.10)	Returns information about the connections between the workstation server and an SMB server. Opnum: 11
Opnum12NotUsedOnWire	Reserved for local use. Opnum: 12
NetrWorkstationStatisticsGet (section 3.2.4.11)	Retrieves workstation statistics. Opnum: 13
Opnum14NotUsedOnWire	Reserved for local use. Opnum: 14
Opnum15NotUsedOnWire	Reserved for local use. Opnum: 15
Opnum16NotUsedOnWire	Reserved for local use. Opnum: 16
Opnum17NotUsedOnWire	Reserved for local use. Opnum: 17
Opnum18NotUsedOnWire	Reserved for local use. Opnum: 18
Opnum19NotUsedOnWire	Reserved for local use. Opnum: 19
NetrGetJoinInformation (section 3.2.4.12)	Retrieves join-status information for a specified computer. Opnum: 20
Opnum21NotUsedOnWire	Reserved for local use. Opnum: 21
NetrJoinDomain2 (section 3.2.4.13)	Uses encrypted credentials to join a computer to a workgroup or domain. Opnum: 22
NetrUnjoinDomain2 (section 3.2.4.14)	Uses encrypted credentials to unjoin a computer from a workgroup or domain. Opnum: 23
NetrRenameMachineInDomain2 (section 3.2.4.15)	Uses encrypted credentials to rename a computer in a domain. Opnum: 24
NetrValidateName2 (section 3.2.4.16)	Uses encrypted credentials to verify the validity of a computer, workgroup, or domain name. Opnum: 25
NetrGetJoinableOUs2 (section 3.2.4.17)	Uses encrypted credentials to retrieve a list of organizational units (OUs) for account creation. Opnum: 26

Method	Description
NetrAddAlternateComputerName (section 3.2.4.18)	Adds an alternate name for a specified server. Opnum: 27
NetrRemoveAlternateComputerName (section 3.2.4.19)	Removes an alternate name for a specified server. Opnum: 28
NetrSetPrimaryComputerName (section 3.2.4.20)	Sets the primary computer name for a specified server. Opnum: 29
NetrEnumerateComputerNames (section 3.2.4.21)	Returns a list of computer names for a specified server. Opnum: 30

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<26> because it does not affect interoperability.

All methods MUST NOT throw exceptions.

Note All methods that establish SMB sessions in the course of message processing MUST close such sessions immediately prior to returning, by providing the server name and the security principal for each session that is being closed ([MS-SMB2] section 3.2.4.23), unless otherwise stated. All methods that establish SMB share connections in the course of message processing MUST close such share connections immediately prior to returning, by providing the server name, the share name, and each security principal that is requesting the share be closed ([MS-SMB2] section 3.2.4.22), unless otherwise stated.<27>

3.2.4.1 NetrWkstaGetInfo (Opnum 0)

The **NetrWkstaGetInfo** method returns information about the configuration of a remote computer, including the computer name and major and minor version numbers of the operating system.

```
unsigned long NetrWkstaGetInfo(
  [in, string, unique] WKSSVC_IDENTIFY_HANDLE ServerName,
  [in] unsigned long Level,
  [out, switch_is(Level)] LPWKSTA_INFO WkstaInfo
);
```

ServerName: A **WKSSVC_IDENTIFY_HANDLE** (section 2.2.2.1) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

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

Value	Meaning
0x00000064	Information to be returned is of type **WKSTA_INFO_100** structure (section 2.2.5.1).
0x00000065	Information to be returned is of type **WKSTA_INFO_101** structure (section 2.2.5.2).
0x00000066	Information to be returned is of type **WKSTA_INFO_102** structure (section 2.2.5.3).
0x000001F6	Information to be returned is of type **WKSTA_INFO_502** structure (section 2.2.5.4).

WkstaInfo: A pointer to the buffer that receives the data. The format of this data depends on the value of the *level* parameter.

Return Values: When the message processing result meets the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	The caller does not have the permissions to perform the operation.
ERROR_INVALID_LEVEL 0x0000007C	The information level is invalid.

The response of the server depends on the value of the *Level* parameter. If the *Level* parameter is not equal to one of the valid values, then the server MUST fail the call and return ERROR_INVALID_LEVEL.

The server SHOULD<28> enforce security measures to verify that the caller has the required permissions to execute this routine. If the server enforces security measures, and the caller does not have the required credentials, then the server MUST fail the call and return ERROR_ACCESS_DENIED. Specifications for determining the identity of the caller for the purpose of performing an access check are in [MS-RPCE] section 3.3.3.1.3.

If the *Level* parameter equals 0x00000064, then the server MUST fill in the **WkstaInfo100** member (**WKSTA_INFO_100** section 2.2.5.1) of the *WkstaInfo* parameter as follows:

- **wki100_computername** is set to **ComputerNameNetBIOS**
- **wki100_langroup** is set to **DomainNameFQDN**
- **wki100_platform_id** is set to **Platform_Id** (section 3.2.1)
- **wki100_ver_major** is set to **Ver_Major** (section 3.2.1)
- **wki100_ver_minor** is set to **Ver_Minor** (section 3.2.1)

If the *Level* parameter equals 0x00000065, then the server MUST fill in the **WkstaInfo101** member (**WKSTA_INFO_101** section 2.2.5.2) of the *WkstaInfo* parameter as follows:

- **wki101_computername** is set to **ComputerNameNetBIOS**
- **wki101_langroup** is set to **DomainNameFQDN**
- **wki101_platform_id** is set to **Platform_Id**
- **wki101_ver_major** is set to **Ver_Major**
- **wki101_ver_minor** is set to **Ver_Minor**
- **wki101_lanroot** is set to NULL

If the *Level* parameter equals 0x00000066, then the server MUST fill in the **WkstaInfo102** member (**WKSTA_INFO_102** section 2.2.5.3) of the *WkstaInfo* parameter as follows:

- **wki102_computername** is set to **ComputerNameNetBIOS**
- **wki102_langroup** is set to **DomainNameFQDN**

- **wki102_platform_id** is set to **Platform_Id**
- **wki102_ver_major** is set to **Ver_Major**
- **wki102_ver_minor** is set to **Ver_Minor**
- **wki102_lanroot** is set to NULL
- **wki102_logged_on_users** is set to the number of users who are currently active on the computer

If the *Level* parameter equals 0x000001F6, then the server MUST fill in the **WkstaInfo502** member (**WKSTA_INFO_502** section 2.2.5.4) of the *WkstaInfo* parameter as follows:

- **wki502_keep_conn** is set to **Keep_Connection** (section 3.2.1)
- **wki502_max_cmds** is set to **Max_Commands** (section 3.2.1)
- **wki502_sess_timeout** is set to **Session_TimeOut** (section 3.2.1)
- **wki502_dormant_file_limit** MAY<29> be set to **DormantFileLimit** (section 3.2.1)

3.2.4.2 NetrWkstaSetInfo (Opnum 1)

The **NetrWkstaSetInfo** method configures a remote computer according to the information structure passed in the call.

```
unsigned long NetrWkstaSetInfo(
  [in, string, unique] WKSSVC_IDENTIFY_HANDLE ServerName,
  [in] unsigned long Level,
  [in, switch_is(Level)] LPWKSTA_INFO WkstaInfo,
  [in, out, unique] unsigned long* ErrorParameter
);
```

ServerName: A **WKSSVC_IDENTIFY_HANDLE** (section 2.2.2.1) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

Level: The information level of the data. This parameter SHOULD be one of the following values.

Value	Meaning
0x000001F6	The *WkstaInfo* parameter points to a **WKSTA_INFO_502** (section 2.2.5.4) structure that contains information about the computer environment.
0x000003F5	The *WkstaInfo* parameter points to a **WKSTA_INFO_1013** (section 2.2.5.5) structure.
0x000003FA	The *WkstaInfo* parameter points to a **WKSTA_INFO_1018** (section 2.2.5.6) structure.
0x00000416	The *WkstaInfo* parameter points to a **WKSTA_INFO_1046** (section 2.2.5.7) structure.

WkstaInfo: A pointer to a buffer that specifies the data. The format of this data depends on the value of the *Level* parameter.

ErrorParameter: A pointer to a value that receives an unsigned 32-bit integer. This parameter is meaningful only if the method returns ERROR_INVALID_PARAMETER and *Level* is equal to one of the values specified in the preceding table.

The *ErrorParameter* value corresponds to the member of the **WKSTA_INFO** (section 2.2.4.1) structure, specified by the *WkstaInfo* parameter, which caused the ERROR_INVALID_PARAMETER error.

Return Values: When the message processing result meets the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_INVALID_PARAMETER 0x00000057	One of the function parameters is not valid.

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

On receiving the **NetrWkstaSetInfo**, if the *Level* parameter does not equal one of the valid values, then the server MUST fail the call as follows.

Note All value ranges are inclusive.

Invalid level value	Failure processing
0x00000000--0x000001F5, 0x000001F7--0x000003F4, 0x000003F6--0x000003F9, 0x000003FB--0x00000415 0x00000417--0xFFFFFFFF	The server SHOULD return ERROR_INVALID_LEVEL.<30>

Otherwise, if the Level parameter equals 0x000001F6, then the server MUST store values from the **WkstaInfo502** member (**WKSTA_INFO_502** structure) of the *WkstaInfo* parameter into elements of the abstract data model, as follows:

- **wki502_keep_conn** stored in **Keep_Connection** (section 3.2.1)
- **wki502_max_cmds** stored in **Max_Commands** (section 3.2.1)
- **wki502_sess_timeout** stored in **Session_TimeOut** (section 3.2.1)
- **wki502_dormant_file_limit** stored in **DormantFileLimit** (section 3.2.1)

If the *Level* parameter equals 0x000003F5, then the server MUST store values from the **WkstaInfo1013** member (**WKSTA_INFO_1013** structure) of the *WkstaInfo* parameter into elements of the abstract data model, as follows:

- **wki1013_keep_conn** stored in **Keep_Connection**.

If the *Level* parameter equals 0x000003FA, then the server MUST store values from the **WkstaInfo1018** member (**WKSTA_INFO_1018** structure) of the *WkstaInfo* parameter into elements of the abstract data model, as follows:

- **wki1018_sess_timeout** stored in **Session_TimeOut**.

If the *Level* parameter equals 0x00000416, then the server MUST store values from the **WkstaInfo1046** member (**WKSTA_INFO_1046** structure) of the *WkstaInfo* parameter into elements of the abstract data model, as follows:

- **wki1046_dormant_file_limit** stored in **DormantFileLimit**.

The server MUST validate the values stored from members of **WKSTA_INFO** structures specified by the *WkstaInfo* parameter. If this validation fails, a value SHOULD be returned in the *ErrorParameter* parameter according to the following table.<31>

For Level value 0x000001F6

Member	Valid Range	ErrorParameter Value Returned
wki502_char_wait	This field is not used. The sender SHOULD initialize it to any value between 0-65535. The receiver SHOULD ignore this field.	0x0000000A
wki502_collection_time	This field is not used. The sender SHOULD initialize it to any value between 0-65535000. The receiver SHOULD ignore this field.	0x0000000B
wki502_maximum_collection_count	This field is not used. The sender SHOULD initialize it to any value between 0-65535. The receiver SHOULD ignore this field.	0x0000000C
wki502_keep_conn	1 - 65535	0x0000000D
wki502_max_cmds	50 - 65535	0x00000000
wki502_sess_timeout	60 - 65535	0x00000012
wki502_siz_char_buf	This field is not used. The sender SHOULD initialize it to any value between 64-4096. The receiver SHOULD ignore this field.	0x00000017
wki502_max_threads	This field is not used. The sender SHOULD initialize it to any value between 1-256. The receiver SHOULD ignore this field.	0x00000021
wki502_lock_quota	This field is not used. The sender SHOULD initialize it to any value between 0-0xFFFFFFFF. The receiver SHOULD ignore this field.	0x00000029
wki502_lock_increment	This field is not used. The sender SHOULD initialize it to any value between 0-0xFFFFFFFF. The receiver SHOULD ignore this field.	0x0000002A
wki502_lock_maximum	This field is not used. The sender SHOULD initialize it to any value between 0-0xFFFFFFFF. The receiver SHOULD ignore this field.	0x0000002B
wki502_pipe_increment	This field is not used. The sender	0x0000002C

Member	Valid Range	ErrorParameter Value Returned
	SHOULD initialize it to any value between 0-0xFFFFFFFF. The receiver SHOULD ignore this field.	
wki502_pipe_maximum	This field is not used. The sender SHOULD initialize it to any value between 0-0xFFFFFFFF. The receiver SHOULD ignore this field.	0x0000002D
wki502_cache_file_timeout	0 - 0xFFFFFFFF	0x0000002F
wki502_dormant_file_limit	1 - 0xFFFFFFFF	0x0000002E
wki502_read_ahead_throughput	This field is not used. The sender SHOULD initialize it to any value between 0-0xFFFFFFFF. The receiver SHOULD ignore this field.	0x0000003E
wki502_num_mailslot_buffers	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.	Not in use.
wki502_num_srv_announce_buffers	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.	Not in use.
wki502_max_illegal_datagram_events	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.	Not in use.
wki502_illegal_datagram_event_reset_frequency	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.	Not in use.
wki502_log_election_packets	This field is not used. The sender MAY initialize it to any value. The receiver SHOULD ignore this field.	Not in use.
wki502_use_opportunistic_locking	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x00000030
wki502_use_unlock_behind	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x00000031
wki502_use_close_behind	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x00000032
wki502_buf_named_pipes	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x00000033
wki502_use_lock_read_unlock	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x00000034
wki502_utilize_nt_caching	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x00000035

Member	Valid Range	ErrorParameter Value Returned
wki502_use_raw_read	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x00000036
wki502_use_raw_write	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x00000037
wki502_use_write_raw_data	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x00000038
wki502_use_encryption	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x00000039
wki502_buf_files_deny_write	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x0000003A
wki502_buf_read_only_files	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x0000003B
wki502_force_core_create_mode	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x0000003C
wki502_use_512_byte_max_transfer	This field is not used. The sender SHOULD initialize it to 0. The receiver SHOULD ignore this field.	0x0000003D

For Level value 0x000003F5

Member	Valid Range	ErrorParameter Value Returned
wki1013_keep_conn	1-65535	0x0000000D

For Level value 0x000003FA

Member	Valid Range	ErrorParameter Value Returned
wki1018_sess_timeout	60-65535	0x00000012

For Level value 0x00000416

Member	Valid Range	ErrorParameter Value Returned
wki1046_dormant_file_limit	1-0xFFFFFFFF	0x0000002E

The server SHOULD<32> enforce security measures to verify that the caller has the required permissions to execute this routine. If the server enforces security measures, and the caller does not have the required credentials, then the server MUST fail the call with ERROR_ACCESS_DENIED. Specifications for determining the identity of the caller for the purpose of performing an access check are in [MS-RPCE] section 3.3.3.1.3.

3.2.4.3 NetrWkstaUserEnum (Opnum 2)

The **NetrWkstaUserEnum** method returns information about users who are currently active on a remote computer.

```
unsigned long NetrWkstaUserEnum(
  [in, string, unique] WKSSVC_IDENTIFY_HANDLE ServerName,
  [in, out] LPWKSTA_USER_ENUM_STRUCT UserInfo,
  [in] unsigned long PreferredMaximumLength,
  [out] unsigned long* TotalEntries,
  [in, out, unique] unsigned long* ResumeHandle
);
```

ServerName: A **WKSSVC_IDENTIFY_HANDLE** (section 2.2.2.1) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

UserInfo: A pointer to the buffer to receive the data. The data MUST be returned as a **WKSTA_USER_ENUM_STRUCT** (section 2.2.5.14) structure that contains a **Level** member that specifies the type of structure to return.

PreferredMaximumLength: The number of bytes to allocate for the return data.

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

ResumeHandle: A pointer that, if specified, and if this method returns ERROR_MORE_DATA, MUST receive an implementation-specific value<33> that can be passed in subsequent calls to this method, to continue with the enumeration of currently logged-on users.

If this parameter is NULL or points to zero, then the enumeration MUST start from the beginning of the list of currently logged-on users.

Return Values: When the message processing result matches the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2). The most common error codes are listed in the following table.

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_INVALID_LEVEL 0x0000007C	The information level is invalid.
ERROR_MORE_DATA 0x000000EA	More entries are available. The *UserInfo* buffer was not large enough to contain all the entries.

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

The server SHOULD<34> enforce security measures to verify that the caller has the required permissions to execute this routine. If the server enforces security measures, and the caller does not have the required credentials, then the server MUST fail the call with ERROR_ACCESS_DENIED. Specifications for determining the identity of the caller for the purpose of performing an access check are in [MS-RPCE] section 3.3.3.1.3.

If the **Level** member of the **WKSTA_USER_ENUM_STRUCT** structure passed in the *UserInfo* parameter does not equal 0x00000000 or 0x00000001, then the server MUST fail the call.

If the **Level** member equals 0x00000000, then the server MUST return an array of the names of users currently logged on the computer. The server MUST return this information by filling the **WKSTA_USER_INFO_0_CONTAINER** (section 2.2.5.14) in the **WkstaUserInfo** field of the *UserInfo* parameter.

If the **Level** member equals 0x00000001, then the server MUST return an array of the names and domain information of each user currently logged on the computer, and a list of **OtherDomains** (section 3.2.1.3) in the computer.

If the *PreferredMaximumLength* parameter equals **MAX_PREFERRED_LENGTH** (section 2.2.1.3), the server MUST return all the requested data. Otherwise, if the *PreferredMaximumLength* is insufficient to hold all the entries, then the server MUST return the maximum number of entries that fit in the *UserInfo* buffer and return ERROR_MORE_DATA.

The following rules specify processing of the *ResumeHandle* parameter:

- If the *ResumeHandle* parameter is either NULL or points to 0x00000000, then the enumeration MUST start from the beginning of the list of the currently logged on users.<35>
- If the *ResumeHandle* parameter points to a non-zero value, then the server MUST continue enumeration based on the value of *ResumeHandle*. The server is not required to maintain any state between calls to the NetrWkstaUserEnum method.
- If the client specifies a *ResumeHandle*, and if the server returns ERROR_MORE_DATA, then the server MUST set the value to which *ResumeHandle* points to an implementation-specific value that allow the server to continue with this enumeration on a subsequent call to this method, with the same value for *ResumeHandle*.

The server is not required to maintain any state between calls to the NetrWkstaUserEnum method. If the server returns NERR_Success or ERROR_MORE_DATA, then it MUST set the *TotalEntries* parameter to equal the total number of entries that could have been enumerated from the current resume position.

3.2.4.4 NetrWkstaTransportEnum (Opnum 5)

The **NetrWkstaTransportEnum** method provides information about the transport protocols currently enabled for use by the SMB network redirector on a remote computer.

```
unsigned long NetrWkstaTransportEnum(
  [in, string, unique] WKSSVC_IDENTIFY_HANDLE ServerName,
  [in, out] LPWKSTA_TRANSPORT_ENUM_STRUCT TransportInfo,
  [in] unsigned long PreferredMaximumLength,
  [out] unsigned long* TotalEntries,
  [in, out, unique] unsigned long* ResumeHandle
);
```

ServerName: A **WKSSVC_IDENTIFY_HANDLE** (section 2.2.2.1) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

TransportInfo: A pointer to a buffer that receives a **WKSTA_TRANSPORT_ENUM_STRUCT** (section 2.2.5.16) structure. This structure contains a **Level** member that MUST be set to zero.

PreferredMaximumLength: The number of bytes to allocate for the return data.

TotalEntries: The total number of entries that could have been enumerated from the current resume position. This field can be set to any value when sent and MUST be ignored on receipt.

ResumeHandle: A pointer that, if specified, and if this method returns NERR_BufTooSmall, MUST receive an implementation-specific value<36> that can be passed in subsequent calls to this method, to continue with the enumeration of currently enabled transport protocols.

If this parameter is NULL or points to zero, then the enumeration MUST start from the beginning of the list of currently enabled transport protocols.

Return Values: When the message processing result matches the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_INVALID_LEVEL 0x0000007C	The information level is invalid.
NERR_BufTooSmall 0x0000084B	More entries are available. The *TransportInfo* buffer was not large enough to contain all the entries.

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

The server SHOULD<37> enforce security measures to verify that the caller has the required permissions to execute this routine. If the server enforces security measures, and the caller does not have the required credentials, then the server MUST fail the call with ERROR_ACCESS_DENIED. Specifications for determining the identity of the caller for the purpose of performing an access check are in [MS-RPCE] section 3.3.3.1.3.

For any other conditions, this method MUST return any other value, and the client MUST treat all other values the same.

If the **Level** member in the **WKSTA_TRANSPORT_ENUM_STRUCT** structure passed in the *TransportInfo* parameter does not equal 0x00000000, then the server MUST fail the call.

If the **Level** member is 0x00000000, then the server MUST return an array of information about the transport protocols currently enabled for use by the SMB network redirector. The server MUST return this information by filling the **WkstaTransportInfo** member (**WKSTA_TRANSPORT_INFO_0_CONTAINER** section 2.2.5.15) of the *TransportInfo* parameter for each transport in **TransportList** (as defined in section 3.2.1.4), as follows:

- **wkti0_transport_address** set to **Transport.Address**
- **wkti0_transport_name** set to **Transport.Name**
- **wkti0_number_of_vcs** set to **Transport.VC_Count**
- **wkti0_wan_ish** set to **Transport.Wannish**

If the *PreferredMaximumLength* parameter equals **MAX_PREFERRED_LENGTH** (section 2.2.1.3), the server MUST return all the requested data. If the *PreferredMaximumLength* is insufficient to hold all the entries, then the server MUST return the maximum number of entries that fit in the *TransportInfo* buffer and return NERR_BufTooSmall.

The following rules specify processing of the *ResumeHandle* parameter:

- If the *ResumeHandle* parameter is either NULL or points to 0x00000000, then the enumeration MUST start from the beginning of the list of the currently enabled transport protocols. <38>
- If the *ResumeHandle* parameter is nonzero, then the server MUST begin enumeration based on the value of *ResumeHandle*. The server is not required to maintain any state between calls invoking the **NetrWkstaTransportEnum** method.
- If the client specified a *ResumeHandle*, and if the server returns NERR_BufTooSmall, then the server MUST set *ResumeHandle* to an implementation-specific value that allow the server to continue with this enumeration on a subsequent call to this method, using the same value for *ResumeHandle*.

The server is not required to maintain any state between calls to the **NetrWkstaTransportEnum** method. If the server returns NERR_Success, then 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 server returns NERR_BufTooSmall, then it SHOULD set the *TotalEntries* value to the total number of entries that could have been enumerated from the current resume position.<39>

3.2.4.5 (Updated Section) NetrWkstaTransportAdd (Opnum 6)

The **NetrWkstaTransportAdd** method enables the SMB network redirector to use a transport protocol on a remote computer.

```
unsigned long NetrWkstaTransportAdd(
  [in, string, unique] WKSSVC_IDENTIFY_HANDLE ServerName,
  [in] unsigned long Level,
  [in] LPWKSTA_TRANSPORT_INFO_0 TransportInfo,
  [in, out, unique] unsigned long* ErrorParameter
);
```

ServerName: A **WKSSVC_IDENTIFY_HANDLE** (section 2.2.2.1) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

Level: The information level of the data. *Level* is set to zero, meaning the *TransportInfo* parameter points to a **WKSTA_TRANSPORT_INFO_0** (section 2.2.5.8) structure.

TransportInfo: A pointer to a **WKSTA_TRANSPORT_INFO_0** structure.

ErrorParameter: A pointer to a value that receives the index, starting at 0, of the first member of the ~~TransportInfo structure~~TransportInfostructure that causes the function to return ERROR_INVALID_PARAMETER. If this parameter is NULL, the index is not returned on error.

Return Values: When the message processing result meets the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_INVALID_PARAMETER 0x00000057	One of the function parameters is not valid.

Value/code	Meaning
ERROR_INVALID_LEVEL 0x0000007C	The information level is invalid.

Any other return value MUST conform to the error code requirements specified in **Protocol Detail**s (section 3).

If the *Level* parameter is not equal to zero, then the server MUST fail the call and return ERROR_INVALID_LEVEL.

If the server does not support this method then it SHOULD be processed as follows.

If any of the input parameters are invalid, the server SHOULD return ERROR_INVALID_PARAMETER. Otherwise, it SHOULD return NERR_Success.<40>

The server SHOULD<41> enforce security measures to verify that the caller has authorization to execute this routine. If the server enforces security measures and the caller does not have the required credentials, then the server SHOULD fail the call with ERROR_ACCESS_DENIED. Specifications for determining the identity of the caller for the purpose of performing an access check are in [MS-RPCE] section 3.3.3.1.3.

The *TransportInfo* parameter contains information about the transport protocol that is to be enabled. If any of the input parameters are invalid, then the server MUST return ERROR_INVALID_PARAMETER. If the caller has passed the *ErrorParameter* parameter, then the server MUST return the zero-based index of the first member of the structure the *TransportInfo* parameter points to that was invalid.

If this method call is successful, then the server MUST store values from members of the **WKSTA_TRANSPORT_INFO_0** structure passed in the *TransportInfo* parameter into the abstract data model elements for each transport in **TransportList** (section 3.2.1.4) as follows:

- **wkti0_transport_address** stored in **Transport.Address**
- **wkti0_transport_name** stored in **Transport.Name**
- **wkti0_number_of_vcs** stored in **Transport.VC_Count**
- **wkti0_wan_ish** stored in **Transport.Wannish**

3.2.4.6 NetrWkstaTransportDel (Opnum 7)

The **NetrWkstaTransportDel** method disables the use of a transport protocol by the SMB network redirector on a remote computer. The transport can be re-enabled by calling the **NetrWkstaTransportAdd** (section 3.2.4.5) method.

```
unsigned long NetrWkstaTransportDel(
  [in, string, unique] WKSSVC_IDENTIFY_HANDLE ServerName,
  [in, string, unique] wchar_t* TransportName,
  [in] unsigned long ForceLevel
);
```

ServerName: A **WKSSVC_IDENTIFY_HANDLE** (section 2.2.2.1) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

TransportName: A pointer to a string that specifies the name of the transport protocol to disconnect from the SMB network redirector.

ForceLevel: The action to take if there are handles open to files or printers using the transport protocol. This parameter MUST be one of the following values:

Value/code	Meaning
USE_NOFORCE 0x00000000	Do not disconnect or close the open handles if open handles are using the transport protocol.
USE_FORCE 0x00000001	Same as 0x00000000 (USE_NOFORCE); do not disconnect or close the open handles if open handles are using the transport protocol.
USE_LOTS_OF_FORCE 0x00000002	Forcefully close any open handles and disable the specified transport protocol from the SMB network redirector.

Return Values: When the message processing result meets the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_INVALID_PARAMETER 0x00000057	One of the function parameters is invalid.
ERROR_OPEN_FILES 0x00002401	There are open files, or printer handles are using the transport protocol pending on this connection.
ERROR_DEVICE_IN_USE 0x00002404	The device or open directory handle is using the transport protocol and cannot be disconnected.

If the *ForceLevel* parameter does not equal 0x00000000, 0x00000001, or 0x00000002, the server MUST fail the call with ERROR_INVALID_PARAMETER. If the *ForceLevel* parameter is 0x00000000 or 0x00000001 and any open directory handle is using the transport protocol provided in the *TransportName* field, the server MUST fail the call with ERROR_DEVICE_IN_USE. If the *ForceLevel* parameter is 0x00000000 or 0x00000001 and any open files or printer handles are using the transport protocol provided in the *TransportName* field, fail the call with ERROR_OPEN_FILES.

If the server does not support this method, it SHOULD<42> return NERR_Success if the *ForceLevel* parameter is valid. If the server does support this method, it MUST be processed as follows.

The server SHOULD<43> enforce security measures to verify that the caller has the required permissions to execute this routine. If the server enforces security measures and the caller does not have the required credentials, the server MUST fail the call with ERROR_ACCESS_DENIED. Specifications for determining the identity of the caller for the purpose of performing an access check are in ([MS-RPCE] section 3.3.3.1.3.

If any open file or printer handles are using the transport protocol that this call is trying to disable, the server behavior MUST depend on the value of the *ForceLevel* parameter. If the *ForceLevel* parameter is 0x00000000 or 0x00000001, the server MUST fail the call. If the *ForceLevel* parameter is 0x00000002, the server MUST forcefully close all open handles and disable the transport protocol.

If this method call is successful, the server MUST remove this protocol from its list of currently enabled transport protocols.

3.2.4.7 NetrUseAdd (Opnum 8)

The **NetrUseAdd** method establishes a connection between the workstation server and an SMB server. Workstation servers SHOULD NOT allow this method to be invoked remotely<44> and SHOULD return ERROR_CALL_NOT_IMPLEMENTED.

```
unsigned long NetrUseAdd(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in] unsigned long Level,
  [in, switch_is(Level)] LPUSE_INFO InfoStruct,
  [in, out, unique] unsigned long* ErrorParameter
);
```

ServerName: A **WKSSVC_IMPERSONATE_HANDLE** (section 2.2.2.2) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

Level: A value that specifies the information level of the data. This parameter MUST be one of the following values; otherwise, the server MUST fail the call with an ERROR_INVALID_LEVEL code.

Value	Meaning
0x00000000	The buffer is of type **USE_INFO_0** (section 2.2.5.21).
0x00000001	The buffer is of type **USE_INFO_1** (section 2.2.5.22).
0x00000002	The buffer is of type **USE_INFO_2** (section 2.2.5.23).
0x00000003	The buffer is of type **USE_INFO_3** (section 2.2.5.24).

InfoStruct: A pointer to the buffer that specifies the data. The format of this data depends on the value of the *Level* parameter.

ErrorParameter: A pointer to a value that receives an unsigned 32-bit integer. This parameter is meaningful only if the method returns ERROR_INVALID_PARAMETER.

Return Values: When the message processing result meets the description in the right-hand column of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_INVALID_PARAMETER 0x00000057	One of the function parameters is not valid.
ERROR_INVALID_LEVEL 0x0000007C	The information level is invalid.
ERROR_CALL_NOT_IMPLEMENTED 0x00000078	This function is not supported on this system.

The server SHOULD<45> enforce security measures to verify that the caller has the required permissions to execute this routine. If the server enforces security measures and the caller does not

have the required credentials, the server SHOULD fail the call with ERROR_ACCESS_DENIED. Specifications for determining the identity of the caller for the purpose of performing an access check are in [MS-RPCE] section 3.3.3.1.3.

The *Level* parameter determines the type of structure that the client has used to specify information about the new connection. 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 ERROR_INVALID_LEVEL error code.

- If the *Level* parameter is 0x00000000, the Buffer parameter points to a **USE_INFO_0** structure.
- If the *Level* parameter is 0x00000001, the Buffer parameter points to a **USE_INFO_1** structure.
- If the *Level* parameter is 0x00000002, the Buffer parameter points to a **USE_INFO_2** structure.
- If the *Level* parameter is 0x00000003, the Buffer parameter points to a **USE_INFO_3** structure.

The server MUST verify the **InfoStruct** elements as follows:

- If ui*_remote is not a UNC path format, the server MUST fail the call with ERROR_INVALID_PARAMETER. If ui*_remote is a UNC path format, it MUST canonicalize the ui*_remote path, as specified in [MS-SRVS] section 3.1.4.33.
- If ui*_local is not NULL, the server MUST canonicalize ui*_local and verify the device name format based on ui*_asg_type.
  - If ui*_asg_type is USE_WILDCARD or USE_IPC, the server MUST fail the call with ERROR_INVALID_PARAMETER.
  - If ui*_asg_type is USE_DISKDEV, ui*_local MUST be in the form "<drive name>:". Otherwise, the server MUST fail the call with ERROR_INVALID_PARAMETER.
  - If ui*_asg_type is USE_SPOOLDEV, ui*_local MUST be in the form "LPTn:" or "PRN:". Otherwise, the server MUST fail the call with ERROR_INVALID_PARAMETER.
  - If ui*_asg_type is USE_CHARDEV, ui*_local MUST be in the form "COMn:" or "AUX:". Otherwise, the server MUST fail the call with ERROR_INVALID_PARAMETER.
- If the *Level* parameter value is greater than or equal to 2 and ui*_username, ui*_password, and ui*_domainname are NULL, the server MUST attempt to establish a null session as specified in [MS-CIFS] section 3.2.4.2.4. If the *Level* parameter value is greater than or equal to 2 and ui*_username, ui*_password, and ui*_domainname are not NULL, the server MUST canonicalize the user name, password, and domain name as specified in [MS-SRVS] section 3.1.4.33.
- If the length of ui*_password is greater than 65, the server MUST fail the call with ERROR_INVALID_PARAMETER.

The server MUST ensure that the remaining steps are executed atomically with respect to other callers performing queries or updates to the **UseTable** and **Connection** tables.

If **IsWorkstationPaused** is TRUE, the server MUST verify the format of ui*_local. If ui*_local is prefixed with "PRN" or "COM", the server MUST fail the call with an ERROR_REDIR_PAUSED error code. Otherwise, the server MUST invoke the events specified in [MS-CIFS] section 3.4.4.10, passing the following as the parameters: name of the server in the ui*_remote field, name of the share in ui*_remote, and user credentials associated with ui*_username constructed from ui*_username, ui*_domainname, and ui*_password.

If the CIFS server returns STATUS_SUCCESS, the server MUST verify the remote resource type and local device type.

If the CIFS server returns the remote resource type "unknown" and ui*_remote is in the form "\\server\IPC$" or "\\server\pipe", the server MUST treat the remote resource type as a "named

pipe". If the CIFS server returns the remote resource type "unknown" and ui*_remote is NOT of the form "\\server\IPC$" or "\\server\pipe", the server MUST treat the remote resource type as a "disk share".

If the remote resource type does not match a local device type, the server MUST fail the call with ERROR_INVALID_PARAMETER. If the remote resource type matches a local device type, the server MUST initialize a new connection and add it to the **Connection** table. The connection MUST be initialized as follows:

- **Connection.local** is set to the canonicalized ui*_local name.
- **Connection.remote** is set to the canonicalized ui*_remote path name.
- **Connection.status** is set to the caller-supplied ui*_status.
- **Connection.asgtype** is set to the caller-supplied ui*_asg_type.
- **Connection.refcount** is set to the caller-supplied ui*_refcount.
- **Connection.username** is set to ui*_username.
- **Connection.usecount** is set to ui*_usecount.
- **Connection.domain** is set to ui*_domainname.
- **Connection.context** is set to the **ClientGenericContext** structure returned by the CIFS server, as specified in [MS-CIFS] section 3.4.4.10.

The server MUST invoke the event to impersonate the client as specified in [MS-RPCE] section 3.3.3.4.3.1, passing NULL as input parameter. If this event fails, the server MUST return an error. If the event returns UserToken, the server MUST look in **UseTable** for an entry where UserToken matches **UseEntry.UserToken**. If no entry is found, the server MUST create a new entry in **UseTable** and insert a new connection entry in **UseEntry.ConnectionTable**. The server MUST invoke the event to end the client impersonation as specified in [MS-RPCE] section 3.3.3.4.3.3 and return NERR_Success to the caller.

If the CIFS server returns a failure, the server MUST invoke the event to end the client impersonation as specified in [MS-RPCE] section 3.3.3.4.3.3 and fail the call with the status code received from the event.

3.2.4.8 NetrUseGetInfo (Opnum 9)

The **NetrUseGetInfo** method retrieves information from a remote workstation about a connection to a shared resource on an SMB server. The server SHOULD NOT allow this method to be invoked remotely<46> and SHOULD return ERROR_CALL_NOT_IMPLEMENTED.

```
unsigned long NetrUseGetInfo(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in, string] wchar_t* UseName,
  [in] unsigned long Level,
  [out, switch_is(Level)] LPUSE_INFO InfoStruct
);
```

ServerName: A **WKSSVC_IMPERSONATE_HANDLE** (section 2.2.2.2) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

UseName: A pointer to a string that specifies the local device name or shared resource name for which to return information.

Level: A value that specifies the information level of the data. This parameter MUST be one of the following values; otherwise, the server MUST fail the call with an ERROR_INVALID_LEVEL code.

Value	Meaning
0x00000000	The buffer is of type **USE_INFO_0** (section 2.2.5.21).
0x00000001	The buffer is of type **USE_INFO_1** (section 2.2.5.22).
0x00000002	The buffer is of type **USE_INFO_2** (section 2.2.5.23).
0x00000003	The buffer is of type **USE_INFO_3** (section 2.2.5.24).

InfoStruct: A pointer to the buffer that specifies the data. The format of this data depends on the value of the *Level* parameter.

Return Values: When the message processing result meets the description in the right-hand column of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_INVALID_PARAMETER 0x00000057	One of the function parameters is not valid.
ERROR_INVALID_LEVEL 0x0000007C	The information level is invalid.
NERR_UseNotFound 0x000008CA	The network connection could not be found.

The server SHOULD<47> enforce security measures to verify that the caller has the required permissions to execute this routine. If the server enforces security measures and the caller does not have the required credentials, the server MUST fail the call with ERROR_ACCESS_DENIED. Specifications for determining the identity of the caller for the purpose of performing an access check are in [MS-RPCE] section 3.3.3.1.3.

The *UseName* parameter specifies the local device name or shared resource name for which to return information. The server MUST canonicalize *UseName* ([MS-SRVS] section 3.1.4.33). This MUST be a nonempty, null-terminated UTF-16 string; otherwise, the server MUST fail the call with an ERROR_INVALID_PARAMETER error code.

The server MUST ensure that the remaining steps are executed atomically with respect to other callers performing queries or updates to the **UseTable** and **Connection** tables.

The server invokes the event to impersonate the client ([MS-RPCE] section 3.3.3.4.3.1) passing in NULL as input parameter. If this event fails, the server MUST return an error. If the event returns UserToken, the server MUST look in **UseTable** for an entry where **UseEntry.UserToken** matches UserToken. If no match is found, the server MUST fail the call with a NERR_UseNotFound error code.

If a match is found and *UseName* is a UNC path type, the server MUST locate the connection where *UseName* matches **Connection.remote**. If *UseName* is a local device name, the server MUST locate a **UseEntry.ConnectionTable** where *UseName* matches **Connection.local**. If no match is found, the

server MUST fail the call with a NERR_UseNotFound error code. If a matching connection is found, the server MUST return information about the connection on the remote workstation.

The server MUST fill the return structures as follows:

- If the **Level** member is 0, the server MUST return the information about the connection by filling the **USE_INFO_0_CONTAINER** (section 2.2.5.25) structure in the **Buffer** field of the *InfoStruct* parameter as follows. **USE_INFO_0_CONTAINER** contains an array of **USE_INFO_0** structures.
  - **ui0_local** set to **Connection.local**
  - **ui0_remote** set to **Connection.Remote**
- If the **Level** member is 1, the server MUST return the information about the connection by filling the **USE_INFO_1_CONTAINER** (section 2.2.5.26) structure in the **Buffer** field of the *InfoStruct* parameter as follows. **USE_INFO_1_CONTAINER** contains an array of **USE_INFO_1** structures.
  - **ui1_local** set to **Connection.local**
  - **ui1_remote** set to **Connection.remote**
  - **ui1_password** set to NULL
  - **ui1_status** set to **Connection.status**
  - **ui1_asg_type** set to **Connection.asgtype**
  - **ui1_refcount** set to **Connection.refcount**
  - **ui1_usecount** set to **Connection.useCount**
- If the **Level** member is 2, the server MUST return the information about the connection by filling the **USE_INFO_2_CONTAINER** (section 2.2.5.27) structure in the **Buffer** field of the *InfoStruct* parameter as follows. **USE_INFO_2_CONTAINER** contains an array of **USE_INFO_2** structures.
  - **ui2_local** set to **Connection.local**
  - **ui2_remote** set to **Connection.remote**
  - **ui2_password** set to NULL
  - **ui2_status** set to **Connection.status**
  - **ui2_asg_type** set to **Connection.asgtype**
  - **ui2_refcount** set to **Connection.refcount**
  - **ui2_usecount** set to **Connection.useCount**
  - **ui2_domainname** set to **Connection.domain**
- If the **Level** member is 3, the server MUST return the information about the connection by filling the **USE_INFO_3_CONTAINER** structure in the **Buffer** field of the *InfoStruct* parameter as follows. **USE_INFO_3_CONTAINER** contains an array of **USE_INFO_3** structures.
  - **ui2_local** set to **Connection.local**
  - **ui2_remote** set to **Connection.remote**
  - **ui2_password** set to NULL
  - **ui2_status** set to **Connection.status**

- **ui2_asg_type** set to **Connection.asgtype**
- **ui2_refcount** set to **Connection.refcount**
- **ui2_usecount** set to **Connection.useCount**
- **ui2_domainname** set to **Connection.domain**
- **ui2_flag** set to 0

The server MUST invoke the event to end the client impersonation ([MS-RPCE] section 3.3.3.4.3.3).

3.2.4.9 (Updated Section) NetrUseDel (Opnum 10)

The ~~NetUseDel~~NetrUseDel function terminates a connection from the workstation server to a shared resource on an SMB server. The server SHOULD NOT allow this method to be invoked remotely<48> and SHOULD return ERROR_CALL_NOT_IMPLEMENTED.

```
unsigned long NetrUseDel(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in, string] wchar_t* UseName,
  [in] unsigned long ForceLevel
);
```

ServerName: A **WKSSVC_IMPERSONATE_HANDLE** (section 2.2.2.2) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

UseName: A pointer to a string that specifies the local device name or shared resource name for which to return information.

ForceLevel: The level of force to use in deleting the connection. This parameter MUST be one of the following values; otherwise, the server MUST fail the call with an ERROR_INVALID_LEVEL error code.

Value/code	Meaning
USE_NOFORCE 0x00000000	Do not disconnect the connection if open files exist on the connection.
USE_FORCE 0x00000001	Same as 0x00000000 (USE_NOFORCE); do not disconnect the connection if open files exist on the connection.
USE_LOTS_OF_FORCE 0x00000002	Close any open files and disconnect the connection.

Return Values: When the message processing result meets the description in the right-hand column of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.

Value/code	Meaning
ERROR_INVALID_PARAMETER 0x00000057	One of the function parameters is not valid.
ERROR_INVALID_LEVEL 0x0000007C	The force level is invalid.
ERROR_DEVICE_IN_USE 0x00002404	The connection handle is in use and cannot be disconnected.
ERROR_REDIR_PAUSED 0x00000048	Remote access to the specified printer or serial communications device has been paused.

The server SHOULD<49> enforce security measures to verify that the caller has the required permissions to execute this routine. If the server enforces security measures and the caller does not have the required credentials, the server MUST fail the call with ERROR_ACCESS_DENIED. Specifications for determining the identity of the caller for the purpose of performing an access check are in [MS-RPCE] section 3.3.3.1.3.

The *UseName* parameter specifies the local device name or shared resource name for which to delete a tree connection. The server MUST canonicalize *UseName* ([MS-SRVS] section 3.1.4.33). This MUST be a nonempty, null-terminated UTF-16 string; otherwise, the server MUST fail the call with an ERROR_INVALID_PARAMETER error code.

The server MUST ensure that the remaining steps are executed atomically with respect to other callers performing queries or updates to the **UseTable** and **Connection** tables.

The server MUST invoke the event to impersonate the client as specified in [MS-RPCE] section 3.3.3.4.3.1, passing in NULL as input parameter. If this event fails, the server MUST return an error. If the event returns UserToken, the server MUST look in **UseTable** for the user where **UseEntry.UserToken** matches UserToken. If no match is found, the server MUST fail the call with a NERR_UseNotFound error code.

If a match is found and *UseName* is a UNC path type, the server MUST locate a **UseEntry.ConnectionTable** where *UseName* matches **Connection.remote**. If *UseName* is a local device name, the server MUST locate a **UseEntry.ConnectionTable** table where *UseName* matches **Connection.local**. If no match is found, the server MUST fail the call with a NERR_UseNotFound error code.

If a matching connection is found and **IsWorkstationPaused** is TRUE, the server MUST verify the format of **Connection.local**. If **Connection.local** is prefixed with "PRN" or "COM", the server MUST fail the call with an ERROR_REDIR_PAUSED error code. If a matching connection is found and **IsWorkstationPaused** is FALSE, the server MUST disconnect the connection with the server by invoking the event specified in [MS-CIFS] section 3.4.4.11, providing the **Connection context** handle and *ForceLevel* as input parameters.

If the CIFS server returns a failure, the server MUST fail the call with the status code ERROR_DEVICE_IN_USE.

If the CIFS server returns STATUS_SUCCESS, the server MUST delete the connection in **UseEntry.ConnectionTable** where *UseName* matches **Connection.remote** and return NERR_Success to the caller.

If **UseEntry.ConnectionTable** is empty, the server MUST remove the **UseEntry** for the user, where **UseEntry.UserToken** matches UserToken.

The server MUST invoke the event to end the client impersonation as specified in [MS-RPCE] section 3.3.3.4.3.3.

3.2.4.10 NetrUseEnum (Opnum 11)

The **NetrUseEnum** method lists open connections between a workstation server and a remote SMB server. The server SHOULD NOT allow this method to be invoked remotely<50> and SHOULD return ERROR_CALL_NOT_IMPLEMENTED.

```
unsigned long NetrUseEnum(
  [in, string, unique] WKSSVC_IDENTIFY_HANDLE ServerName,
  [in, out] LPUSE_ENUM_STRUCT InfoStruct,
  [in] unsigned long PreferredMaximumLength,
  [out] unsigned long* TotalEntries,
  [in, out, unique] unsigned long* ResumeHandle
);
```

ServerName: A **WKSSVC_IDENTIFY_HANDLE** (section 2.2.2.1) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

InfoStruct: The **USE_ENUM_STRUCT** (section 2.2.5.28) structure contains a *Level* parameter that indicates the type of structure to return.

Value	Meaning
0x00000000	Specifies a local device name and the share name of a remote resource.
0x00000001	Specifies information about the connection between a local device and a shared resource, including connection status and type.
0x00000002	Specifies information about the connection between a local device and a shared resource, including the connection status, connection type, user name, and domain name.

PreferredMaximumLength: The number of bytes to allocate for the return data.

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

ResumeHandle: A pointer that, if specified and if this method returns ERROR_MORE_DATA, MUST receive an implementation-specific value that can be passed in subsequent calls to this method in order to continue with the enumeration of currently logged-on users.

If this parameter is NULL or points to zero, the enumeration MUST start from the beginning of the list of currently logged-on users.

Return Values: The method returns 0x00000000 (NERR_Success) to indicate success; otherwise, it returns a nonzero error code. The method can take any specific error code value, as specified in [MS-ERREF] section 2.2. The most common error codes are listed in the following table.

Value/code	Meaning
NERR_Success 0x00000000	The client request succeeded.
ERROR_INVALID_LEVEL 0x0000007C	The system call level is not correct.
ERROR_MORE_DATA 0x000000EA	The client request succeeded. More entries are available. Not all entries could be returned in the buffer size that is specified by

Value/code	Meaning
	PreferredMaximumLength.
ERROR_NOT_ENOUGH_MEMORY 0x00000008	Not enough storage is available to process this command.
NERR_BufTooSmall 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.

The server SHOULD<51> enforce security measures to verify that the caller has the required permissions to execute this routine. If the server enforces security measures and the caller does not have the required credentials, the server MUST fail the call with ERROR_ACCESS_DENIED. Specifications for determining the identity of the caller for the purpose of performing an access check are in [MS-RPCE] section 3.3.3.1.3.

The *InfoStruct* parameter has a **Level** member. The value of **Level** MUST be 0, 1, or 2. If the Level member is not equal to one of the valid values, the server MUST fail the call with an ERROR_INVALID_LEVEL error code.

The server MUST invoke the event to impersonate the client as specified in [MS-RPCE] section 3.3.3.4.3.1, passing in NULL as input parameter. If this event fails, the server MUST return an error. If the event returns UserToken, the server MUST look in the **UseTable** for the user where **UseEntry.UserToken** matches UserToken. If no match is found, the server MUST set the value of *TotalEntries* to 0 and return a NERR_Success.

If a matching UserToken is found for the user in **UseTable**, the server MUST enumerate connections in **UseEntry.ConnectionTable** and fill the return structures as follows:

- If the **Level** member is 0, the server MUST return the information about the connection by filling the **USE_INFO_0_CONTAINER** (section 2.2.5.25) structure in the **Buffer** field of the *InfoStruct* parameter as follows. **USE_INFO_0_CONTAINER** contains an array of **USE_INFO_0** (section 2.2.5.21) structures.
  - **ui0_local** set to **Connection.local**
  - **ui0_remote** set to **Connection.remote**
- If the **Level** member is 1, the server MUST return the information about the connection by filling the **USE_INFO_1_CONTAINER** (section 2.2.5.26) structure in the **Buffer** field of the *InfoStruct* parameter as follows. The **USE_INFO_1_CONTAINER** structure contains an array of **USE_INFO_1** (section 2.2.5.22) structures.
  - **ui1_local** set to **Connection.local**
  - **ui1_remote** set to **Connection.remote**
  - **ui1_password** set to NULL
  - **ui1_status** set to **Connection.status**
  - **ui1_asg_type** set to **Connection.asgtype**
  - **ui1_refcount** set to **Connection.refcount**
  - **ui1_usecount** set to **Connection.useCount**
- If the **Level** member is 2, the server MUST return the information about the connection by filling the **USE_INFO_2_CONTAINER** (section 2.2.5.27) structure in the **Buffer** field of the *InfoStruct*

parameter as follows. **USE_INFO_2_CONTAINER** contains an array of **USE_INFO_2** (section 2.2.5.23) structures.

- **ui2_local** set to **Connection.local**
- **ui2_remote** set to **Connection.remote**
- **ui2_password** set to NULL
- **ui2_status** set to **Connection.status**
- **ui2_asg_type** set to **Connection.asgtype**
- **ui2_refcount** set to **Connection.refcount**
- **ui2_usecount** set to **Connection.useCount**
- **ui2_domainname** set to **Connection.domain**

If the *PreferredMaximumLength* parameter equals MAX_PREFERRED_LENGTH, the server MUST return all the requested data. If *PreferredMaximumLength* is insufficient to hold all the entries, the server MUST return the maximum number of entries that fit in the *InfoStruct* buffer and return NERR_BufTooSmall.

The following rules specify processing of the *ResumeHandle* parameter:

- If the *ResumeHandle* parameter either is NULL or points to 0x00000000, the enumeration MUST start from the beginning of the list of the currently established connections.
- If the *ResumeHandle* parameter is nonzero, the server MUST begin enumeration based on the value of *ResumeHandle*. The server is not required to maintain any state between calls invoking the **NetrUseEnum** method.
- If the client specified *ResumeHandle* and if the server returns NERR_BufTooSmall, the server MUST set *ResumeHandle* to an implementation-specific value<52> that allow the server to continue with this enumeration on a subsequent call to this method, using the same value for *ResumeHandle*.

The server is not required to maintain any state between calls to the **NetrUseEnum** method. If the server returns NERR_Success, 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 server returns NERR_BufTooSmall, it SHOULD set the *TotalEntries* value to the total number of entries that could have been enumerated from the current resume position.

The server MUST invoke the event to end the client impersonation as specified in [MS-RPCE] section 3.3.3.4.3.3.

3.2.4.11 NetrWorkstationStatisticsGet (Opnum 13)

The **NetrWorkstationStatisticsGet** method returns various statistics about the SMB network redirector on a remote computer.

```
unsigned long NetrWorkstationStatisticsGet(
  [in, string, unique] WKSSVC_IDENTIFY_HANDLE ServerName,
  [in, string, unique] wchar_t* ServiceName,
  [in] unsigned long Level,
  [in] unsigned long Options,
  [out] LPSTAT_WORKSTATION_0* Buffer
);
```

ServerName: A **WKSSVC_IDENTIFY_HANDLE** (section 2.2.2.1) that identifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

ServiceName: A pointer to a string specifying the name of the workstation service. This value MUST be ignored on receipt.

Level: The information level of the data. This value MUST be zero.

Options: This value MUST be zero.

Buffer: A pointer to a **STAT_WORKSTATION_0** (section 2.2.5.11) structure that contains the statistical information.

Return Values: When the message processing result matches the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_INVALID_LEVEL 0x0000007C	The information level is invalid.
ERROR_INVALID_PARAMETER 0x00000057	One of the function parameters is invalid.

If the *Level* parameter does not equal 0x00000000, then the server MUST fail the call and return ERROR_INVALID_LEVEL.

If the *Options* parameter does not equal 0x00000000, then the server MUST fail the call and return ERROR_INVALID_PARAMETER. The server SHOULD<53> enforce security measures to verify that the caller has the required permissions to execute this routine. If the server enforces security measures and the caller does not have the required credentials, then the server MUST fail the call and return ERROR_ACCESS_DENIED.

The server MUST fill in all the members of the **STAT_WORKSTATION_0** structure that the *Buffer* parameter points to with the corresponding statistics about the SMB network redirector.

Some fields of the **STAT_WORKSTATION_0** structure are implementation-specific, as specified in section 2.2.5.11. These fields indicate certain performance characteristics of an operating system and do not apply to all servers. If a field does not apply to the server, then it MUST set that field to zero.<54>

3.2.4.12 NetrGetJoinInformation (Opnum 20)

The **NetrGetJoinInformation** method retrieves information about the workgroup or domain to which the specified computer is joined.

```
unsigned long NetrGetJoinInformation(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in, out, string] wchar_t** NameBuffer,
  [out] PNETSETUP_JOIN_STATUS BufferType
);
```

ServerName: A **WKSSVC_IMPERSONATE_HANDLE** (section 2.2.2.2) that specifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

NameBuffer: A pointer to the address of the buffer that receives the name of the domain or workgroup to which the computer is joined, and that also holds the computer name as input. The server MUST ignore this parameter on input.

BufferType: A pointer to a value from the **NETSETUP_JOIN_STATUS** (section 2.2.3.1) enumeration that specifies the status of a workstation.

Return Values: When the message processing result meets the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
RPC_S_PROTSEQ_NOT_SUPPORTED 0x000006A7	The RPC protocol sequence is not supported.

Any other return value MUST conform to the error code requirements specified in Protocol Details (section 3).

The following statements define the sequence of message-processing operations.

1. The server MUST retrieve the RPC protocol sequence used for the current call, as specified in [MS-RPCE] section 3.1.3.4.1, specifying the server binding handle maintained by the RPC runtime ([C706] section 6.2.1). If that RPC protocol sequence is not **NCACN_NP**, the server SHOULD return **RPC_S_PROTSEQ_NOT_SUPPORTED**.<55>

2. The server MUST check that the caller has been granted access rights using the algorithm specified in the **Access Control Abstract Data Model** (section 3.2.1.1), with *Access Request mask* initialized to **WKSTA_NETAPI_QUERY**; if not, the server MUST return ERROR_ACCESS_DENIED.

3. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the server MUST return an error.

4. The server MUST compute the response in the following way.

    - If **DomainNameFQDN** is set to NULL, then *BufferType* is set to NetSetupUnjoined, and *NameBuffer* is set to NULL.

    - Else if **DomainSid** is set to NULL, then *BufferType* is set to NetSetupWorkgroupName and *NameBuffer* is set to **DomainNameNetBIOS**.

    - Else *BufferType* is set to NetSetupDomainName and *NameBuffer* is set to **DomainNameFQDN**.

5. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).

If no errors occur, the server MUST return NERR_Success.

3.2.4.13 NetrJoinDomain2 (Opnum 22)

The **NetrJoinDomain2** method uses encrypted credentials to join a computer to a domain or a workgroup.<56>

For high-level, informative discussions about domain controller location and domain join and unjoin, see [MS-ADOD] section 2.7.7 and [MS-ADOD] section 3.1. Also, see the example in section 4.3 for more information.

```
unsigned long NetrJoinDomain2(
  [in] handle_t RpcBindingHandle,
  [in, string, unique] wchar_t* ServerName,
  [in, string] wchar_t* DomainNameParam,
  [in, string, unique] wchar_t* MachineAccountOU,
  [in, string, unique] wchar_t* AccountName,
  [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
  [in] unsigned long Options
);
```

RpcBindingHandle: An RPC binding handle [C706].

ServerName: This parameter has no effect on message processing in any environment. The client MUST set this parameter to a value that resolves to the IP protocol layer destination address of the RPC packets it transmits ([MS-RPCE] section 2.1.1.2). The server MUST ignore this parameter.

DomainNameParam: A pointer to a string that specifies the domain name or workgroup name to join, and optionally the domain controller machine name within the domain. This parameter MUST NOT be NULL.

If the string specifies the name of the preferred domain controller to perform the join operation, then the string MUST be of the form *DomainNameToJoin**MachineName*, where *DomainNameToJoin* is the domain to join, "\" is a delimiter, and *MachineName* is the name of the domain controller to perform the join operation. In all cases, the *DomainNameToJoin* portion of this parameter MUST be either the NetBIOS name of the domain or the fully qualified domain name (FQDN) of the domain. If the *MachineName* is passed, it MUST be either the NetBIOS name of the domain controller or the Internet host name of the domain controller. The format of *DomainNameToJoin* places no constraint on the format of *MachineName* and vice versa; thus, each of the following permutations are accepted:

- NetBIOS name\NetBIOS name
- NetBIOS name\Internet host name
- FQDN\NetBIOS name

MachineAccountOU: A pointer to a string that MUST contain [RFC1777] the format name of the organizational unit (OU) directory object under which the machine account directory object is created. This parameter is optional. If specified, this string MUST contain the full path; for example, OU=testOU,DC=domain,DC=Domain,DC=com.

AccountName: A pointer to a string that specifies an account name in the domain *DomainNameParam* to use when connecting to a domain controller. This parameter is optional. If this parameter is NULL, the caller's account name MUST be used. If this parameter is specified, the format MUST be one of the following:

- <NetBIOSDomainName>\<UserName>
- <FullyQualifiedDNSDomainName>\<UserName>
- <UserName>@<FullyQualifiedDNSDomainName>

Password: A pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** (section 2.2.5.18) structure that specifies the encrypted password to use with the *AccountName* parameter. Sections 3.2.4.13.1 and 3.2.4.13.3 specify the processing of this parameter.

Options: A 32-bit bitfield that specifies modifications to default server behavior in message processing.<57>

Value/code	Meaning
NETSETUP_JOIN_DOMAIN 0x00000001	Joins the computer to a domain. The default action is to join the computer to a workgroup.
NETSETUP_ACCT_CREATE 0x00000002	Creates the account on the domain. The name is the persisted abstract state **ComputerNameNetBIOS** unless this behavior is altered by another option such as NETSETUP_JOIN_WITH_NEW_NAME.
NETSETUP_ACCT_DELETE 0x00000004	Disables the old account when the join operation occurs on a computer that is already joined to a domain. **Important** This flag is neither supported nor tested for use with **NetrJoinDomain2**; its use is therefore not specified in any message processing.
NETSETUP_DOMAIN_JOIN_IF_JOINED 0x00000020	Allows a join to a new domain even if the computer is already joined to a domain.
NETSETUP_JOIN_UNSECURE 0x00000040	Performs an unsecured join. MUST be used only in conjunction with the NETSETUP_MACHINE_PWD_PASSED flag.
NETSETUP_MACHINE_PWD_PASSED 0x00000080	Indicates that the *Password* parameter SHOULD<58> specify the password for the machine joining the domain. This flag is valid only for unsecured joins, which MUST be indicated by setting the NETSETUP_JOIN_UNSECURE flag. If this flag is set, the value of *Password* determines the value stored for the computer password during the join process.
NETSETUP_DEFER_SPN_SET 0x00000100	Indicates that the service principal name (SPN) and the **DnsHostName** properties on the computer SHOULD NOT<59> be updated at this time, but instead SHOULD<60> be updated during a subsequent call to **NetrRenameMachineInDomain2** (section 3.2.4.15).
NETSETUP_JOIN_DC_ACCOUNT 0x00000200	Indicates that the join SHOULD<61> be allowed if an existing account exists and it is a domain controller account.<62>
NETSETUP_JOIN_WITH_NEW_NAME 0x00000400	Indicates that the join SHOULD<63>occur using the new computer name.
NETSETUP_INSTALL_INVOCATION 0x00040000	Indicates that the protocol method was invoked during installation.

Return Values: When the message processing result meets the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_FILE_NOT_FOUND	The object was not found.

Value/code	Meaning
0x00000002	
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_NOT_SUPPORTED 0x00000032	The request is not supported.
ERROR_INVALID_PASSWORD 0x00000056	The specified network password is not correct.
ERROR_INVALID_PARAMETER 0x00000057	The parameter is incorrect.
ERROR_PASSWORD_RESTRICTION 0x0000052D	Unable to update the password. The value provided for the new password does not meet the length, complexity, or history requirements of the domain.
ERROR_LOGON_FAILURE 0x0000052E	Logon failure: unknown user name or bad password.
ERROR_NONE_MAPPED 0x00000534	The account was not found.
ERROR_INVALID_DOMAIN_ROLE 0x0000054A	The name of a domain controller was provided in the *DomainNameParam* parameter, and validation of that domain controller failed. Validation is specified in the message-processing steps for the section "Domain Join" later.
ERROR_NO_SUCH_DOMAIN 0x0000054B	The specified domain either does not exist or could not be contacted.
RPC_S_PROTSEQ_NOT_SUPPORTED 0x000006A7	The RPC protocol sequence is not supported.
RPC_S_CALL_IN_PROGRESS 0x000006FF	A remote procedure call is already in progress.<64>
NERR_UserExists 0x000008B0	The user account already exists.
NERR_SetupAlreadyJoined 0x00000A83	This computer is already joined to a domain.
NERR_SetupDomainController 0x00000A85	This computer is a domain controller and cannot be unjoined from a domain.
NERR_InvalidWorkgroupName 0x00000A87	The specified workgroup name is invalid.

Any other return value MUST conform to the error code requirements specified in **Protocol Deta**ils (section 3).

Message processing for the **NetrJoinDomain2** method specifies the behavior of joining either a domain or a workgroup. The behavior of this method is covered in the following subsections:

- Section 3.2.4.13.1 specifies the message processing that is common to both domain and workgroup joins.

- Section 3.2.4.13.2 specifies the state transition associated with a domain join.
- Section 3.2.4.13.3 specifies the message processing that is involved in a domain join.
- Section 3.2.4.13.4 specifies the message processing that is involved in a workgroup join.

Several password data elements are involved in message processing for the **NetrJoinDomain2** method, and they are distinguished as follows:

Password: A parameter to this method, either the password corresponding to the *AccountName* that is used to authenticate at the domain controller or the password used for the computer account. The bits in the *Options* parameter determine how *Password* is used. This element is distinct from the client data model element **Password** that is defined in section 3.2.1.6.

PasswordString: The Unicode UTF-8 string that corresponds to the plaintext form of the password in *Password*. This variable is relevant to sections 3.2.4.13.1 and 3.2.4.13.3.

ComputerPasswordString: The ASCII string that contains the plaintext form of the password for the computer account. This variable is relevant to section 3.2.4.13.3.

3.2.4.13.1 Common Message Processing

The following statements pertain to all message processing:

- The server MUST ignore any flags set in the *Options* parameter that it does not support.<65><66><67>
- Unless otherwise noted, if the server encounters an error during message processing, the following actions are specified:
  - The server SHOULD revert any state changes made.
  - The server MUST stop message processing.
  - The server MUST return the error to the caller.

The following ordered statements specify the sequence of message processing operations:

1. The server MUST retrieve the RPC protocol sequence used for the current call, as specified in [MS-RPCE] section 3.1.3.4.1, specifying the server binding handle maintained by the RPC runtime ([C706] section 6.2.1). If that RPC protocol sequence is not **NCACN_NP**, the server SHOULD return **RPC_S_PROTSEQ_NOT_SUPPORTED**.<68>
2. The server MUST check that the caller has been granted access rights using the algorithm specified in the **Access Control Abstract Data Model** (section 3.2.1.1), with *Access Request mask* initialized to **WKSTA_NETAPI_CHANGE_CONFIG**; if not, the server MUST return ERROR_ACCESS_DENIED.
3. If *Password* is NULL, then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *Password*, as specified in section 2.2.5.18. *PasswordString* MUST be equal to the decrypted and decoded value. The decrypted buffer is represented as **JOINPR_USER_PASSWORD**, as specified in section 2.2.5.17. The value of the Length member MUST be less than 513; otherwise, message processing is stopped, and the server MUST return ERROR_INVALID_PASSWORD.
4. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the server MUST return an error.
5. If the value of the *DomainNameParam* parameter is NULL, the server MUST stop message processing and return ERROR_INVALID_PARAMETER. Otherwise, message processing continues.

6. The server SHOULD return ERROR_NOT_SUPPORTED if the server does not support processing of this message.<69>

7. If the server that is processing the message is a domain controller, the server MUST stop message processing and return NERR_SetupDomainController. Otherwise, message processing continues.

8. If *Options* does not have the **NETSETUP_JOIN_DOMAIN** bit set, then the server MUST continue processing this message, as specified in section 3.2.4.13.4; otherwise, the server MUST process the message as specified in section 3.2.4.13.3.

3.2.4.13.2 State Changes Required for Domain Join

A computer is said to be joined to a domain if a certain state exists on the computer and in the domain NC. See the specific state requirements that MUST occur both locally and in the domain NC at a domain controller (DC) (sections 3.2.1.2 and 3.2.1.5, and [MS-ADTS] section 6.4).

The state changes referenced above appear in the sequence of message processing steps later in this specification but are listed here to aid the reader. To understand the domain join process, the following normative description identifies the state manipulation performed during message processing to affect the state changes required for a computer to join a domain.

The server MUST persist in the machine:

- The domain-name ([MS-ADTS] section 6.4.1) and the domain prefix for the account domain of the DC, as queried by the server from the DC. The server MUST store these values in the local **DomainName** and **DomainSid** elements (section 3.2.1.6), respectively.
- The domain-secret ([MS-ADTS] section 6.4).

The server MUST persist in the domain:

- A computer account object with the following LDAP attributes.<70> See [RFC2252] and [RFC2253] for more information about LDAP.

LDAP attribute name	Value
userAccountControl ([MS-ADA3] section 2.342)	The USER_WORKSTATION_TRUST_ACCOUNT bit is set, and the USER_ACCOUNT_DISABLED bit is not set ([MS-SAMR] section 2.2.1.12). See the userAccountControl mapping table ([MS-SAMR] section 3.1.5.14.2) for information on the mapping of these bits in the LDAP protocol.
sAMAccountName ([MS-ADA3] section 2.222)	The value of machine-account-name ([MS-ADTS] section 6.4.1). This is **ComputerNameNetBIOS** (see section 3.2.1.5), suffixed with a "$" character.
unicodePwd ([MS-ADA3] section 2.332)	The value of domain-secret ([MS-ADTS] section 6.4.1). Protocols that expose this attribute persist the NT hash of domain-secret ([MS-SAMR] section 3.1.5.10).
dNSHostName ([MS-ADA1] section 2.185)	The value of **ComputerNameFQDN** (see section 3.2.1.5).
servicePrincipalName ([MS-ADA3] section 2.253)	Two values, as specified in the message processing sequence later in this specification: ▪ A DNS-based SPN. ▪ A NetBIOS-based SPN for the computer joining the domain.

3.2.4.13.3 Domain Join Specific Message Processing

The following definitions are used in the specification of message processing that follows.

- *DomainNameString*: A Unicode UTF-8 string with the same properties specified for the parameter *DomainNameParam*.
- *DomainControllerString*: A UTF-8 string that contains the name of a domain controller in the domain that the server is joining.
- *DomainObject*: An object in the domain database ([MS-ADTS] section 6.4).
- *MachineAccountOUString*: A UTF-8 string that contains the organizational unit (OU) in the directory for the machine account.
- *ComputerAccountString*: A UTF-8 string that contains the value stored in the sAMAccountName attribute of the computer object in the domain database.
- *DNSComputerNameString*: A UTF-8 string that contains the Internet host name of the computer.
- *Spn1*: A UTF-8 string that contains a DNS-based service principal name (SPN) for the computer joining the domain.
- *Spn2*: A UTF-8 string that contains a NetBIOS-based SPN for the computer joining the domain.

The following statements define the sequence of message-processing operations:

1. If the **NETSETUP_MACHINE_PWD_PASSED** bit is set in *Options*, and the **NETSETUP_JOIN_UNSECURE** bit is not set in *Options*, the server MUST return ERROR_INVALID_PARAMETER. Otherwise, message processing continues.
2. If the **NETSETUP_MACHINE_PWD_PASSED** bit is set in *Options*, and *AccountName* is not NULL, the server MUST return ERROR_INVALID_PARAMETER. Otherwise, message processing continues.
3. If the **NETSETUP_MACHINE_PWD_PASSED** bit is set in *Options*, and either *Password* is NULL or the length of the *PasswordString* is zero, the server MUST return ERROR_PASSWORD_RESTRICTION. Otherwise, message processing continues.
4. If the **NETSETUP_MACHINE_PWD_PASSED** bit is set in *Options*, the value of *PasswordString* MUST be copied to the value of *ComputerPasswordString*, and *PasswordString* MUST be set to NULL.
5. If the server processing the message is already joined to a domain, and the **NETSETUP_DOMAIN_JOIN_IF_JOINED** bit is not set in *Options*, the server MUST return NERR_SetupAlreadyJoined. Otherwise, message processing continues.
6. If *DomainNameString* contains the character "\", *DomainNameString* MUST be truncated such that the value of *DomainNameString* is equal to the substring of *DomainNameString* that ends prior to the first "\" character, and *DomainControllerString* MUST be equal to the substring beginning after the first "\" character. This is the name of the target domain controller as specified by the caller.

   The specified domain controller MUST be validated by invoking the **DsrGetDcNameEx2** method ([MS-NRPC] section 3.5.4.3.1) on the DomainControllerString computer, specifying the following parameters:

   - *ComputerName* = *DomainControllerString*
   - *AccountName* = NULL
   - *AllowableAccountControlBits* = 0

- *DomainName* = *DomainNameString*
- *SiteName* = 0
- *Flags* = B | J | R

If the call succeeds and DomainControllerInfo->DomainControllerName matches DomainControllerString, execution continues at step 8.

If the call fails, or the returned domain controller name does not match DomainControllerString, the server MUST invoke the **DsrGetDcNameEx2** method ([MS-NRPC] section 3.5.4.3.1) on the DomainControllerString computer, specifying the following parameters:

- *ComputerName* = *DomainControllerString*
- *AccountName* = NULL
- *AllowableAccountControlBits* = 0
- *DomainName* = *DomainNameString*
- *SiteName* = 0
- *Flags* = B | J | S

If the call fails, the server MUST stop message processing and return ERROR_NO_SUCH_DOMAIN. If the call succeeds and DomainControllerInfo->DomainControllerName matches DomainControllerString, execution continues at step 8. Otherwise, the server MUST stop message processing and return ERROR_INVALID_DOMAIN_ROLE.

7. If *DomainControllerString* was not initialized in the preceding step, the server MUST locate a domain controller for the domain specified in *DomainNameString*, and *DomainControllerString* MUST be set to the string name of the located domain controller. The same parameter values that are shown above are used except that the *ComputerName* parameter is set to NULL.

8. The **SiteName** ADM element SHOULD be updated with the client site name information that was returned as part of the call to **DsrGetDcNameEx2**.

9. *DomainNameString* MUST be a validated domain name. The validation process is specified in section 3.2.4.16, where *NameType* is **NetSetupDomain** from the **NETSETUP_NAME_TYPE** (section 2.2.3.2) enumeration. If this validation fails, the server MUST stop message processing and return the error specified in the validation process.

10. If **ComputerNameNetBIOS** is identical to *DomainNameString*, the server MUST return ERROR_INVALID_DOMAINNAME. Otherwise, message processing continues.

11. If the NETSETUP_MACHINE_PWD_PASSED bit is set in *Options*, the server MUST attempt to establish an authenticated SMB session with the domain controller named by the value of *DomainControllerString*. The client identity and authorization information that were used when establishing the SMB session are retrieved from RPC ([MS-RPCE] section 2.2.1.1.10 and [MS-RPCE] section 3.3.3.4.3).

12. If the NETSETUP_MACHINE_PWD_PASSED bit is set in *Options*, and the session fails to be established in the previous step with a non-authentication failure, the server MUST stop message processing and return the error. If the session fails to be established for some other reason, the server MUST attempt to establish an anonymous session. If an error occurs, the server MUST stop message processing and return that error. Otherwise, message processing continues.

13. If the NETSETUP_MACHINE_PWD_PASSED bit is not set in *Options*, the server MUST establish an authenticated SMB session with the domain controller named by the value of *DomainControllerString*. The credentials that are supplied during authentication are those specified

in *PasswordString*, and the security context that is established MUST be that of *AccountName*. If an error occurs, the server MUST stop message processing and return that error. Otherwise, message processing continues.

14. The SMB session that was established in the previous steps and the security context associated with it MUST be used for any higher-layer RPC calls made to the domain controller over the SMB NCACN_NP protocol sequence ([MS-RPCE] section 2.1.1.2 and [MS-SMB] section 3.2.4.2.4).

15. The server MUST query the domain controller for its domain name and SID ([MS-LSAD] section 3.1.4.4.3).<71>

16. The server MUST store the values queried in the previous step in the local **DomainName** and **DomainSid** elements, as defined in section 3.2.1.6.

17. If the NETSETUP_MACHINE_PWD_PASSED bit is not set in *Options*, and either the NETSETUP_WIN9X_UPGRADE bit or the NETSETUP_JOIN_UNSECURE bit is set in *Options*, *ComputerPasswordString* is the first 14 characters of **ComputerName.NetBIOS** in lowercase.

18. If the NETSETUP_MACHINE_PWD_PASSED bit is not set in *Options*, and neither the NETSETUP_WIN9X_UPGRADE bit nor the NETSETUP_JOIN_UNSECURE bit is set in *Options*, *ComputerPasswordString* MUST be an ASCII string of randomly chosen characters. Each character's ASCII code MUST be between 32 and 122 inclusive. When randomly generating a password string, the server MUST generate 120 characters. Each character SHOULD be generated using the algorithm specified in [FIPS186-2] Appendix 3.1 and [RFC4086].<72>

19. The server MUST store the value of *ComputerPasswordString* locally for consumption by security-provider services when authenticating the computer. The stored password MUST be maintained by the Netlogon Protocol [MS-NRPC] in the **Password** ADM element, as defined in section 3.2.1.6.

20. If the value of the *MachineAccountOU* parameter is not NULL, the value of *MachineAccountOUString* MUST equal *MachineAccountOU*. If the value of *MachineAccountOU* is NULL, *MachineAccountOUString* MUST equal the value specified by the well-known object identified by the GUID with value GUID_COMPUTERS_CONTAINER_W ([MS-ADTS] section 6.1.1.4).

21. If the [RFC1777]-format name of the organizational unit (OU) where the object exists, as specified by the value of *MachineAccountOUString*, cannot be found in the domain database, the server MUST return ERROR_FILE_NOT_FOUND. Otherwise, message processing continues.

22. *ComputerAccountString* MUST be set to the UTF-8 string consisting of **ComputerNameNetBIOS** suffixed with a "$" character.

23. *DNSComputerNameString* MUST equal the UTF-8 string **ComputerNameFQDN**.

24. *Spn1* MUST be a UTF-8 string equal to the concatenation of "HOST/" and the value of *DNSComputerNameString*.

25. *Spn2* MUST be a UTF-8 string equal to the concatenation of "HOST/" and the value of *ComputerAccountString*.

26. If the NETSETUP_ACCT_CREATE bit is set in *Options*, the server MUST create the domain object in the domain *DomainNameParam* at *DomainControllerString*. Manipulation of the domain computer object state is exposed through LDAP protocols ([RFC2252], [RFC2253], and [MS-SAMR]). If the domain object already exists in an organizational unit (OU) ([MS-ADSC] section 2.217) that is different from the one specified in *MachineAccountOU*, the server MUST stop message processing and return NERR_UserExists. If the domain object already exists but the *MachineAccountOU* is NULL or refers to the organizational unit (OU) of the domain object, the server MUST return NERR_Success. Otherwise, message processing continues.

27. If the NETSETUP_ACCT_CREATE bit is not set in *Options* and the domain object does not already exist in the domain *DomainNameParam* at the domain controller, the server MUST stop message processing and return ERROR_NONE_MAPPED. Otherwise, message processing continues.

28. If the NETSETUP_ACCT_CREATE bit is not set in *Options*, and either the NETSETUP_WIN9X_UPGRADE bit or the NETSETUP_JOIN_UNSECURE bit is set in *Options*, the server MUST send a request to the Netlogon Remote Protocol on the local computer to perform Netlogon authentication with the domain controllers. This is to validate that the value of *ComputerPasswordString* persisted locally equals the value of the password on the domain object in the LDAP attribute unicodePwd. If the authentication fails, the server MUST stop message processing and return ERROR_LOGON_FAILURE. Otherwise, message processing continues. For more information about Netlogon authentication between domain-joined computers and domain controllers, see [MS-NRPC].

29. The following LDAP attributes on *DomainObject* MUST be set to the values shown in the table. The security context provided to the LDAP protocol is *AccountName* and the credential is *PasswordString*. For details about attributes and attribute names, see [MS-ADTS]. For details about LDAP, see [RFC2252] and [RFC2253].

LDAP attribute name	Value
userAccountControl ([MS-ADA3] section 2.342)	The USER_WORKSTATION_TRUST_ACCOUNT bit is set and the USER_ACCOUNT_DISABLED bit is not set. See the userAccountControl mapping table ([MS-SAMR] section 3.1.5.14.2) for details about the mapping of these bits in the LDAP protocol.
sAMAccountName ([MS-ADA3] section 2.222)	The value of ComputerAccountString.
unicodePwd ([MS-ADA3] section 2.332)	The value of ComputerPasswordString. Protocols that expose this attribute persist the NT hash of the ComputerPasswordString ([MS-SAMR] section 3.1.5.10).

30. The following LDAP attributes on *DomainObject* MUST be set to the values shown in the table unless the NETSETUP_DEFER_SPN_SET bit is set in *Options*.

LDAP attribute name	Value
dNSHostName ([MS-ADA1] section 2.185)	The value of *DNSComputerNameString*.
servicePrincipalName ([MS-ADA3] section 2.253)	Two values: *Spn1* *Spn2*

31. The server MUST configure the local Netlogon Remote Protocol [MS-NRPC] so that it is aware of being joined to a domain with the name *DomainNameParam*.

32. The server MUST configure the local Windows Time Service (W32Time) [WTSREF] so that it is aware of being joined to a domain.

33. The server SHOULD store the value *DNSComputerNameString* locally so that the DNS service registers name records for the local computer [NIS].<73>

34. The server SHOULD add the Domain Admins group to the local administrators group and the Domain Users group to the local users groups ([MS-SAMR] section 3.1.4.2).

35. The server MUST apply all state changes specified in section 3.2.4.13.2.

36. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).

If no errors occur, the server MUST return NERR_Success.

3.2.4.13.4 Workgroup Join Specific Message Processing

The following statements define the sequence of message processing operations.

1. If the server processing the message is already joined to a domain, the server MUST return NERR_SetupAlreadyJoined. Otherwise, message processing continues.

2. The *DomainNameParam* parameter MUST be validated as a valid workgroup name. The validation process is specified in section 3.2.4.16, where *NameType* is **NetSetupWorkgroup** from the **NETSETUP_NAME_TYPE** (section 2.2.3.2) enumeration. If this validation fails, the server MUST return the error specified in the preceding validation process.

3. The server's ADM elements in section 3.2.1.6 MUST be set as follows:

   - **DomainName.NetBIOS** = *DomainNameParam*
   - **DomainName.FQDN** = **NULL**
   - **DomainGuid** = **NULL**
   - **DomainSid** = **NULL**

4. The server MUST configure the local Netlogon Remote Protocol [MS-NRPC] so that it is aware of being joined to a workgroup by the name of *DomainNameParam*.

5. The server MUST configure the local W32Time [WTSREF] so that it is aware of being joined to a workgroup.

6. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).

If no errors occur, the server MUST return NERR_Success.

3.2.4.14 NetrUnjoinDomain2 (Opnum 23)

The **NetrUnjoinDomain2** method uses encrypted credentials to unjoin a computer from a workgroup or domain.<74>

```
unsigned long NetrUnjoinDomain2(
  [in] handle_t RpcBindingHandle,
  [in, string, unique] wchar_t* ServerName,
  [in, string, unique] wchar_t* AccountName,
  [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
  [in] unsigned long Options
);
```

RpcBindingHandle: An RPC binding handle [C706].

ServerName: This parameter has no effect on message processing in any environment. The client MUST set this parameter to a value that resolves to the IP destination address of the RPC packets it transmits ([MS-RPCE] section 2.1.1.2). The server MUST ignore this parameter.

AccountName: A pointer to a string that specifies the account name in the joined domain to use when connecting to a domain controller. This parameter is optional. If this parameter is NULL, the caller's account name MUST be used.

Password: An optional pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** (section 2.2.5.18) structure that specifies the encrypted password to use with the *AccountName* parameter. If this parameter is NULL, the caller's security context MUST be used.

Options: A 32-bit bitfield specifying modifications to default message processing behavior.

Value/code	Meaning
NETSETUP_ACCT_DELETE 0x00000004	Disables the account when the unjoin operation occurs.
NETSETUP_IGNORE_UNSUPPORTED_FLAGS 0x10000000	The server ignores undefined flags when this bit is set.<75> This option is present to allow for the addition of new optional values in the future.

Return Values: When the message processing result meets the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_INVALID_PASSWORD 0x00000056	The specified network password is not correct.
ERROR_INVALID_PARAMETER 0x00000057	One of the function parameters is not valid.
ERROR_INVALID_FLAGS 0x000003EC	Invalid option flags are specified.
RPC_S_PROTSEQ_NOT_SUPPORTED 0x000006A7	The RPC protocol sequence is not supported.
NERR_SetupNotJoined 0x00000A84	This computer is not currently joined to a domain.
NERR_SetupDomainController 0x00000A85	This computer is a domain controller and cannot be unjoined from a domain.

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

Unless otherwise noted, if the server encounters an error during message processing, the server SHOULD revert any state changes made, MUST stop message processing, and MUST return the error to the caller.<76>

The following definitions are used in the specification of message processing that follows.

- *DomainControllerString*: A UTF-8 string containing the name of a domain controller in the domain to which the server is joined.
- *DomainObject*: An object in the domain database ([MS-ADTS] section 6.4) having the value of **ComputerNameNetBIOS** suffixed with a "$" character for the **SamAccountName** attribute.

- *PasswordString*: A UTF-8 string that contains a password in cleartext.

The following statements define the sequence of message processing operations.

1. The server MUST retrieve the RPC protocol sequence used for the current call, as specified in [MS-RPCE] section 3.1.3.4.1, specifying the server binding handle maintained by the RPC runtime ([C706] section 6.2.1). If that RPC protocol sequence is not **NCACN_NP**, the server SHOULD return **RPC_S_PROTSEQ_NOT_SUPPORTED**.<77>

2. The server MUST check that the caller has been granted access rights using the algorithm specified in the **Access Control Abstract Data Model** (section 3.2.1.1), with *Access Request mask* initialized to **WKSTA_NETAPI_CHANGE_CONFIG**; if not, the server MUST return ERROR_ACCESS_DENIED.

3. If *Password* is NULL, then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *Password*, as defined in section 2.2.5.18. *PasswordString* MUST equal the decrypted and decoded value. The decrypted buffer is represented as a **JOINPR_USER_PASSWORD** (section 2.2.5.17). The value of the **Length** member MUST be less than 513; otherwise, message processing is stopped, and the server MUST return ERROR_INVALID_PASSWORD.

4. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the server MUST return an error.

5. The server MUST stop message processing and return NERR_SetupNotJoined if **DomainSid**, as defined in section 3.2.1.6, is NULL.

6. If any bits other than NETSETUP_ACCT_DELETE are set in *Options*, the server MUST check the NETSETUP_IGNORE_UNSUPPORTED_FLAGS bit. If it is not set, the server MUST stop message processing and return ERROR_INVALID_FLAGS. Otherwise, message processing continues.

7. The server MUST stop message processing and return NERR_SetupDomainController if the server processing the message is a domain controller. Otherwise, message processing continues.

8. The server MUST locate a domain controller in the joined domain. *DomainControllerString* MUST be equal to the string name of the located domain controller.

9. The server MUST establish an authenticated SMB session with the domain controller named by the value of *DomainControllerString*. The credentials that are supplied during authentication are those specified in *PasswordString*, and the security context that is established MUST be that of *AccountName*. If an error occurs, the server MUST stop message processing and return that error. Otherwise, message processing continues.

10. The SMB session established in the previous step and the security context associated with it MUST be used for any higher layer RPC calls made to the domain controller over the SMB **NCACN_NP** protocol sequence ([MS-SMB] section 3.2.4.2.4).

11. The server MUST configure the local Net Logon Remote Protocol [MS-NRPC] so that it is aware of no longer being joined to a domain.

12. The server MUST configure the local W32Time [WTSREF] so that it is aware of no longer being joined to a domain.

13. The server MUST set **DomainSid** to NULL.

14. The server MUST delete the persisted password that was stored previously in the **Password** ADM element when processing a **NetrJoinDomain2** message.

15. If the NETSETUP_ACCT_DELETE bit is set in *Options*, then the server MUST update the *DomainObject* **userAccountControl** attribute by setting the USER_ACCOUNT_DISABLED bit. See the userAccountControl Mapping Table ([MS-SAMR] section 3.1.5.14.2) for details on the mapping of these bits in LDAP.

The security context provided to LDAP is *AccountName*, and the credential is *PasswordString*. For details on attributes and attribute names, see [MS-ADTS]. For details on LDAP, see [RFC2252] and [RFC2253].

16. The server MUST configure the Certificate Auto Enrollment Service ([MSFT-AUTOENROLLMENT] and [MS-CERSOD] section 2.1.2.2.2) so that it is aware of no longer being joined to a domain.

17. The server MUST configure the local Net Logon Remote Protocol [MS-NRPC] such that it is aware of no longer being joined to a domain.

18. The server SHOULD store the Internet host name locally such that the DNS service unregisters name records for the local computer [NIS].<78>

19. The server SHOULD remove the domain admins group from the local administrators group, and the server SHOULD remove the domain users group from the local users groups [MS-SAMR].

20. The server MUST set the following ADM elements, as defined in section 3.2.1.6, to NULL: **ClientName**, **DomainName**, **DomainSid**, **ForestNameFQDN**, **DomainGuid**, **SiteName**, and **Password**.

21. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).

If no errors occur, the server MUST return NERR_Success.

3.2.4.15 NetrRenameMachineInDomain2 (Opnum 24)

The **NetrRenameMachineInDomain2** method uses encrypted credentials to change the locally persisted **ComputerNameNetBIOS**, and to optionally rename the computer account for a server currently in a domain, without first removing the computer from the domain and then adding it back.<79>

```
unsigned long NetrRenameMachineInDomain2(
  [in] handle_t RpcBindingHandle,
  [in, string, unique] wchar_t* ServerName,
  [in, string, unique] wchar_t* MachineName,
  [in, string, unique] wchar_t* AccountName,
  [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
  [in] unsigned long Options
);
```

RpcBindingHandle: An RPC binding handle [C706].

ServerName: This parameter has no effect on message processing in any environment. The client MUST set this parameter to a value that resolves to the IP protocol layer destination address of the RPC packets it transmits ([MS-RPCE] section 2.1.1.2). The server MUST ignore this parameter.

MachineName: A pointer to a string that specifies the new computer name. This parameter is optional. If this parameter is NULL, the current machine name is used.

AccountName: A pointer to a string that specifies an account name in the joined domain to use when connecting to a domain controller. This parameter is optional. If this parameter is NULL, the caller's account name is used.

Password: An optional pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** (section 2.2.5.18) structure that specifies the encrypted password to use with the *AccountName* parameter. If this parameter is NULL, the caller's security context MUST be used.

Options: A 32-bit bitfield that specifies modifications to default server behavior in message processing.

Value/code	Meaning
NETSETUP_ACCT_CREATE 0x00000002	Renames the computer account in the domain. If this flag is not set, the computer name is changed locally but no changes are made to the computer account in the domain.
NETSETUP_DNS_NAME_CHANGES_ONLY 0x00001000	Limits any updates to DNS-based names only.

Return Values: When the message processing result meets the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_NOT_SUPPORTED 0x00000032	The request is not supported.
ERROR_INVALID_PASSWORD 0x00000056	The specified network password is not correct.
ERROR_INVALID_PARAMETER 0x00000057	The parameter is incorrect.
RPC_S_PROTSEQ_NOT_SUPPORTED 0x000006A7	The RPC protocol sequence is not supported.
NERR_SetupNotJoined 0x00000A84	This computer is not currently joined to a domain.
NERR_SetupDomainController 0x00000A85	This computer is a domain controller and cannot be renamed.<80>

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

Unless otherwise noted, if the server encounters an error during message processing, the server SHOULD revert any state changes made, MUST stop message processing, and MUST return the error to the caller.<81>

The following definitions are used in the specification of message processing that follows:

- *DomainControllerString*: A UTF-8 string that contains the name of a domain controller in the domain to which the server is joined.
- *NewComputerAccountString*: A UTF-8 string that contains the value to be stored in the samAccountName attribute of the server's computer account in the domain database.
- *ComputerNameString*: A UTF-8 string that contains the new NetBIOS name of the server.
- *DNSComputerNameString*: A UTF-8 string that contains the new Internet host name of the server.
- *Spn1*: A UTF-8 string.

- *Spn2*: A UTF-8 string.
- *PasswordString*: A UTF-8 string that contains a password in cleartext.
- *DomainControllerConnection*: An ADCONNECTION_HANDLE ([MS-DTYP] section 2.2.2) to a domain controller.
- *LdapResultMessages*: A list of LDAPMessage ([RFC2251]) containing results from an operation performed on DomainControllerConnection.
- *ComputerAccountDN*: A UTF-8 string that contains the distinguished name (DN) of the computer account.

The following statements define the sequence of message processing operations.

1. The server MUST retrieve the RPC protocol sequence used for the current call, as specified in [MS-RPCE] section 3.1.3.4.1, specifying the server binding handle maintained by the RPC runtime ([C706] section 6.2.1). If that RPC protocol sequence is not **NCACN_NP**, the server SHOULD return **RPC_S_PROTSEQ_NOT_SUPPORTED**.<82>
2. The server MUST check that the caller has been granted access rights using the algorithm specified in the **Access Control Abstract Data Model** (section 3.2.1.1), with *Access Request mask* initialized to **WKSTA_NETAPI_CHANGE_CONFIG**; if not, the server MUST return ERROR_ACCESS_DENIED.
3. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the server MUST return an error.
4. If *Password* is NULL then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *Password* as defined in section 2.2.5.18). *PasswordString* MUST be equal to the decrypted and decoded value. The decrypted buffer is represented as a **JOINPR_USER_PASSWORD** (section 2.2.5.17). The value of the **Length** member MUST be less than 513; otherwise, message processing is stopped, and the server MUST return ERROR_INVALID_PASSWORD.
5. If the server is not a domain controller, or is not a member of a domain, then the server MUST fail the call with NERR_SetupNotJoined. Otherwise, message processing continues.
6. If the server is an RODC ([MS-DRSR] section 5.7), the server MUST fail the call with ERROR_NOT_SUPPORTED. Otherwise, message processing continues.
7. If the *Options* parameter does not contain **NETSETUP_ACCT_CREATE**, the server MUST apply the new name locally, updating **ComputerNameNetBIOS** so that other protocols on the server can operate using the new name. If this operation fails, the server MUST return an implementation-specific error and stop message processing. If the operation is successful, then the server MUST stop message processing and return successfully.
8. If the *Options* parameter contains **NETSETUP_ACCT_CREATE**, the server MUST continue message processing.
9. The server MUST convert<83> the name in the *MachineName* parameter to a string NetBIOS name. This conversion MUST match with the conversion used in Netlogon Remote Protocol. *ComputerNameString* MUST equal the resulting value. *NewComputerAccountString* MUST equal the resulting value concatenated with the "$" character.
10. The server MUST use the security context associated with the credentials provided in the *AccountName* and *Password* parameters to perform the rest of the remote operations.
11. The server MUST locate a writable domain controller for the domain to which the server is joined, by invoking the **DsrGetDcNameEx2** method on the local [MS-NRPC] server specifying the following parameters:

- *ComputerName* = NULL
- *AccountName* = ComputerNameNetBIOS
- *AllowableAccountControlBits* = ADS_UF_WORKSTATION_TRUST_ACCOUNT | ADS_UF_SERVER_TRUST_ACCOUNT ([MS-ADTS] section 2.2.16)
- *DomainName* = DomainNameFQDN
- *DomainGuid* = NULL
- *SiteName* = NULL
- *Flags* = (J | B) ([MS-NRPC] section 3.5.4.3.1)

If the DsrGetDcNameEx2 method fails, the server MUST retry the call specifying the following parameters:

- *ComputerName* = NULL
- *AccountName* = NULL
- *AllowableAccountControlBits* = 0
- *DomainName* = DomainNameFQDN
- *DomainGuid* = NULL
- *SiteName* = NULL
- *Flags* = (J | B) ([MS-NRPC] section 3.5.4.3.1)

If both calls fail, the method MUST fail.

Otherwise, DomainControllerString MUST equal the string name of the returned writable domain controller.

12. If the NETSETUP_DNS_NAME_CHANGES_ONLY bit is not set in *Options*, then the server MUST make the following attribute update:
    - samAccountName updated to equal NewComputerAccountString.
13. *DNSComputerNameString* is the concatenation of *ComputerNameString* and the DNS suffix on the server.<84>
14. *Spn1* is the concatenation of "HOST/" with *DNSComputerNameString*.
15. *Spn2* is the concatenation of "HOST/" with *ComputerNameString*.
16. The server invokes **LDAP Bind** (section 3.2.4.22.2) with the following parameters:
    - *DomainControllerBindTarget*: DomainControllerString
    - *AccountNameForBind*: AccountName
    - *PasswordForBind*: PasswordString
    - *Encrypt*: FALSE
    - *DisallowReferrals*: FALSE

    The result is stored in DomainControllerConnection.

17. The server invokes **Query Computer Account DN for the Local Machine** (section 3.2.4.22.1), specifying DomainControllerString for the *DomainControllerQueryTarget* parameter, storing the result in ComputerAccountDN.

18. The server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: DomainControllerConnection
    - *TaskInputRequestMessage*: LDAP modifyRequest message [RFC2251] section 4.6 as follows:
        - Object: ComputerAccountDN
        - The modification sequence has two list entries, set as follows:
            - First list entry
                - operation: replace
                - modification:
                    - type: DnsHostName
                    - vals: DNSComputerNameString
            - Second list entry
                - operation: replace
                - modification:
                    - type: ServicePrincipalName
                    - vals: Spn1 and Spn2
    - *TaskOutputResultMessages*: LDAPResultMessages

19. The server invokes **LDAP Unbind** (section 3.2.4.22.3), with ADConnectionToUnbind set to DomainControllerConnection.

20. If any of these updates fail, the server MUST fail the request and return the error from the writable domain controller.

21. The server MUST invoke "Update Display Name using SAMR" (section 3.2.4.22.5), specifying the following parameters:

    - *DomainController*: *DomainControllerString*
    - *MachineName*: *NewComputerNameString*

    The result of this operation MUST be ignored.

22. The server MUST apply the new name locally, updating **ComputerNameNetBIOS** so that other protocols on the server can operate using the new name.

23. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).

If no errors occur, the server MUST return NERR_Success.

3.2.4.16 NetrValidateName2 (Opnum 25)

The **NetrValidateName**2 method verifies the validity of a computer, workgroup, or domain name.<85>

```
unsigned long NetrValidateName2(
  [in] handle_t RpcBindingHandle,
  [in, string, unique] wchar_t* ServerName,
  [in, string] wchar_t* NameToValidate,
  [in, string, unique] wchar_t* AccountName,
  [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
  [in] NETSETUP_NAME_TYPE NameType
);
```

RpcBindingHandle: An RPC binding handle [C706].

ServerName: This parameter has no effect on message processing in any environment. The client MUST set this parameter to a value that resolves to the IP protocol layer destination address of the RPC packets it transmits ([MS-RPCE] section 2.1.1.2). The server MUST ignore this parameter.

NameToValidate: A pointer to a string that specifies the name to validate, according to its type.

AccountName: The server SHOULD ignore this parameter.

Password: The server SHOULD ignore this parameter.

NameType: Specifies the type of validation to perform (section 2.2.3.2).

Return Values: When the message processing result matches the description in column 2 of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_DUP_NAME 0x00000034	The connection was denied because a duplicate name exists on the network.
ERROR_INVALID_PASSWORD 0x00000056	The specified network password is incorrect.
ERROR_INVALID_PARAMETER 0x00000057	The parameter is incorrect.
ERROR_INVALID_NAME 0x0000007B	The file name, directory name, or volume label syntax is incorrect.
ERROR_INVALID_DOMAINNAME 0x000004BC	The format of the specified domain name is invalid.
ERROR_NO_SUCH_DOMAIN 0x0000054B	The specified domain either does not exist or could not be contacted.
RPC_S_PROTSEQ_NOT_SUPPORTED 0x000006A7	The RPC protocol sequence is not supported.

Value/code	Meaning
NERR_InvalidComputer 0x0000092F	This computer name is invalid.
NERR_InvalidWorkgroupName 0x00000A87	The specified workgroup name is invalid.
DNS_ERROR_NON_RFC_NAME 0x00002554	The Internet host name does not comply with RFC specifications.
DNS_ERROR_INVALID_NAME_CHAR 0x00002558	The Internet host name contains an invalid character.
RPC_E_REMOTE_DISABLED 0x8001011C	Remote calls are not allowed for this process.

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

The following definition is used in the specification of message processing that follows.

- *PasswordString*: A Unicode UTF-8 string containing a password in cleartext.

The following statements define the sequence of message processing operations.

1. The server MUST retrieve the RPC protocol sequence used for the current call, as specified in [MS-RPCE] section 3.1.3.4.1, specifying the server binding handle maintained by the RPC runtime ([C706] section 6.2.1). If that RPC protocol sequence is not **NCACN_NP**, the server SHOULD return **RPC_S_PROTSEQ_NOT_SUPPORTED**.<86>

2. The server SHOULD<87> stop message processing and return an implementation-specific error if the caller is not local. Specifications for determining if the caller is local are in [MS-RPCE].

3. The server MUST check that the caller has been granted access rights using the algorithm specified in the **Access Control Abstract Data Model** (section 3.2.1.1), with *Access Request mask* initialized to **WKSTA_NETAPI_QUERY**; if not, the server MUST return ERROR_ACCESS_DENIED.

4. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the server MUST return an error.

5. If *Password* is NULL then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *Password* (section 2.2.5.18). *PasswordString* MUST equal the decrypted and decoded value. The decrypted buffer is represented as a **JOINPR_USER_PASSWORD** (section 2.2.5.17). The value of the **Length** member MUST be less than 513; otherwise, message processing is stopped, and the server MUST return ERROR_INVALID_PASSWORD.

6. If *NameType* is *NetSetupUnknown* the server MUST stop message processing and return ERROR_INVALID_PARAMETER.

7. First, the method MUST perform syntactic validation of the name as follows. For all types of validation except the **NetSetupDnsMachine** type, the syntactic validation is performed on the name expressed in the OEM character set.

   NetSetupWorkgroup

   - The length of the name MUST NOT be less than 1 or greater than 15 characters, inclusive.

   - The name MUST NOT contain characters that have any one of the following octal values:

```
001, 002, 003, 004, 005, 006, 007, 010, 011, 012, 013, 014,
015, 016, 017, 020, 021, 022, 023, 024, 025, 026, 027, 030,
031, 032, 033, 034, 035, 036, 037
```

- The name MUST NOT contain any of the following characters:

```
" / \ [ ] : | < > + = ; , ?
```

- The name MUST NOT consist entirely of the dot and space characters.

NERR_InvalidWorkgroupName MUST be returned if the check fails unless the conditions of this type are being checked as part of another type, which specifies alternate error return behavior.

NetSetupMachine

All conditions for the **NetSetupWorkgroup** type apply for this type. Additionally:

- The name MUST NOT contain an asterisk (*).
- The first character and the last character of the name MUST NOT be the space character.

NERR_InvalidComputer MUST be returned if the check fails.

NetSetupDomain

- The name MUST NOT consist entirely of the dot and space characters.

ERROR_INVALID_NAME MUST be returned if this condition is violated.

All conditions for the **NetSetupWorkgroup** type apply for this type. If the checks for **NetSetupWorkgroup** fail, then all conditions for the **NetSetupDnsMachine** apply for this type.

NetSetupNonExistentDomain

All conditions for the **NetSetupDomain** type apply for this type. Additionally:

- The name MUST contain only characters [RFC1035].

DNS_ERROR_NON_RFC_NAME MUST be returned if this restriction is violated.

NetSetupDnsMachine

The validation [RFC1035] is performed in the following order. Specifically, the name MUST NOT:

- Contain characters that have any one of the following octal values:

```
001, 002, 003, 004, 005, 006, 007, 010, 011, 012, 013, 014,
015, 016, 017, 020, 021, 022, 023, 024, 025, 026, 027, 030,
031, 032, 033, 034, 035, 036, 037
```

- Be longer than 255 octets.
- Contain a label longer than 63 octets.
- Contain two or more consecutive dots.
- Begin with a dot.

ERROR_INVALID_NAME MUST be returned if any condition in this group is violated.

- Contain a space.
- Contain any of the following characters:

```
{ | } ~ [ \ ] ^ ' : ; < = > ? @ ! " # $ % ^ ` ( ) + / , *
```

DNS_ERROR_INVALID_NAME_CHAR MUST be returned if any condition in this group is violated.

8. Second, after validating the name syntactically, the method MUST perform the following verification for the respective types of validation:

NetSetupWorkgroup

- The name MUST NOT be the name of the server receiving this call. NERR_InvalidWorkgroupName MUST be returned if this condition is violated.
- The name MUST be valid for registration as a NetBIOS group name [RFC1001].<88> If the name is not valid then ERROR_INVALID_PARAMETER MUST be returned.

NetSetupMachine

- The name MUST be valid for registration as a NetBIOS unique name [RFC1001]. Otherwise, the server MUST return NERR_InvalidComputer.
- The name MUST NOT be in use by a computer accessible on the network except for the server receiving this call. ERROR_DUP_NAME MUST be returned if this condition is violated.<89>

NetSetupDomain

- The name MUST differ from the name of the built-in domain, "BUILTIN" (**Builtin Domain Principal View**, [MS-LSAT] section 3.1.1.1.3); the comparison MUST be case-insensitive. NERR_InvalidComputer MUST be returned if this condition is violated.
- The name MUST be a name of an existing domain. ERROR_NO_SUCH_DOMAIN MUST be returned if this condition is not satisfied.<90>

NetSetupNonExistentDomain

- The name MUST differ from the name of the built-in domain, "BUILTIN" (**Builtin Domain Principal View**); the comparison MUST be case-insensitive. NERR_InvalidComputer MUST be returned if this condition is violated.
- The name MUST NOT be a name of an existing domain accessible on the network. ERROR_DUP_NAME MUST be returned if this condition is not satisfied.<91>

9. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).

If no errors occur, the server MUST return NERR_Success.

3.2.4.17 NetrGetJoinableOUs2 (Opnum 26)

The **NetrGetJoinableOUs2** method returns a list of organizational units (OUs) in which the user can create an object.<92>

```
unsigned long NetrGetJoinableOUs2(
  [in] handle_t RpcBindingHandle,
```

```
    [in, string, unique] wchar_t* ServerName,
    [in, string] wchar_t* DomainNameParam,
    [in, string, unique] wchar_t* AccountName,
    [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
    [in, out] unsigned long* OUCount,
    [out, string, size_is(, *OUCount)]
      wchar_t*** OUs
  );
```

RpcBindingHandle: An RPC binding handle [C706].

ServerName: This parameter has no effect on message processing in any environment. The client MUST set this parameter to a value that resolves to the IP protocol layer destination address of the RPC packets it transmits ([MS-RPCE] section 2.1.1.2). The server MUST ignore this parameter.

DomainNameParam: A pointer to a string that specifies the root domain under which the method searches for OUs. This parameter is also the domain of the account that the *AccountName* parameter is in.

AccountName: A pointer to a string that specifies the account name to use when connecting to a domain controller. This parameter is optional. If this parameter is NULL, the caller's account name MUST be used.

Password: An optional pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** (section 2.2.5.18) structure that specifies the encrypted password to use with the *AccountName* parameter. If the *AccountName* parameter is NULL, the caller's security context MUST be used, and this parameter MUST be ignored.

OUCount: A pointer to the count of OUs that the method returned. The server MUST ignore this parameter on input.

OUs: A pointer to a pointer of size *OUCount* to a block of strings that are the joinable OUs that the method returned.

Return Values: When the message processing result matches the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_NOT_ENOUGH_MEMORY 0x00000008	Not enough storage is available to process this command.
ERROR_INVALID_PARAMETER 0x00000057	A parameter is incorrect.<93>
RPC_S_PROTSEQ_NOT_SUPPORTED 0x000006A7	The RPC protocol sequence is not supported.
NERR_InvalidAPI 0x0000085E	The requested API is not supported on domain controllers.
NERR_DefaultJoinRequired 0x00000A86	The destination domain controller does not support creating machine accounts in OUs.

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

The following definitions are used in the specification of message processing that follows.

- *PasswordString*: A Unicode UTF-8 string containing a password in cleartext.
- *DomainControllerString*: A UTF-8 string that contains the name of a domain controller in the domain that the server is joining.
- *DomainControllerConnection*: An ADCONNECTION_HANDLE ([MS-DTYP] section 2.2.2) to a domain controller.
- *LdapResultMessages*: A list of LDAPMessage ([RFC2251]) containing results from an operation performed on DomainControllerConnection.
- *ComputerAccountDN*: A UTF-8 string that contains the DN of the computer account.

The following statements define the sequence of message processing operations.

1. The server MUST retrieve the RPC protocol sequence used for the current call, as specified in [MS-RPCE] section 3.1.3.4.1, specifying the server binding handle maintained by the RPC runtime ([C706] section 6.2.1). If that RPC protocol sequence is not **NCACN_NP**, the server SHOULD return **RPC_S_PROTSEQ_NOT_SUPPORTED**.<94>
2. The server SHOULD<95> ensure that the caller is local. Specifications for determining that the caller is local are in [MS-RPCE].
3. The server MUST check that the caller has been granted access rights using the algorithm specified in the **Access Control Abstract Data Model** (section 3.2.1.1), with *Access Request mask* initialized to **WKSTA_NETAPI_QUERY**; if not, the server MUST return ERROR_ACCESS_DENIED.
4. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the server MUST return an error.
5. If *Password* is NULL then *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *Password* (section 2.2.5.18). *PasswordString* MUST equal the decrypted and decoded value. The decrypted buffer is represented as a **JOINPR_USER_PASSWORD** (section 2.2.5.17). The value of the **Length** member MUST be less than 513; otherwise, message processing is stopped, and the server MUST return ERROR_INVALID_ PARAMETER.
6. The server SHOULD<96> enforce that this call fails on a domain controller. Otherwise, message processing continues.
7. The server MUST locate a domain controller in the domain, by invoking the **DsrGetDcNameEx2** method on the local [MS-NRPC] server specifying the following parameters:
   - *ComputerName* = NULL
   - *AccountName* = NULL
   - *AllowableAccountControlBits* = 0
   - *DomainName* = DomainNameFQDN
   - *DomainGuid* = NULL
   - *SiteName* = NULL
   - *Flags* = (J | B) ([MS-NRPC] section 3.5.4.3.1).

If a domain controller cannot be located, the method MUST fail. Otherwise, DomainControllerString MUST equal the string name of the returned writable domain controller.

8. The server invokes **LDAP Bind** (section 3.2.4.22.2) with the following parameters:

   - *DomainControllerBindTarget*: DomainControllerString
   - *AccountNameForBind*: AccountName
   - *PasswordForBind*: PasswordString
   - *Encrypt*: FALSE

   If this fails, the server MUST return NERR_DefaultJoinRequired; otherwise, the result is stored in DomainControllerConnection.

9. The server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

   - *TaskInputADConnection*: DomainControllerConnection
   - *TaskInputRequestMessage*: LDAP SearchRequest message [RFC2251] section 4.5.1 as follows:
     - baseObject: The root of the default naming context (NC)
     - scope: wholeSubtree
     - filter: ObjectClass=OrganizationalUnit
     - attributes: AllowedChildClassesEffective
     - derefAliases: neverDerefAliases
     - typesOnly: FALSE
   - *TaskOutputResultMessages*: LDAPResultMessages

10. The server MUST process the results returned from the DC in LDAPResultMessages. For each entry (SearchResultEntry, [RFC2251] section 4.5.2) returned by the search in LDAPResultMessages, if the AllowedChildClassesEffective attribute contains the value "computer", the server MUST add the DN of that entry to the results to be returned in OUs as a NULL-terminated string, and increment the value in *OUCount*.

11. The server invokes **LDAP Unbind** (section 3.2.4.22.3) with *ADConnectionToUnbind* set to DomainControllerConnection.

12. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).

If no errors occur, the server MUST return NERR_Success.

3.2.4.18 NetrAddAlternateComputerName (Opnum 27)

The **NetrAddAlternateComputerName** method adds an alternate name for a specified server.<97>

```
unsigned long NetrAddAlternateComputerName(
  [in] handle_t RpcBindingHandle,
  [in, string, unique] wchar_t* ServerName,
  [in, string, unique] wchar_t* AlternateName,
  [in, string, unique] wchar_t* DomainAccount,
  [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD EncryptedPassword,
  [in] unsigned long Reserved
```

```
);
```

RpcBindingHandle: An RPC binding handle [C706].

ServerName: This parameter has no effect on message processing in any environment. The client MUST set this parameter to a value that resolves to the IP protocol layer destination address of the RPC packets it transmits ([MS-RPCE] section 2.1.1.2). The server MUST ignore this parameter.

AlternateName: A pointer to a string that specifies the new alternate name to add. The name MUST be a valid DNS host name [RFC1035].

DomainAccount: A pointer to a string that specifies the account name in the domain to use when connecting to a domain controller. This parameter is optional. If this parameter is NULL, the caller's account name MUST be used. If this parameter is specified, the format MUST be one of the following:

- <NetBIOSDomainName>\<UserName>
- <FullyQualifiedDNSDomainName>\<UserName>
- <UserName>@<FullyQualifiedDNSDomainName>

EncryptedPassword: An optional pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** (section 2.2.5.18) structure that specifies the encrypted password to use with the *DomainAccount* parameter. If the *DomainAccount* parameter is NULL, the caller's security context MUST be used, and this parameter MUST be ignored.

Reserved: A 32-bit bitfield that SHOULD be set to zero.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	IU

Where the bits are defined as:

Value	Meaning
IU NET_IGNORE_UNSUPPORTED_FLAGS	If 1, the server MUST ignore the values of the other bits in this field. If 0, the values of the other bits in this field MUST be 0; otherwise, the server MUST return ERROR_INVALID_FLAGS.<98>

Return Values: When the message processing result matches the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_NOT_SUPPORTED 0x00000032	This method is not supported by this server.

Value/code	Meaning
ERROR_INVALID_PASSWORD 0x00000056	The specified network password is incorrect.
ERROR_INVALID_PARAMETER 0x00000057	One of the function parameters is not valid.
ERROR_INVALID_NAME 0x0000007B	The file name, directory name, or volume label syntax is incorrect.
ERROR_INVALID_FLAGS 0x000003EC	Reserved contains an invalid value.
RPC_S_PROTSEQ_NOT_SUPPORTED 0x000006A7	The RPC protocol sequence is not supported.
RPC_S_CALL_IN_PROGRESS 0x000006FF	A remote procedure call is already in progress.<99>
DNS_ERROR_INVALID_NAME_CHAR 0x00002558	The Internet host name contains an invalid character.

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

Unless otherwise noted, if the server encounters an error during message processing, it SHOULD revert any state changes made, MUST stop message processing, and MUST return the error to the caller.<100>

These definitions are used in the specification of message processing that follows.

- *NewAlternateNames*: MUST be a new tuple entry for **alternate-computer-names** (section 3.2.1.2).
- *PasswordString*: A UTF-8 string containing a password in cleartext.
- *DomainControllerString*: A UTF-8 string that contains the name of a domain controller in the domain that the server is joining.
- *DomainControllerConnection*: An ADConnection ([MS-ADTS] section 7.3) to a domain controller.
- *WritableDomainControllerDN*: A UTF-8 string that contains the DN of the nTDSDSA object ([MS-ADTS] section 6.1.1.2.2.1.2.1.1) for the domain controller named in *DomainControllerString*.
- *ReadOnlyDomainControllerConnection*: An ADConnection ([MS-ADTS] section 7.3) to a read-only domain controller.
- *LdapResultMessages*: A list of LDAPMessage ([RFC2251]) containing results from an operation performed on DomainControllerConnection.
- *ComputerAccountDN*: A UTF-8 string that contains the DN of the computer account.
- *ComputerAccountExtendedDN*: A UTF-8 string that contains the extended DN ([MS-ADTS] section 3.1.1.3.4.1.5) of the computer account.
- *IsRODC*: A Boolean that is TRUE if the server is a read-only domain controller as specified in [MS-DRSR] section 5.7 and FALSE otherwise.

The following statements define the sequence of message processing operations.

1. The server MUST retrieve the RPC protocol sequence used for the current call, as specified in [MS-RPCE] section 3.1.3.4.1, specifying the server binding handle maintained by the RPC runtime ([C706] section 6.2.1). If that RPC protocol sequence is not **NCACN_NP**, the server SHOULD return **RPC_S_PROTSEQ_NOT_SUPPORTED**.<101>

2. The server MUST check that the caller has been granted access rights using the algorithm specified in the **Access Control Abstract Data Model** (section 3.2.1.1), with *Access Request mask* initialized to **WKSTA_NETAPI_CHANGE_CONFIG**; if not, the server MUST return ERROR_ACCESS_DENIED.

3. The server SHOULD<102> stop message processing and return ERROR_NOT_SUPPORTED if the server is a client Stock Keeping Unit (SKU). Otherwise, message processing continues.

4. The server invokes **AmIRodc** ([MS-DRSR] section 5.7), storing the result in *IsRODC*.

5. If *EncryptedPassword* is NULL or *DomainAccount* is NULL, *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *EncryptedPassword* (section 2.2.5.18). *PasswordString* MUST be equal to the decrypted and decoded value. The decrypted buffer is represented as a **JOINPR_USER_PASSWORD** (section 2.2.5.17). The value of the **Length** member MUST be less than 513; otherwise, message processing is stopped, and the server MUST return ERROR_INVALID_PASSWORD.

6. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the server MUST return an error.

7. The server MUST validate *AlternateName*. The validation [RFC1035] is performed in the following order. Specifically, the name MUST NOT:

   - Be longer than 255 octets.
   - Contain a label longer than 63 octets.
   - Contain two or more consecutive dots.
   - Begin with a dot.

8. ERROR_INVALID_NAME MUST be returned if any condition in the preceding group is violated. Otherwise, *AlternateName* validation continues. The name MUST NOT:

   - Contain a space.
   - Contain any of the following characters:

     ```
     { | } ~ [ \ ] ^ ' : ; < = > ? @ ! " # $ % ^ ` ( ) + / , *
     ```

9. DNS_ERROR_INVALID_NAME_CHAR MUST be returned if any condition in the preceding group is violated. Otherwise, *AlternateName* validation continues.

   - *NewAlternateNames*.FQDN MUST be equal to *AlternateName*.
   - *NewAlternateNames*.NetBIOS MUST be equal to *NewAlternateNames*.FQDN converted to a NetBIOS name.<103>

10. *NewAlternateNames* MUST be appended to the list in **alternate-computer-names** persisted locally such that the set of NetBIOS and Internet host name currently assigned to this computer can be resolved on the network ([RFC1001] and [NIS]). If the append is not successful, steps 11 through 21 are not processed and the server MUST return an error after processing steps 22 and 23.

11. If the server is not joined to a domain ([MS-ADTS] section 6.4), proceed to step 22. Otherwise, the server MUST make the following update in the domain.

12. The server MUST locate a writable domain controller for the domain to which the computer is joined, by invoking the **DsrGetDcNameEx2** method on the local [MS-NRPC] server specifying the following parameters:

    - *ComputerName* = NULL
    - *AccountName* = ComputerNameNetBIOS
    - *AllowableAccountControlBits* = ADS_UF_WORKSTATION_TRUST_ACCOUNT | ADS_UF_SERVER_TRUST_ACCOUNT ([MS-ADTS] section 2.2.16)
    - *DomainName* = DomainNameFQDN
    - *DomainGuid* = NULL
    - *SiteName* = NULL
    - *Flags* = (J | B) ([MS-NRPC] section 3.5.4.3.1).

    If a domain controller cannot be located, steps 13 through 21 are not processed and the server MUST return an error after processing steps 22 and 23.

    Otherwise, DomainControllerString MUST equal the string name of the returned writable domain controller.

13. The server invokes **LDAP Bind** (section 3.2.4.22.2) with the following parameters:

    - *DomainControllerBindTarget*: DomainControllerString
    - *AccountNameForBind*: AccountName
    - *PasswordForBind*: PasswordString
    - *Encrypt*: FALSE
    - *DisallowReferrals*: TRUE

    The result is stored in DomainControllerConnection. If the LDAP bind returns an error, steps 14 through 21 are not processed and the server MUST return the error after processing steps 22 and 23.

14. If *IsRODC* is TRUE, the server invokes **LDAP Bind** with the following parameters:

    - *DomainControllerBindTarget*: **ComputerNameNetBIOS**
    - *AccountNameForBind*: AccountName
    - *PasswordForBind*: PasswordString
    - *Encrypt*: FALSE
    - *DisallowReferrals*: TRUE

    The result is stored in *ReadOnlyDomainControllerConnection*. If the LDAP bind returns an error, steps 15 through 19 are not processed and the server MUST return the error after processing steps 20 and 21.

15. The server invokes **Query Computer Account DN for the Local Machine** (section 3.2.4.22.1), specifying DomainControllerString for the *DomainControllerQueryTarget* parameter, storing the

result in ComputerAccountDN. If the query returns an error, steps 16 through 21 are not processed and the server MUST return the error after processing steps 22 and 23.

16. The server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: DomainControllerConnection
    - *TaskInputRequestMessage*: LDAP modifyRequest message ([RFC2251] section 4.6) as follows:
    - Object: ComputerAccountDN
        - The modification sequence has one list entry, set as follows:
            - First list entry
                - operation: replace
                - modification:
                    - type: msDS-AdditionalDnsHostName
                    - vals: NewAlternateNames.FQDN
        - controls: Sequence of one Control structure, as follows:
            - controlType: LDAP_SERVER_PERMISSIVE_MODIFY_OID ([MS-ADTS] section 3.1.1.3.4.1.8)
            - criticality: FALSE
    - *TaskOutputResultMessages*: LDAPResultMessages

      If the LDAP operation returns an error, steps 17 through 21 are not processed and the server MUST return the error after processing steps 22 and 23.

17. If *IsRODC* is TRUE, the server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: DomainControllerConnection
    - *TaskInputRequestMessage*: LDAP SearchRequest message ([RFC2251] section 4.5.1) as follows:
        - baseObject: DN of the rootDSE (empty string)
        - scope: base
        - filter: ObjectClass=*
        - attributes: dsServiceName
        - derefAliases: neverDerefAliases
        - typesOnly: FALSE
    - *TaskOutputResultMessages*: LDAPResultMessages

      The server MUST process the results returned from the DC in LDAPResultMessages. For the entry (SearchResultEntry, [RFC2251] section 4.5.2) returned by the search in LDAPResultMessages, WritableDomainControllerDN MUST equal the value of the attribute

dsServiceName. If the LDAP operation is not successful, steps 18 and 19 are not processed and processing continues at step 20.

18. If *IsRODC* is TRUE, the server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: DomainControllerConnection
    - *TaskInputRequestMessage*: LDAP SearchRequest message ([RFC2251] section 4.5.1) as follows:
        - baseObject: ComputerAccountDN
        - scope: base
        - filter: ObjectClass=*
        - attributes: distinguishedName, serverReferenceBL
        - derefAliases: neverDerefAliases
        - typesOnly: FALSE
        - controls: Sequence of one Control structure, as follows:
            - controlType: LDAP_SERVER_EXTENDED_DN_OID ([MS-ADTS] section 3.1.1.3.4.1.5)
            - criticality: TRUE
    - *TaskOutputResultMessages*: LDAPResultMessages

      The server MUST process the results returned from the DC in LDAPResultMessages. For the entry (SearchResultEntry, [RFC2251] section 4.5.2) returned by the search in LDAPResultMessages, ComputerAccountExtendedDN MUST equal the value of the attribute distinguishedName unless distinguishedName contains the character ';'. If ';' is present, ComputerAccountExtendedDN MUST equal distinguishedName truncated at the first occurrence of the character ';', exclusive of the ';' itself. If the LDAP operation returns an error, step 19 is not processed and processing continues at step 20.

19. If *IsRODC* is TRUE, the server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: ReadOnlyDomainControllerConnection
    - *TaskInputRequestMessage*: LDAP modifyRequest message ([RFC2251] section 4.6) as follows:
        - Object: DN of the rootDSE (empty string)
        - The modification sequence has one list entry, set as follows:
            - First list entry
                - operation: replace
                - modification:
                    - type: replicateSingleObject ([MS-ADTS] section 3.1.1.3.3.18)
                    - vals: WritableDomainControllerDN:ComputerAccountExtendedDN
    - *TaskOutputResultMessages*: LDAPResultMessages

If the LDAP operation returns an error, processing continues as if it had succeeded.

20. The server invokes **LDAP Unbind** (section 3.2.4.22.3) with ADConnectionToUnbind set to DomainControllerConnection.

21. If *IsRODC* is TRUE, the server invokes **LDAP Unbind** with ADConnectionToUnbind set to ReadOnlyDomainControllerConnection.

22. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).

23. If an error occurred while processing steps 12 through 16, the server removes *NewAlternateNames* from the list in **alternate-computer-names** persisted locally.

If no errors occur, the server MUST return NERR_Success.

3.2.4.19 NetrRemoveAlternateComputerName (Opnum 28)

The **NetrRemoveAlternateComputerName** method removes an alternate name for a specified server.<104>

```
unsigned long NetrRemoveAlternateComputerName(
  [in] handle_t RpcBindingHandle,
  [in, string, unique] wchar_t* ServerName,
  [in, string, unique] wchar_t* AlternateName,
  [in, string, unique] wchar_t* DomainAccount,
  [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD EncryptedPassword,
  [in] unsigned long Reserved
);
```

RpcBindingHandle: An RPC binding handle [C706].

ServerName: This parameter has no effect on message processing in any environment. The client MUST set this parameter to a value that resolves to the IP protocol layer destination address of the RPC packets it transmits ([MS-RPCE] section 2.1.1.2). The server MUST ignore this parameter.

AlternateName: A pointer to a string that specifies the alternate name to remove. The name MUST be a valid DNS host name [RFC1035].

DomainAccount: A pointer to a string that specifies the account name in the domain to use when connecting to a domain controller. This parameter is optional. If this parameter is NULL, the caller's account name MUST be used. If this parameter is specified, the format MUST be one of the following:

- <NetBIOSDomainName>\<UserName>
- <FullyQualifiedDNSDomainName>\<UserName>
- <UserName>@<FullyQualifiedDNSDomainName>

EncryptedPassword: An optional pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** (section 2.2.5.18) structure that specifies the encrypted password to use with the *DomainAccount* parameter. If the *DomainAccount* parameter is NULL, the caller's security context MUST be used, and this parameter MUST be ignored.

Reserved: A 32-bit bitfield that SHOULD be set to zero.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	IU

Where the bits are defined as:

Value	Meaning
IU NET_IGNORE_UNSUPPORTED_FLAGS	If 1, the server MUST ignore the values of the other bits in this field. If 0, the values of the other bits in this field MUST be 0; otherwise, the server MUST return ERROR_INVALID_FLAGS.<105>

Return Values: When the message processing result matches the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_NOT_SUPPORTED 0x00000032	This method is not supported by this server.
ERROR_INVALID_PASSWORD 0x00000056	The specified network password is not correct.
ERROR_INVALID_PARAMETER 0x00000057	One of the function parameters is not valid.
ERROR_INVALID_NAME 0x0000007B	An invalid name parameter is specified.
ERROR_INVALID_FLAGS 0x000003EC	The *Reserved* parameter contains an invalid value.
ERROR_NOT_FOUND 0x00000490	*AlternateName* was not found in the current list of alternate names.
RPC_S_PROTSEQ_NOT_SUPPORTED 0x000006A7	The RPC protocol sequence is not supported.
RPC_S_CALL_IN_PROGRESS 0x000006FF	A remote procedure call is already in progress.<106>
DNS_ERROR_INVALID_NAME_CHAR 0x00002558	The Internet host name contains an invalid character.

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

Unless otherwise noted, if the server encounters an error during message processing, the server SHOULD revert any state changes made, MUST stop message processing, and MUST return the error to the caller.<107>

The following definitions are used in the specification of message processing that follows.

- *OldAlternateNames*: MUST be a tuple entry for **alternate-computer-names** (section 3.2.1.2).
- *PasswordString*: A UTF-8 string containing a password in cleartext.
- *DomainControllerString*: A UTF-8 string that contains the name of a domain controller in the domain that the server is joining.
- *DomainControllerConnection*: An ADCONNECTION_HANDLE ([MS-DTYP] section 2.2.2) to a domain controller.
- *WritableDomainControllerDN*: A UTF-8 string that contains the DN of the nTDSDSA object ([MS-ADTS] section 6.1.1.2.2.1.2.1.1) for the domain controller named in *DomainControllerString*.
- *ReadOnlyDomainControllerConnection*: An ADConnection ([MS-ADTS] section 7.3) to a read-only domain controller.
- *LdapResultMessages*: A list of LDAPMessage ([RFC2251]) containing results from an operation performed on DomainControllerConnection.
- *ComputerAccountDN*: A UTF-8 string that contains the DN of the computer account.
- *ComputerAccountExtendedDN*: A UTF-8 string that contains the extended DN ([MS-ADTS] section 3.1.1.3.4.1.5) of the computer account.
- *IsRODC*: A Boolean that is TRUE if the server is a read-only domain controller as specified in [MS-DRSR] section 5.7 and FALSE otherwise.

The following statements define the sequence of message processing operations.

1. The server MUST retrieve the RPC protocol sequence used for the current call, as specified in [MS-RPCE] section 3.1.3.4.1, specifying the server binding handle maintained by the RPC runtime ([C706] section 6.2.1). If that RPC protocol sequence is not **NCACN_NP**, the server SHOULD return RPC_S_PROTSEQ_NOT_SUPPORTED.<108>
2. The server MUST check that the caller has been granted access rights using the algorithm specified in the **Access Control Abstract Data Model** (section 3.2.1.1), with *Access Request mask* initialized to **WKSTA_NETAPI_CHANGE_CONFIG**; if not, the server MUST return ERROR_ACCESS_DENIED.
3. The server SHOULD<109> return ERROR_NOT_SUPPORTED if the server is a client SKU configuration.
4. The server invokes **AmIRodc** ([MS-DRSR] section 5.7), storing the result in *IsRODC*.
5. If *EncryptedPassword* is NULL or *DomainAccount* is NULL, *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *EncryptedPassword* as defined in section 2.2.5.18. *PasswordString* MUST equal the decrypted and decoded value. The decrypted buffer is represented as a **JOINPR_USER_PASSWORD** (section 2.2.5.17). The value of the **Length** member MUST be less than 513; otherwise, message processing is stopped, and the server MUST return ERROR_INVALID_PASSWORD.
6. The server MUST validate *AlternateName*. The validation [RFC1035] is performed in the following order. Specifically, the name MUST NOT:
    - Be longer than 255 octets.

- Contain a label longer than 63 octets.
- Contain two or more consecutive dots.
- Begin with a dot.

7. ERROR_INVALID_NAME MUST be returned if any condition in the preceding group is violated. Otherwise, *AlternateName* validation continues. The name MUST NOT:

   - Contain a space.
   - Contain any of the following characters:

     ```
     { | } ~ [ \ ] ^ ' : ; < = > ? @ ! " # $ % ^ ` ( ) + / , *
     ```

8. DNS_ERROR_INVALID_NAME_CHAR MUST be returned if any condition in the preceding group is violated. Otherwise, *AlternateName* validation continues.

   - The server MUST locate the tuple *OldAlternateNames* in **alternate-computer-names**, where *OldAlternateNames*.FQDN MUST equal *AlternateName*.
   - *OldAlternateNames*.NetBIOS MUST equal *AlternateName* converted to a NetBIOS name.<110> If tuple *OldAlternateNames* cannot be found the server MUST return ERROR_NOT_FOUND.

9. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the server MUST return an error.

10. If *OldAlternateNames* is found, then *OldAlternateNames* MUST be removed from the list in **alternate-computer-names** persisted locally so that the set of NetBIOS and Internet host names currently assigned to this computer can be resolved on the network ([RFC1001] and [NIS]). If the removal is not successful, steps 11 through 21 are not processed and the server MUST return an error after processing steps 22 and 23.

11. If the server is not joined to a domain ([MS-ADTS] section 6.4), proceed to step 22. Otherwise, the server MUST make the following update in the domain.

12. The server MUST locate a writable domain controller for the domain to which the computer is joined, by invoking the **DsrGetDcNameEx2** method on the local [MS-NRPC] server specifying the following parameters:

    - *ComputerName* = NULL
    - *AccountName* = ComputerNameNetBIOS
    - *AllowableAccountControlBits* = ADS_UF_WORKSTATION_TRUST_ACCOUNT | ADS_UF_SERVER_TRUST_ACCOUNT ([MS-ADTS] section 2.2.16)
    - *DomainName* = DomainNameFQDN
    - *DomainGuid* = NULL
    - *SiteName* = NULL
    - *Flags* = (J | B) ([MS-NRPC] section 3.5.4.3.1).

    If a domain controller cannot be located, steps 13 through 21 are not processed and the server MUST return an error after processing steps 22 and 23.

    Otherwise, *DomainControllerString* MUST equal the string name of the returned writable domain controller.

13. The server invokes **LDAP Bind** (section 3.2.4.22.2) with the following parameters:

    - *DomainControllerBindTarget*: DomainControllerString
    - *AccountNameForBind*: AccountName
    - *PasswordForBind*: PasswordString
    - *Encrypt*: FALSE
    - *DisallowReferrals*: TRUE

    The result is stored in *DomainControllerConnection*. If the LDAP bind returns an error, steps 14 through 21 are not processed, and the server MUST return the error after processing steps 22 and 23.

14. If *IsRODC* is TRUE, the server invokes **LDAP Bind** with the following parameters:

    - *DomainControllerBindTarget*: **ComputerNameNetBIOS**
    - *AccountNameForBind*: AccountName
    - *PasswordForBind*: PasswordString
    - *Encrypt*: FALSE
    - *DisallowReferrals*: TRUE

    The result is stored in *ReadOnlyDomainControllerConnection*. If the LDAP bind is not successful, steps 15 through 21 are not processed and the server MUST return an error after processing steps 22 and 23.

15. The server invokes **Query Computer Account DN for the Local Machine** (section 3.2.4.22.1), specifying DomainControllerString for the *DomainControllerQueryTarget* parameter, storing the result in ComputerAccountDN. If the query returns an error, steps 16 through 21 are not processed and the server MUST return the error after processing steps 22 and 23.

16. The server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: DomainControllerConnection
    - *TaskInputRequestMessage*: LDAP modifyRequest message ([RFC2251] section 4.6) as follows:
        - Object: ComputerAccountDN
        - The modification sequence has one list entry, set as follows:
            - First list entry
                - operation: delete
                - modification:
                    - type: msDS-AdditionalDnsHostName
                    - vals: OldAlternateNames.FQDN
        - controls: Sequence of one Control structure, as follows:
            - controlType: LDAP_SERVER_PERMISSIVE_MODIFY_OID ([MS-ADTS] section 3.1.1.3.4.1.8)

        - criticality: FALSE

    - *TaskOutputResultMessages*: LDAPResultMessages

    If the LDAP operation returns an error, steps 17 through 21 are not processed and the server MUST return the error after processing steps 22 and 23.

17. If *IsRODC* is TRUE, the server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: DomainControllerConnection
    - *TaskInputRequestMessage*: LDAP SearchRequest message ([RFC2251] section 4.5.1) as follows:
        - baseObject: DN of the rootDSE (empty string)
        - scope: base
        - filter: ObjectClass=*
        - attributes: dsServiceName
        - derefAliases: neverDerefAliases
        - typesOnly: FALSE
    - *TaskOutputResultMessages*: LDAPResultMessages

    The server MUST process the results returned from the DC in LDAPResultMessages. For the entry (SearchResultEntry, [RFC2251] section 4.5.2) returned by the search in LDAPResultMessages, the WritableDomainControllerDN MUST equal the value of the attribute dsServiceName. If the LDAP operation returns an error, steps 18 and 19 are not processed and processing continues at step 20.

18. If *IsRODC* is TRUE, the server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: DomainControllerConnection
    - *TaskInputRequestMessage*: LDAP SearchRequest message ([RFC2251] section 4.5.1) as follows:
        - baseObject: ComputerAccountDN
        - scope: base
        - filter: ObjectClass=*
        - attributes: distinguishedName, serverReferenceBL
        - derefAliases: neverDerefAliases
        - typesOnly: FALSE
        - controls: Sequence of one Control structure, as follows:
            - controlType: LDAP_SERVER_EXTENDED_DN_OID ([MS-ADTS] section 3.1.1.3.4.1.5)
            - criticality: TRUE
    - *TaskOutputResultMessages*: LDAPResultMessages

The server MUST process the results returned from the DC in LDAPResultMessages. For the entry (SearchResultEntry, [RFC2251] section 4.5.2) returned by the search in LDAPResultMessages, ComputerAccountExtendedDN MUST equal the value of the attribute distinguishedName unless distinguishedName contains the character ';'. If ';' is present, ComputerAccountExtendedDN MUST equal distinguishedName truncated at the first occurrence of the character ';', exclusive of the ';' itself. If the LDAP operation returns an error, step 19 is not processed and processing continues at step 20.

19. If *IsRODC* is TRUE, the server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:
    - *TaskInputADConnection*: ReadOnlyDomainControllerConnection
    - *TaskInputRequestMessage*: LDAP modifyRequest message ([RFC2251] section 4.6) as follows:
        - Object: DN of the rootDSE (empty string)
        - The modification sequence has one list entry, set as follows:
            - First list entry
                - operation: replace
                - modification:
                    - type: replicateSingleObject ([MS-ADTS] section 3.1.1.3.3.18)
                    - vals: WritableDomainControllerDN:ComputerAccountExtendedDN
    - *TaskOutputResultMessages*: LDAPResultMessages

    If the LDAP operation returns an error, processing continues as if it had succeeded.
20. The server invokes **LDAP Unbind** (section 3.2.4.22.3) with ADConnectionToUnbind set to DomainControllerConnection.
21. If *IsRODC* is TRUE, the server invokes **LDAP Unbind** with ADConnectionToUnbind set to ReadOnlyDomainControllerConnection.
22. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).
23. If an error occurred while processing steps 12 through 16, the server adds *OldAlternateNames* to the list in **alternate-computer-names** persisted locally.

If no errors occur, the server MUST return NERR_Success.

3.2.4.20 NetrSetPrimaryComputerName (Opnum 29)

The **NetrSetPrimaryComputerName** method sets the primary computer name for a specified server.<111>

```
unsigned long NetrSetPrimaryComputerName(
  [in] handle_t RpcBindingHandle,
  [in, string, unique] wchar_t* ServerName,
  [in, string, unique] wchar_t* PrimaryName,
  [in, string, unique] wchar_t* DomainAccount,
  [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD EncryptedPassword,
  [in] unsigned long Reserved
);
```

RpcBindingHandle: An RPC binding handle [C706].

ServerName: This parameter has no effect on message processing in any environment. The client MUST set this parameter to a value that resolves to the IP protocol layer destination address of the RPC packets it transmits ([MS-RPCE] section 2.1.1.2). The server MUST ignore this parameter.

PrimaryName: A pointer to a string that specifies the primary computer name to set. The name MUST be a valid DNS host name [RFC1035].

DomainAccount: A pointer to a string that specifies the account name in the joined domain to use when connecting to a domain controller. This parameter is optional. If this parameter is NULL, the caller's account name MUST be used. This parameter is not used if the server is not joined to a domain.

- <NetBIOSDomainName>\<UserName>
- <FullyQualifiedDNSDomainName>\<UserName>
- <UserName>@<FullyQualifiedDNSDomainName>

EncryptedPassword: An optional pointer to a **JOINPR_ENCRYPTED_USER_PASSWORD** (section 2.2.5.18) structure that specifies the encrypted password to use with the *DomainAccount* parameter. If the *DomainAccount* parameter is NULL, the caller's security context MUST be used, and this parameter MUST be ignored.

Reserved: A 32-bit bitfield that SHOULD be set to zero.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	IU

Where the bits are defined as:

Value	Meaning
IU NET_IGNORE_UNSUPPORTED_FLAGS	If 1, the server MUST ignore the values of the other bits in this field. If 0, the values of the other bits in this field MUST be 0; otherwise, the server MUST return ERROR_INVALID_FLAGS.<112>

Return Values: When the message processing result matches the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_NOT_SUPPORTED 0x00000032	This method is not supported by this server.
ERROR_INVALID_PASSWORD 0x00000056	The specified network password is incorrect.

Value/code	Meaning
ERROR_INVALID_PARAMETER 0x00000057	The parameter is incorrect.
0x0000007B ERROR_INVALID_NAME	An invalid name parameter is specified.
ERROR_INVALID_FLAGS 0x000003EC	Reserved contains an invalid value.
RPC_S_PROTSEQ_NOT_SUPPORTED 0x000006A7	The RPC protocol sequence is not supported.
RPC_S_CALL_IN_PROGRESS 0x000006FF	A remote procedure call is already in progress.<113>
NERR_DefaultJoinRequired 0x00000A86	The destination domain controller does not support creating machine accounts in OUs.
DNS_ERROR_INVALID_NAME_CHAR 0x00002558	The Internet host name contains an invalid character.

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

Unless otherwise noted, if the server encounters an error during message processing, the server SHOULD revert any state changes made, MUST stop message processing, and MUST return the error to the caller.<114>

The following definitions are used in the specification of message processing that follows.

- *OldAlternateNames*: MUST be a tuple entry for **alternate-computer-names** (section 3.2.1.2).
- *NewAlternateNames*: MUST be a new tuple entry for **alternate-computer-names**.
- *NetBIOSNameString*: A Unicode UTF-8 string containing the value of *PrimaryName* converted to a NetBIOS name.
- *I*: Unsigned integer used for indexing **alternate-computer-names**.
- *PasswordString*: A UTF-8 string containing a password in cleartext.
- *DomainControllerString*: A UTF-8 string that contains the name of a domain controller in the domain that the server is joining.
- *DomainControllerConnection*: An ADCONNECTION_HANDLE ([MS-DTYP] section 2.2.2) to a domain controller.
- *WritableDomainControllerDN*: A UTF-8 string that contains the DN of the nTDSDSA object ([MS-ADTS] section 6.1.1.2.2.1.2.1.1) for the domain controller named in *DomainControllerString*.
- *ReadOnlyDomainControllerConnection*: An ADConnection ([MS-ADTS] section 7.3) to a read-only domain controller.
- *LdapResultMessages*: A list of LDAPMessage ([RFC2251]) containing results from an operation performed on DomainControllerConnection.
- *ComputerAccountDN*: A UTF-8 string that contains the DN of the computer account.

- *ComputerAccountExtendedDN*: A UTF-8 string that contains the extended DN ([MS-ADTS] section 3.1.1.3.4.1.5) of the computer account.
- *ServerObjectDN*: A UTF-8 string that contains the DN of the nTDSDSA object ([MS-ADTS] section 6.1.1.2.2.1.2.1.1) for the server when the server is an RODC.
- *IsRODC*: A Boolean that is TRUE if the server is a read-only domain controller as specified in [MS-DRSR] section 5.7 and FALSE otherwise.

The following statements define the sequence of message processing operations.

1. The server MUST retrieve the RPC protocol sequence used for the current call, as specified in [MS-RPCE] section 3.1.3.4.1, specifying the server binding handle maintained by the RPC runtime ([C706] section 6.2.1). If that RPC protocol sequence is not **NCACN_NP**, the server SHOULD return **RPC_S_PROTSEQ_NOT_SUPPORTED**.<115>
2. The server MUST check that the caller has been granted access rights using the algorithm specified in the **Access Control Abstract Data Model** (section 3.2.1.1), with *Access Request mask* initialized to **WKSTA_NETAPI_CHANGE_CONFIG**; if not, the server MUST return ERROR_ACCESS_DENIED.
3. The server SHOULD<116> return ERROR_NOT_SUPPORTED if the server is a client SKU configuration.
4. The server invokes **AmIRODC** ([MS-DRSR] section 5.7), storing the result in *IsRODC*.
5. If *EncryptedPassword* is NULL or *DomainAccount* is NULL, *PasswordString* MUST be NULL. Otherwise, the server MUST decrypt and decode the *EncryptedPassword* as defined in section 2.2.5.18. *PasswordString* MUST equal the decrypted and decoded value. The decrypted buffer is represented as a **JOINPR_USER_PASSWORD** (section 2.2.5.17). The value of the **Length** member MUST be less than 513; otherwise, message processing is stopped, and the server MUST return ERROR_INVALID_PASSWORD.
6. The server MUST validate *PrimaryName*. The validation [RFC1035] is performed in the following order. Specifically, the name MUST NOT:
   - Be longer than 255 octets.
   - Contain a label longer than 63 octets.
   - Contain two or more consecutive dots.
   - Begin with a dot.
7. ERROR_INVALID_NAME MUST be returned if any condition in the preceding group is violated. Otherwise, *PrimaryName* validation continues. The name MUST NOT:
   - Contain a space.
   - Contain any of the following characters:

     ```
     { | } ~ [ \ ] ^ ' : ; < = > ? @ ! " # $ % ^ ` ( ) + / , *
     ```

8. DNS_ERROR_INVALID_NAME_CHAR MUST be returned if any condition in the preceding group is violated. Otherwise, processing continues.
9. The server MUST convert the name in the *PrimaryName* parameter to a string NetBIOS name.<117> This conversion MUST match the conversion used in [MS-NRPC]. *NetBIOSNameString* MUST be equal to this converted value.

10. The server MUST locate the tuple in the list of **alternate-computer-names**, where *alternate-computer-names[I]*.FQDN is equal to *PrimaryName* and *alternate-computer-names[I]*.NetBIOS MUST be equal to *NetBIOSNameString*. *OldAlternateNames* MUST be equal to the tuple identified.

11. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the server MUST return an error.

12. The server MUST remove the tuple located above from the list in **alternate-computer-names** persisted locally.

13. *NewAlternateNames*.FQDN MUST be equal to the current **ComputerNameFQDN**.

14. *NewAlternateNames*.NetBIOS MUST be equal to the current **ComputerNameNetBIOS**.

15. *NewAlternateNames* MUST be appended to the list in **alternate-computer-names** persisted locally.

16. The server MUST set **ComputerNameNetBIOS** to equal *NetBIOSNameString*.

17. The server MUST set **ComputerNameFQDN** to equal *PrimaryName*.

18. The server MUST store the values **ComputerName.NetBIOS**, **ComputerName.FQDN**, and names in **alternate-computer-names** locally so that the set of NetBIOS and Internet host names currently assigned to this computer can be resolved on the network ([RFC1001] and [NIS]). If an error occurs while storing the values, steps 19 through 31 are not processed and the server MUST return the error after processing steps 32 through 34.

19. If the server is not joined to a domain ([MS-ADTS] section 6.4), proceed to step 32. Otherwise, the server MUST make the following updates in the domain.

20. The server MUST stop the Netlogon Remote Protocol ([MS-NRPC]) if it is running. If an error occurs in this operation, steps 21 through 31 are not processed, and the server MUST return the error after processing steps 32 through 34.

21. The server MUST locate a writable domain controller by invoking the **DsrGetDcNameEx2** method on the local [MS-NRPC] server specifying the following parameters:

    - *ComputerName* = NULL
    - *AccountName* = ComputerNameNetBIOS
    - *AllowableAccountControlBits* = ADS_UF_WORKSTATION_TRUST_ACCOUNT | ADS_UF_SERVER_TRUST_ACCOUNT ([MS-ADTS] section 2.2.16)
    - *DomainName* = DomainNameFQDN
    - *DomainGuid* = NULL
    - *SiteName* = NULL
    - *Flags* = (J | B) ([MS-NRPC] section 3.5.4.3.1).

    If a domain controller cannot be located, steps 22 through 31 are not processed and the server MUST return the error after processing steps 32 through 34.

    Otherwise, DomainControllerString MUST equal the string name of the returned writable domain controller.

22. The server invokes **LDAP Bind** (section 3.2.4.22.2) with the following parameters:

    - *DomainControllerBindTarget*: DomainControllerString

- *AccountNameForBind*: AccountName
- *PasswordForBind*: PasswordString
- *Encrypt*: FALSE
- *DisallowReferrals*: TRUE

The result is stored in DomainControllerConnection. If the LDAP bind returns an error, steps 23 through 31 are not processed and the server MUST return the error after processing steps 32 through 34.

23. If *IsRODC* is TRUE, the server invokes **LDAP Bind** with the following parameters:

    - *DomainControllerBindTarget*: **ComputerNameNetBIOS**
    - *AccountNameForBind*: AccountName
    - *PasswordForBind*: PasswordString
    - *Encrypt*: FALSE
    - *DisallowReferrals*: TRUE

    The result is stored in *ReadOnlyDomainControllerConnection*. If the LDAP bind returns an error, steps 24 through 30 are not processed and the server MUST return the error after processing steps 31 through 34.

24. The server invokes **Query Computer Account DN for the Local Machine** (section 3.2.4.22.1), specifying DomainControllerString for the *DomainControllerQueryTarget* parameter, storing the result in ComputerAccountDN. If the query returns an error, steps 25 through 29 are not processed and the server MUST return the error after processing steps 30 through 34.

25. The server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: DomainControllerConnection
    - *TaskInputRequestMessage*: LDAP modifyRequest message ([RFC2251] section 4.6) as follows:
        - Object: ComputerAccountDN
        - The modification sequence has three list entries, set as follows:
            - First list entry
                - operation: replace
                - modification:
                    - type: DnsHostName
                    - vals: ComputerNameFQDN
            - Second list entry
                - operation: add
                - modification:
                    - type: msDS-AdditionalDnsHostName
                    - vals: NewAlternateNames.FQDN

- Third list entry
  - operation: delete
  - modification:
    - type: msDS-AdditionalDnsHostName
    - vals: OldAlternateNames.FQDN
- controls: Sequence of one Control structure, as follows:
  - controlType: LDAP_SERVER_PERMISSIVE_MODIFY_OID ([MS-ADTS] section 3.1.1.3.4.1.8)
  - criticality: FALSE

- *TaskOutputResultMessages*: LDAPResultMessages

If the LDAP operation returns an error, steps 26 through 29 are not processed and the server MUST return the error after processing steps 30 through 34.

26. If *IsRODC* is TRUE, the server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: DomainControllerConnection
    - *TaskInputRequestMessage*: LDAP SearchRequest message ([RFC2251] section 4.5.1) as follows:
      - baseObject: DN of the rootDSE (empty string)
      - scope: base
      - filter: ObjectClass=*
      - attributes: dsServiceName
      - derefAliases: neverDerefAliases
      - typesOnly: FALSE
    - *TaskOutputResultMessages*: LDAPResultMessages

    The server MUST process the results returned from the DC in LDAPResultMessages. For the entry (SearchResultEntry, [RFC2251] section 4.5.2) returned by the search in LDAPResultMessages, WritableDomainControllerDN MUST equal the value of the attribute **dsServiceName**. If the LDAP operation returns an error, steps 27 through 29 are not processed and processing continues at step 30.

27. If *IsRODC* is TRUE, the server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

    - *TaskInputADConnection*: DomainControllerConnection
    - *TaskInputRequestMessage*: LDAP SearchRequest message ([RFC2251] section 4.5.1) as follows:
      - baseObject: ComputerAccountDN
      - scope: base

- filter: ObjectClass=*
- attributes: distinguishedName, serverReferenceBL
- derefAliases: neverDerefAliases
- typesOnly: FALSE
- controls: Sequence of one Control structure, as follows:
  - controlType: LDAP_SERVER_EXTENDED_DN_OID
  - criticality: TRUE

- *TaskOutputResultMessages*: LDAPResultMessages

The server MUST process the results returned from the DC in LDAPResultMessages. For the entry (SearchResultEntry, [RFC2251] section 4.5.2) returned by the search in LDAPResultMessages, ComputerAccountExtendedDN MUST equal the value of the attribute distinguishedName and ServerObjectDN MUST equal the value of the attribute serverReferenceBL, except in cases where the character ';' is present in the attribute. If ';' is present in distinguishedName, ComputerAccountExtendedDN MUST equal distinguishedName truncated at the first occurrence of the character ';', exclusive of the ';' itself. If ';' is present in serverReferenceBL, ServerObjectDN MUST equal serverReferenceBL truncated at the first occurrence of the character ';', exclusive of the ';' itself. If the LDAP operation returns an error, steps 28 and 29 are not processed and processing continues at step 30.

28. If *IsRODC* is TRUE, the server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

- *TaskInputADConnection*: ReadOnlyDomainControllerConnection
- *TaskInputRequestMessage*: LDAP modifyRequest message ([RFC2251] section 4.6) as follows:
  - Object: DN of the rootDSE (empty string)
  - The modification sequence has one list entry, set as follows:
    - First list entry
      - operation: replace
      - modification:
        - type: replicateSingleObject ([MS-ADTS] section 3.1.1.3.3.18)
        - vals: WritableDomainControllerDN:ComputerAccountExtendedDN
- *TaskOutputResultMessages*: LDAPResultMessages

If the LDAP operation returns an error, step 29 is not processed and processing continues at step 30.

29. If *IsRODC* is TRUE, the server invokes the "Performing an LDAP Operation on an ADConnection" task of [MS-ADTS] section 7.6.1.6 with the following parameters:

- *TaskInputADConnection*: ReadOnlyDomainControllerConnection
- *TaskInputRequestMessage*: LDAP modifyRequest message ([RFC2251] section 4.6) as follows:
  - Object: DN of the rootDSE (empty string)

- The modification sequence has one list entry, set as follows:
  - First list entry
    - operation: replace
    - modification:
      - type: replicateSingleObject ([MS-ADTS] section 3.1.1.3.3.18)
      - vals: WritableDomainControllerDN:ServerObjectDN
- *TaskOutputResultMessages*: LDAPResultMessages

If the LDAP operation returns an error, processing continues as if it had succeeded.

30. If *IsRODC* is TRUE, the server invokes **LDAP Unbind** (section 3.2.4.22.3) with ADConnectionToUnbind set to ReadOnlyDomainControllerConnection.
31. The server invokes **LDAP Unbind** with *ADConnectionToUnbind* set to DomainControllerConnection.
32. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).
33. If an error occurred while processing steps 18 through 24, the server:
    - Removes *NewAlternateNames* from the list in **alternate-computer-names** persisted locally.
    - Sets **ComputerNameFQDN** to OldAlternateNames.FQDN.
    - Sets **ComputerNameNetBIOS** to OldAlternateNames.NetBIOS.
    - Appends *OldAlternateNames* to the list in **alternate-computer-names** persisted locally.
    - Stores the values **ComputerName.NetBIOS**, **ComputerName.FQDN**, and names in **alternate-computer-names** locally so that the set of NetBIOS and Internet host names currently assigned to this computer can be resolved on the network ([RFC1001] and [NIS]).
    - Invokes the **Computer Account Update over SAMR** task (section 3.2.4.22.4), specifying the following parameters:
      - *DomainController* = DomainControllerString
      - *CurrentSamAccountName* = *PrimaryName*
      - *NewSamAccountName* = **ComputerNameNetBIOS**
      - *DomainAccount* = *DomainAccount*
      - *DomainAccountPassword* = PasswordString
34. The server MUST start the Netlogon Remote Protocol ([MS-NRPC]) if it was stopped in step 20.

If no errors occur, the server MUST return NERR_Success.

3.2.4.21 NetrEnumerateComputerNames (Opnum 30)

The **NetrEnumerateComputerNames** method returns a list of computer names for a specified server. The results of the query are determined by the type of the name. <118>

```
unsigned long NetrEnumerateComputerNames(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in] NET_COMPUTER_NAME_TYPE NameType,
  [in] unsigned long Reserved,
  [out] PNET_COMPUTER_NAME_ARRAY* ComputerNames
);
```

ServerName: A **WKSSVC_IMPERSONATE_HANDLE** (section 2.2.2.2) that specifies the server. The client MUST map this structure to an RPC binding handle ([C706] sections 4.3.5 and 5.1.5.2). The server MUST ignore this parameter.

NameType: The type of query issued. See **NET_COMPUTER_NAME_TYPE** (section 2.2.3.3).

Reserved: A 32-bit bitfield that SHOULD be set to zero.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	IU

Where the bits are defined as:

Value	Meaning
IU NET_IGNORE_UNSUPPORTED_FLAGS	If 1, the server MUST ignore the values of the other bits in this field. If 0, the values of the other bits in this field MUST be 0; otherwise, the server MUST return ERROR_INVALID_FLAGS. <119>

ComputerNames: A pointer to structure containing a list of computer name strings. See **NET_COMPUTER_NAME_ARRAY** (section 2.2.5.20).

Return Values: When the message processing result meets the description in column two of the following table, this method MUST return one of the following values ([MS-ERREF] section 2.2).

Value/code	Meaning
NERR_Success 0x00000000	The operation completed successfully.
ERROR_ACCESS_DENIED 0x00000005	Access is denied.
ERROR_NOT_ENOUGH_MEMORY 0x00000008	Not enough storage is available to process this command.
ERROR_INVALID_PARAMETER 0x00000057	The parameter is incorrect.
ERROR_NOT_SUPPORTED 0x00000032	This method is not supported by this server.
ERROR_INVALID_FLAGS 0x000003EC	Reserved contains an invalid value.
RPC_S_PROTSEQ_NOT_SUPPORTED	The RPC protocol sequence is not supported.

Value/code	Meaning
0x000006A7	
RPC_S_CALL_IN_PROGRESS 0x000006FF	A remote procedure call is already in progress.

Any other return value MUST conform to the error code requirements specified in **Protocol Details** (section 3).

The following statements define the sequence of message processing operations.

1. The server MUST retrieve the RPC protocol sequence used for the current call, as specified in [MS-RPCE] section 3.1.3.4.1, specifying the server binding handle maintained by the RPC runtime ([C706] section 6.2.1). If that RPC protocol sequence is not **NCACN_NP**, the server SHOULD return **RPC_S_PROTSEQ_NOT_SUPPORTED**.<120>

2. The server MUST check that the caller has been granted access rights using the algorithm specified in the **Access Control Abstract Data Model** (section 3.2.1.1), with *Access Request mask* initialized to **WKSTA_NETAPI_CHANGE_CONFIG**; if not, the server MUST return ERROR_ACCESS_DENIED.

3. The server SHOULD<121> return ERROR_NOT_SUPPORTED if the server is a client SKU configuration.

4. The server MUST return ERROR_INVALID_PARAMETER if *NameType* is greater than or equal to **NetComputerNameTypeMax**.

5. The server MUST impersonate the client by invoking the **StartImpersonatingClient** task (section 3.2.4.22.6). If this operation fails, the server MUST return an error.

6. The server MUST initialize the output parameter *ComputerNames* as follows depending on the input query type specified in *NameType*.

   - **NetPrimaryComputerName**:

     The server MUST set *ComputerNames.EntryCount* to 1 and initialize the UNICODE_STRING values in *ComputerNames* to **ComputerNameFQDN**.

   - **NetAlternateComputerNames**:

     The server MUST set *ComputerNames.EntryCount* to the number of tuples contained in **alternate-computer-names** (section 3.2.1.2). For each tuple, I, the server MUST initialize the next available UNICODE_STRING elements in the *ComputerNames* array to equal the values stored in *alternate-computer-names[I]*.FQDN.

   - **NetAllComputerNames**:

     The server MUST set *ComputerNames.EntryCount* to the number of tuples contained in **alternate-computer-names** + 1. The *ComputerNames* array MUST be initialized to return all the names specified for the **NetPrimaryComputerName** and **NetAlternateComputerNames** input query types.

7. The server MUST stop impersonating the client by invoking the **StopImpersonatingClient** task (section 3.2.4.22.7).

If no errors occur, the server MUST return NERR_Success.

3.2.4.22 Common Message Processing

The following sections represent common processing tasks used by several of the above opnums.

3.2.4.22.1 Query Computer Account DN for the Local Machine

This task accepts as input the following:

- DomainControllerQueryTarget: the name of the domain controller to query.

Upon success, this task returns the following:

- ComputerAccountDN: A UTF-8 string that contains the DN of the computer account DN for the local machine.

Otherwise, a failure is returned.

This task executes as follows:

1. The server MUST bind to the DRS RPC endpoint ([MS-DRSR] section 2.1) on DomainControllerQueryTarget.
2. The server MUST invoke the **IDL_DRSCrackNames** method ([MS-DRSR] section 4.1.4) with the following parameter values:
    - *rpNames* = DomainNameNetBIOS "\" ComputerNameNetBIOS
    - *formatOffered* = DS_NT4_ACCOUNT_NAME
    - *formatDesired* = DS_FQDN_1779_NAME
3. If step 2 succeeds, and only one result was returned, the task sets ComputerAccountDN equal to the DN returned from the **IDL_DRSCrackNames** call, and returns it. Otherwise, the task returns an error.

3.2.4.22.2 LDAP Bind

This task accepts as input the following:

- *DomainControllerBindTarget*: the name of the domain controller to bind to
- *AccountNameForBind*: the account name used for authentication on the bind
- *PasswordForBind*: the password used to authenticate the bind
- *Encrypt*: specifies whether to set LDAP_OPT_ENCRYPT to LDAP_OPT_ON on the returned connection
- *DisallowReferrals*: specifies whether to set LDAP_OPT_REFERRALS to LDAP_OPT_OFF on the returned connection

Upon success, this task returns the following:

- NewADConnection: an ADCONNECTION_HANDLE ([MS-DTYP] section 2.2.2).

This task executes as follows:

1. The server invokes the "Initializing an ADConnection" task of [MS-ADTS] section 7.6.1.1 with the following parameters:
    - *TaskInputTargetName*: DomainControllerBindTarget

- *TaskInputPortNumber*: 389

Upon success, the result is stored in NewADConnection.

2. The server invokes the "Setting an LDAP option on an ADConnection" task ([MS-ADTS] section 7.6.1.2) with the following parameters:
   - *TaskInputADConnection*: NewADConnection
   - *TaskInputOptionName*: LDAP_OPT_AUTH_INFO
   - *TaskInputOptionValue*:
     - bindMethod: SASL, using the GSS-SPNEGO protocol ([MS-ADTS] section 3.1.1.3.4.5.2)
     - name: *AccountNameForBind*
     - password: *PasswordForBind*
3. The server invokes the "Setting an LDAP option on an ADConnection" task ([MS-ADTS] section 7.6.1.2) with the following parameters:
   - *TaskInputADConnection*: NewADConnection
   - *TaskInputOptionName*: LDAP_OPT_AREC_EXCLUSIVE
   - *TaskInputOptionValue*: TRUE
4. If Encrypt is equal to TRUE, the server invokes the "Setting an LDAP option on an ADConnection" task ([MS-ADTS] section 7.6.1.2) with the following parameters:
   - *TaskInputADConnection*: NewADConnection
   - *TaskInputOptionName*: LDAP_OPT_ENCRYPT
   - *TaskInputOptionValue*: LDAP_OPT_ON
5. If DisallowReferrals is equal to TRUE, the server invokes the "Setting an LDAP option on an ADConnection" task ([MS-ADTS] section 7.6.1.2) with the following parameters:
   - *TaskInputADConnection*: NewADConnection
   - *TaskInputOptionName*: LDAP_OPT_REFERRALS
   - *TaskInputOptionValue*: LDAP_OPT_OFF
6. The server invokes the "Establishing an ADConnection" task ([MS-ADTS] section 7.6.1.3) with the *TaskInputADConnection* parameter set to NewADConnection.
7. The server invokes the "Performing an LDAP Bind on an ADConnection" task ([MS-ADTS] section 7.6.1.4) with the *TaskInputADConnection* parameter set to NewADConnection.
8. Upon success, NewADConnection is returned to the caller. Otherwise, an error is returned.

3.2.4.22.3 LDAP Unbind

This task accepts as input the following:

- ADConnectionToUnbind: an ADCONNECTION_HANDLE ([MS-DTYP] section 2.2.2) to unbind.

This task executes as follows:

1. The server invokes the "Performing an LDAP Unbind on an ADConnection" task ([MS-ADTS] section 7.6.1.5) with the *TaskInputADConnection* parameter set to ADConnectionToUnbind.

3.2.4.22.4 Computer Account Update over SAMR

This task accepts as input the following:

- *DomainController*: the name of the domain controller on which to update the computer account.
- *CurrentSamAccountName*: the SAM account name of the computer.
- *NewSamAccountName*: the new SAM account name of the computer.
- *DomainAccount*: the domain account to be used for accessing computer account object in the directory service.
- *DomainAccountPassword*: the password that matches *DomainAccount*.

The following definitions are used in the specification of message processing that follows:

- *LocalSMBSession*: Contains the SMB state for the SMB/CIFS session established to the domain controller.
- *LocalServerHandle*: Contains the RPC context handle representing a SAM RPC server object.
- *LocalDomainHandle*: Contains the RPC context handle representing a domain object.
- *LocalUserHandle*: Contains the RPC context handle representing a user object.

This task executes as follows:

1. The server MUST establish an authenticated SMB/CIFS session to the IPC$ share on the *DomainController* domain controller by invoking [MS-CIFS] section 3.4.4.7, specifying the following parameters:

   - ServerName = *DomainController*
   - UserCredentials = *DomainAccount* \ *DomainAccountPassword*

   Upon success, the server MUST store the result in *LocalSMBSession*.

2. The server MUST update the SAM account name with *NewSamAccountName* using the following steps:

   1. The server MUST bind to the named pipe endpoint \PIPE\samr, as shown in [MS-SAMR] section 2.1.
   2. The server MUST connect to the SAM RPC server on the domain controller using one of the **SamrConnect** variants. See [MS-SAMR] section 1.7.2 for information about invoking the **SamrConnect** variants in order to determine the version and method supported by the RPC server. See [MS-SAMR] section 3.1.5.1 for using the Open pattern in the SAM interface.

      - ServerName = *DomainController*
      - DesiredAccess = GENERIC_ALL

      Upon success, the server MUST store the result in *LocalServerHandle*.

   3. The server MUST call **SamrLookupDomainInSamServer** ([MS-SAMR] section 3.1.5.11.1) to retrieve *LocalDomainSID* specifying the following parameters:

      - ServerHandle = *LocalServerHandle*

- Name = the name of the local machine

4. The server MUST call **SamrOpenDomain** ([MS-SAMR] section 3.1.5.1.5) specifying the following parameters:

   - DesiredAccess = GENERIC_ALL

   - DomainId = the domain security identifier (SID) obtained from prior step

   Upon success, the server MUST store the result in *LocalDomainHandle*.

5. The server MUST call **SamrLookupNamesInDomain** ([MS-SAMR] section 3.1.5.11.2) specifying the following parameters:

   - DomainHandle = *LocalDomainHandle*

   - Names = *CurrentSamAccountName*

6. The server MUST call **SamrOpenUser** ([MS-SAMR] section 3.1.5.1.9) to obtain a handle to the computer account specifying the following parameters:

   - DomainHandle = the domain handle obtained from step 5

   - DesiredAccess = 0x0

   - UserId = the relative ID obtained from prior step

   Upon success, the server MUST store the result in *LocalUserHandle*.

7. The server MUST call **SamrSetInformationUser** ([MS-SAMR] section 3.1.5.6.5) specifying the following parameters:

   - UserHandle = *LocalUserHandle*

   - UserInformationClass = **UserAllInformation** (specified in [MS-SAMR] section 2.2.7.28)

   - Buffer = a buffer of type **SAMPR_USER_ALL_INFORMATION** that contains *NewSamAccountName*. See [MS-SAMR] section 2.2.7.6 for structure details.

8. Regardless of whether an error was encountered in any of the preceding calls, any SAM RPC domain controller handles opened MUST be closed using **SamrCloseHandle** method ([MS-SAMR] section 3.1.5.13.1).

3. The server MUST disconnect the SMB/CIFS session as specified in [MS-CIFS] section 3.2.4.24 specifying *LocalSMBSession*.

3.2.4.22.5 Update Display Name Using SAMR

This task accepts as input the following:

- *DomainController*: the domain controller to perform the update on

- *MachineName*: the name of the machine account to update

This task returns no results to the caller. It executes as follows:

1. The server MUST invoke the **SamrConnect5** method on *DomainController*, specifying the following parameters:

   - *DesiredAccess*: SAM_SERVER_LOOKUP_DOMAIN | SAM_SERVER_ENUMERATE_DOMAINS

2. The server MUST invoke the **SamrOpenDomain** method on *DomainController*, specifying the following parameters:

   - *ServerHandle*: The handle obtained in step 1
   - *DesiredAccess*: DOMAIN_LOOKUP
   - *DomainId*: **DomainSid** (defined in section 3.2.1.6)

3. The server MUST invoke the **SamrLookupNamesInDomain** method on *DomainController*, specifying the following parameters:

   - *DomainHandle*: The handle obtained in step 2
   - *Count*: 1
   - *Names*: MachineName

4. The server MUST invoke the **SamrOpenUser** method on *DomainController*, specifying the following parameters:

   - *DomainHandle*: The handle obtained in step 2
   - *DesiredAccess*: USER_READ_GENERAL | USER_WRITE_ACCOUNT
   - *UserId*: The ID of the account obtained in step 3

5. The server MUST invoke the **SamrQueryInformationUser2** method on *DomainController*, specifying the following parameters:

   - *UserHandle*: The handle obtained in step 4
   - *UserInformationClass*: **UserAllInformation** ([MS-SAMR] section 2.2.7.28)

6. If the **FullName** field of the **SAMPR_USER_ALL_INFORMATION** structure returned in step 5 is equal to *MachineName* using a case-insensitive comparison, the server MUST continue executing at step 8. Otherwise, execution continues at step 7.

7. The server MUST invoke the **SamrSetInformationUser2** method on *DomainController*, specifying the following parameters:

   - *UserHandle*: The handle obtained in step 4
   - *UserInformationClass*: **UserAllInformation** ([MS-SAMR] section 2.2.7.28)
   - *Buffer*: A **SAMPR_USER_INFO_BUFFER** structure ([MS-SAMR] section 2.2.7.29), with the **WhichFields** field of the embedded **SAMPR_USER_ALL_INFORMATION** structure set to USER_ALL_FULLNAME ([MS-SAMR] section 2.2.1.8), and the **FullName** field of the same structure set to *MachineName*.

8. The server MUST invoke **SamrCloseHandle** on *DomainController*, specifying the user handle obtained in step 4.

9. The server MUST invoke **SamrCloseHandle** on *DomainController*, specifying the domain handle obtained in step 2.

10. The server MUST invoke **SamrCloseHandle** on *DomainController*, specifying the server handle obtained in step 1.

3.2.4.22.6 StartImpersonatingClient

This task accepts no inputs. It executes as follows:

- The server MUST invoke the **RpcImpersonateClient** abstract interface ([MS-RPCE] section 3.3.3.4.3.2), specifying NULL for the *BindingHandle* parameter. The result MUST be returned to the caller.

3.2.4.22.7 StopImpersonatingClient

This task accepts no inputs and does not return any results to the caller. It executes as follows:

- The server MUST stop impersonating the client by invoking the **RpcRevertToSelf** abstract interface ([MS-RPCE] section 3.3.3.4.3.3).

3.2.5 Timer Events

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

3.2.6 Other Local Events

The first two of the following subsections specify local events that are invoked by the browser server and used by the Common Internet File System (CIFS) Browser Protocol [MS-BRWS] and the Common Internet File System (CIFS) Browser Auxiliary Protocol [MS-BRWSA].

The final two of the following subsections specify local events to control redirection pause and redirection resume, and can only be invoked by an Administrator.

3.2.6.1 WkstaQueryOtherDomains Event

The calling application requests the list of **OtherDomains** from the server. The calling application provides no parameters, and the server returns the **OtherDomains** list, as specified in the **OtherDomains Name Abstract Data Model** (section 3.2.1.3).

3.2.6.2 WkstaAddOtherDomains Event

The calling application provides a list of NetBIOS domains in the format specified in the **OtherDomains Name Abstract Data Model** (section 3.2.1.3). The server appends entries to the **OtherDomains** list that are not already present.

3.2.6.3 Administrator Requests Redirection to Be Paused

If the administrator requests to pause redirection of printer and serial communication, the server sets **IsWorkstationPaused** to TRUE, as specified in section 3.2.1.

3.2.6.4 Administrator Requests Redirection to Be Resumed

If administrator requests to resume redirection of printer and serial communication, the server sets **IsWorkstationPaused** to FALSE, as specified in section 3.2.1.

4 Protocol Examples

4.1 NetrWkstaGetInfo Example

As an example, the client calls the **NetrWkstaGetInfo** (section 3.2.4.1) method on a server named srvr1.example.com.

```
NetrWkstaGetInfo (
    [in,string,unique] WKSSVC_IDENTIFY_HANDLE ServerName
            = "srvr1.example.com",
    [in] unsigned long Level = 0x00000064,
    [out, switch_is(Level)] LPWKSTA_INFO WkstaInfo
    );
```

On receiving this method the server executes the method locally and returns

```
(type unsigned long)return_status = NERR_Success
NetrWkstaGetInfo (
    [in,string,unique] WKSSVC_IDENTIFY_HANDLE ServerName
            = {unchanged},
    [in] unsigned long Level = {unchanged},
    [out, switch_is(Level)] LPWKSTA_INFO WkstaInfo
            = {filled in as shown below}
    );
```

where *WkstaInfo* is set as follows.

```
typedef struct _WKSTA_INFO_100 {
    unsigned long wki100_platform_id = 0x000001F4;
    wchar_t* wki100_computername = "srvr1.example.com";
    wchar_t* wki100_langroup = "example.com";
    unsigned long wki100_ver_major = 0x00000005;
    unsigned long wki100_ver_minor = 0x00000000;
} WKSTA_INFO_100, *PWKSTA_INFO_100, *LPWKSTA_INFO_100;
```

4.2 NetrWkstaUserEnum Example

In this example, the client calls the **NetrWkstaUserEnum** (section 3.2.4.3) method to enumerate the names of currently logged-on users on a server named "SrvrA". Five active users are logged on to server "SrvrA".

The client calls **NetrWkstaUserEnum** with *ServerName* equal to "SrvrA" and the **Level** field of the **WKSTA_USER_ENUM_STRUCT** (section 2.2.5.14) structure passed in the *UserInfo* parameter set to 0x00000000. The client also sets the *PreferredMaximumLength* parameter to 0x00000100 and passes a non-NULL pointer in parameters *TotalEntries* and *ResumeHandle*.

Only the names of the first two logged-on users fit into 0x00000100 bytes. On receiving this method, the server executes the method locally and returns ERROR_MORE_DATA. The server returns the names of the first two logged-on users in the *UserInfo* parameter. It also sets the value of *TotalEntries* to 0x00000005 and *ResumeHandle* to 0x00000120. The value of *ResumeHandle* is implementation-specific.

To continue enumerating the names of logged-on users, the client calls **NetrWkstaUserEnum** with *ServerName* equal to "SrvrA", and the **Level** field of the **WKSTA_USER_ENUM_STRUCT** structure passed in the *UserInfo* parameter set to 0x00000000. The client also sets the

PreferredMaximumLength parameter to **MAX_PREFERRED_LENGTH** (section 2.2.1.3) 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 NERR_Success. The server returns the names of the next three logged-on users in the *UserInfo* parameter. It also sets the value of *TotalEntries* to 0x00000005. The value of *ResumeHandle* is irrelevant.

4.3 NetrJoinDomain2 Example

In this example, "SrvrA" is a machine that is not joined to a domain, and there exists a domain with the name "DomainA" and a domain controller of that domain named "DC-A".

A client calls the NetrJoinDomain2 (section 3.2.4.13) method on the server named "SrvrA":

```
unsigned long NetrJoinDomain2(
   [in] handle_t RpcBindingHandle,
   [in, string, unique] wchar_t* ServerName = { "SrvrA" },
   [in, string] wchar_t* DomainName = { "DomainA\DC-A" },
   [in, string, unique] wchar_t* MachineAccountOU = { NULL },
   [in, string, unique] wchar_t* AccountName = { NULL },
   [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password = { NULL },
   [in] unsigned long Options = { 0x00000001 }
);
```

Upon receiving this message, "SrvrA" establishes an SMB session with the domain controller "DC-A", using the credentials presented. In this example, there are no credentials; therefore the session is established as an anonymous session.

The server then queries for the domain_name and security identifier (SID) properties of "DC-A", and stores them locally.

As a last step, "SrvrA" creates a domain-object as specified in [MS-ADTS] section 6.4.2 for itself, and sets the following attributes:

- sAMAccountName
- userAccountControl
- unicodePwd
- dNSHostName
- servicePrincipalName

The server then responds to the client with the following message:

```
unsigned long = { 0 }
NetrJoinDomain2(
   [in] handle_t RpcBindingHandle,
   [in, string, unique] wchar_t* ServerName,
   [in, string] wchar_t* DomainName,
   [in, string, unique] wchar_t* MachineAccountOU,
   [in, string, unique] wchar_t* AccountName,
   [in, unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
   [in] unsigned long Options
);
```

The following sequence diagram shows the sequence of messages that can be exchanged as a result of a **NetrJoinDomain2** message sent to the server.

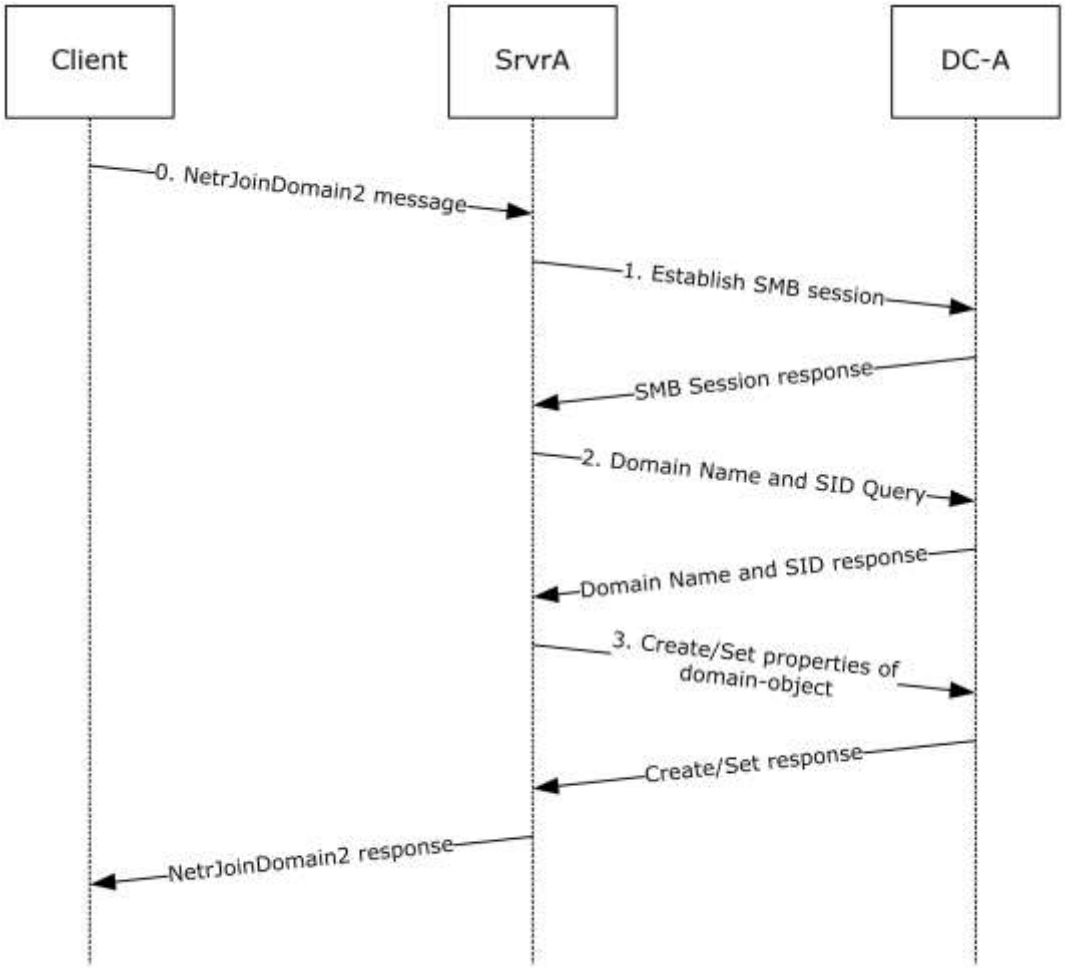

Figure 6: NetrJoinDomain2 sequence

The following state changes occur as a part of this message:

- Initial states:

SrvrA	DC-A
DomainNameNetBIOS = "Workgroup". DomainSID = NULL. Domain-Secret-Value = NULL.	DomainNameNetBIOS = "DomainA". DomainSID = { SID-Value}.

- End states:

SrvrA	DC-A
DomainNameNetBIOS = "DomainA". DomainSID = {SID-Value}. Domain-Secret-Value = { domain-secret }.	DomainNameNetBIOS = "DomainA". DomainSID = { SID-Value }. A computer account object exists with the following LDAP attributes:

SrvrA	DC-A

	▪ userAccountControl = USER_WORKSTATION_TRUST_ACCOUNT.
	▪ samAccountName = "SrvrA$".
	▪ unicodePwd = { domain-secret }.
	▪ dNSHostName = "SrvrA.<DomainA-DNS-Name>.
	▪ servicePrincipalName = {
	"HOST/SrvrA",
	"HOST/SrvrA.<DomainA-DNS-Name>"
	}

5 Security

5.1 Security Considerations for Implementers

As specified in section 2.1, this protocol allows any user to connect to the server. Therefore, any security bug in the server implementation could be exploitable. It is recommended that the server implementation enforce security on each method.

There is only one security parameter, Authentication Protocol (section 2.1).

5.2 Entropy Sources

How entropy is acquired is up to the implementer of any protocol. The literature on measurement of entropy and on methods of harvesting entropy in computer systems is extensive and well known to anyone skilled in the cryptographic art. Probably the best entropy source is a properly verified hardware random bit generator that has circuitry attached to monitor all bits produced and verify their entropy, raising an error condition if the hardware starts to malfunction. Such a hardware source of entropy can be used to drive a conditioning function (sometimes called a "whitening" function) and might be used to drive a pseudo-random number generator (PRNG) that is compliant with recognized standards, such as FIPS 140-2 Annex C [FIPS140].

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided below, where "ms-dtyp.idl" is the IDL found in [MS-DTYP] Appendix A.

```
import "ms-dtyp.idl";

[
    uuid(6BFFD098-A112-3610-9833-46C3F87E345A),version(1.0),
    pointer_default(unique)
]
interface wkssvc
{
    typedef enum _NETSETUP_JOIN_STATUS {
        NetSetupUnknownStatus = 0,
        NetSetupUnjoined,
        NetSetupWorkgroupName,
        NetSetupDomainName
    } NETSETUP_JOIN_STATUS, *PNETSETUP_JOIN_STATUS;

    typedef enum  _NETSETUP_NAME_TYPE {
        NetSetupUnknown = 0,
        NetSetupMachine,
        NetSetupWorkgroup,
        NetSetupDomain,
        NetSetupNonExistentDomain,
        NetSetupDnsMachine
    } NETSETUP_NAME_TYPE, *PNETSETUP_NAME_TYPE;

    typedef enum _NET_COMPUTER_NAME_TYPE {
        NetPrimaryComputerName = 0,
        NetAlternateComputerNames,
        NetAllComputerNames,
        NetComputerNameTypeMax
    } NET_COMPUTER_NAME_TYPE, *PNET_COMPUTER_NAME_TYPE;

    typedef struct _STAT_WORKSTATION_0 {
        LARGE_INTEGER   StatisticsStartTime;
        LARGE_INTEGER   BytesReceived;
        LARGE_INTEGER   SmbsReceived;
        LARGE_INTEGER   PagingReadBytesRequested;
        LARGE_INTEGER   NonPagingReadBytesRequested;
        LARGE_INTEGER   CacheReadBytesRequested;
        LARGE_INTEGER   NetworkReadBytesRequested;
        LARGE_INTEGER   BytesTransmitted;
        LARGE_INTEGER   SmbsTransmitted;
        LARGE_INTEGER   PagingWriteBytesRequested;
        LARGE_INTEGER   NonPagingWriteBytesRequested;
        LARGE_INTEGER   CacheWriteBytesRequested;
        LARGE_INTEGER   NetworkWriteBytesRequested;
        unsigned long   InitiallyFailedOperations;
        unsigned long   FailedCompletionOperations;
        unsigned long   ReadOperations;
        unsigned long   RandomReadOperations;
        unsigned long   ReadSmbs;
        unsigned long   LargeReadSmbs;
        unsigned long   SmallReadSmbs;
        unsigned long   WriteOperations;
        unsigned long   RandomWriteOperations;
        unsigned long   WriteSmbs;
        unsigned long   LargeWriteSmbs;
        unsigned long   SmallWriteSmbs;
        unsigned long   RawReadsDenied;
        unsigned long   RawWritesDenied;
        unsigned long   NetworkErrors;
        unsigned long   Sessions;
        unsigned long   FailedSessions;
```

```
    unsigned long   Reconnects;
    unsigned long   CoreConnects;
    unsigned long   Lanman20Connects;
    unsigned long   Lanman21Connects;
    unsigned long   LanmanNtConnects;
    unsigned long   ServerDisconnects;
    unsigned long   HungSessions;
    unsigned long   UseCount;
    unsigned long   FailedUseCount;
    unsigned long   CurrentCommands;
} STAT_WORKSTATION_0, *PSTAT_WORKSTATION_0, *LPSTAT_WORKSTATION_0;

typedef struct _WKSTA_INFO_100 {
    unsigned long  wki100_platform_id;
    [string] wchar_t* wki100_computername;
    [string] wchar_t* wki100_langroup;
    unsigned long wki100_ver_major;
    unsigned long wki100_ver_minor;
} WKSTA_INFO_100, *PWKSTA_INFO_100, *LPWKSTA_INFO_100;

typedef struct _WKSTA_INFO_101 {
    unsigned long wki101_platform_id;
    [string] wchar_t* wki101_computername;
    [string] wchar_t* wki101_langroup;
    unsigned long wki101_ver_major;
    unsigned long wki101_ver_minor;
    [string] wchar_t* wki101_lanroot;
} WKSTA_INFO_101, *PWKSTA_INFO_101, *LPWKSTA_INFO_101;

typedef struct _WKSTA_INFO_102 {
    unsigned long  wki102_platform_id;
    [string] wchar_t* wki102_computername;
    [string] wchar_t* wki102_langroup;
    unsigned long wki102_ver_major;
    unsigned long wki102_ver_minor;
    [string] wchar_t* wki102_lanroot;
    unsigned long  wki102_logged_on_users;
} WKSTA_INFO_102, *PWKSTA_INFO_102, *LPWKSTA_INFO_102;

typedef struct _WKSTA_INFO_502{
    unsigned long   wki502_char_wait;
    unsigned long   wki502_collection_time;
    unsigned long   wki502_maximum_collection_count;
    unsigned long   wki502_keep_conn;
    unsigned long   wki502_max_cmds;
    unsigned long   wki502_sess_timeout;
    unsigned long   wki502_siz_char_buf;
    unsigned long   wki502_max_threads;
    unsigned long   wki502_lock_quota;
    unsigned long   wki502_lock_increment;
    unsigned long   wki502_lock_maximum;
    unsigned long   wki502_pipe_increment;
    unsigned long   wki502_pipe_maximum;
    unsigned long   wki502_cache_file_timeout;
    unsigned long   wki502_dormant_file_limit;
    unsigned long   wki502_read_ahead_throughput;
    unsigned long   wki502_num_mailslot_buffers;
    unsigned long   wki502_num_srv_announce_buffers;
    unsigned long   wki502_max_illegal_datagram_events;
    unsigned long   wki502_illegal_datagram_event_reset_frequency;
    int  wki502_log_election_packets;
    int  wki502_use_opportunistic_locking;
    int  wki502_use_unlock_behind;
    int  wki502_use_close_behind;
    int  wki502_buf_named_pipes;
    int  wki502_use_lock_read_unlock;
    int  wki502_utilize_nt_caching;
    int  wki502_use_raw_read;
    int  wki502_use_raw_write;
    int  wki502_use_write_raw_data;
```

```
    int  wki502_use_encryption;
    int  wki502_buf_files_deny_write;
    int  wki502_buf_read_only_files;
    int  wki502_force_core_create_mode;
    int  wki502_use_512_byte_max_transfer;
} WKSTA_INFO_502, *PWKSTA_INFO_502, *LPWKSTA_INFO_502;

typedef struct _WKSTA_INFO_1013 {
     unsigned long  wki1013_keep_conn;
} WKSTA_INFO_1013, *PWKSTA_INFO_1013, *LPWKSTA_INFO_1013;

typedef struct _WKSTA_INFO_1018 {
     unsigned long  wki1018_sess_timeout;
} WKSTA_INFO_1018, *PWKSTA_INFO_1018, *LPWKSTA_INFO_1018;

typedef struct _WKSTA_INFO_1046 {
    unsigned long  wki1046_dormant_file_limit;
} WKSTA_INFO_1046, *PWKSTA_INFO_1046, *LPWKSTA_INFO_1046;

typedef struct _WKSTA_USER_INFO_0 {
    [string] wchar_t* wkui0_username;
} WKSTA_USER_INFO_0, *PWKSTA_USER_INFO_0, *LPWKSTA_USER_INFO_0;

typedef struct _WKSTA_USER_INFO_1 {
    [string]  wchar_t* wkui1_username;
    [string] wchar_t* wkui1_logon_domain;
    [string] wchar_t* wkui1_oth_domains;
    [string] wchar_t* wkui1_logon_server;
} WKSTA_USER_INFO_1, *PWKSTA_USER_INFO_1, *LPWKSTA_USER_INFO_1;

typedef struct _WKSTA_TRANSPORT_INFO_0 {
    unsigned long wkti0_quality_of_service;
    unsigned long wkti0_number_of_vcs;
    [string] wchar_t* wkti0_transport_name;
    [string] wchar_t* wkti0_transport_address;
    unsigned long wkti0_wan_ish;
} WKSTA_TRANSPORT_INFO_0, *PWKSTA_TRANSPORT_INFO_0,
 *LPWKSTA_TRANSPORT_INFO_0;

typedef [handle] wchar_t* WKSSVC_IDENTIFY_HANDLE;

typedef [handle] wchar_t* WKSSVC_IMPERSONATE_HANDLE;

typedef [switch_type(unsigned long)] union _WKSTA_INFO {
    [case(100)]
        LPWKSTA_INFO_100 WkstaInfo100;
    [case(101)]
        LPWKSTA_INFO_101 WkstaInfo101;
    [case(102)]
        LPWKSTA_INFO_102 WkstaInfo102;
    [case(502)]
        LPWKSTA_INFO_502 WkstaInfo502;
    [case(1013)]
        LPWKSTA_INFO_1013 WkstaInfo1013;
    [case(1018)]
        LPWKSTA_INFO_1018 WkstaInfo1018;
    [case(1046)]
        LPWKSTA_INFO_1046 WkstaInfo1046;
    [default]
        ;
} WKSTA_INFO, *PWKSTA_INFO, *LPWKSTA_INFO;

typedef struct _USE_INFO_0 {
    [string] wchar_t* ui0_local;
    [string] wchar_t* ui0_remote;
} USE_INFO_0, *PUSE_INFO_0, *LPUSE_INFO_0;

typedef struct _USE_INFO_1 {
    [string] wchar_t* ui1_local;
    [string] wchar_t* ui1_remote;
```

```
    [string] wchar_t* ui1_password;
    unsigned long ui1_status;
    unsigned long ui1_asg_type;
    unsigned long ui1_refcount;
    unsigned long ui1_usecount;
} USE_INFO_1, *PUSE_INFO_1, *LPUSE_INFO_1;

typedef struct _USE_INFO_2 {
    USE_INFO_1 ui2_useinfo;
    [string] wchar_t* ui2_username;
    [string] wchar_t* ui2_domainname;
} USE_INFO_2, *PUSE_INFO_2, *LPUSE_INFO_2;

typedef struct _USE_INFO_3 {
    USE_INFO_2 ui3_ui2;
    ULONG ui3_flags;
} USE_INFO_3, *PUSE_INFO_3, *LPUSE_INFO_3;

typedef [switch_type(unsigned long)] union _USE_INFO {
    [case(0)]
        LPUSE_INFO_0 UseInfo0;
    [case(1)]
        LPUSE_INFO_1 UseInfo1;
    [case(2)]
        LPUSE_INFO_2 UseInfo2;
    [case(3)]
        LPUSE_INFO_3 UseInfo3;
    [default]
        ;
} USE_INFO, *PUSE_INFO, *LPUSE_INFO;

typedef struct _USE_INFO_0_CONTAINER {
    unsigned long EntriesRead;
    LPUSE_INFO_0 Buffer;
} USE_INFO_0_CONTAINER, *PUSE_INFO_0_CONTAINER, *LPUSE_INFO_0_CONTAINER;

typedef struct _USE_INFO_1_CONTAINER {
    unsigned long EntriesRead;
    LPUSE_INFO_1 Buffer;
} USE_INFO_1_CONTAINER, *PUSE_INFO_1_CONTAINER, *LPUSE_INFO_1_CONTAINER;

typedef struct _USE_INFO_2_CONTAINER {
    unsigned long EntriesRead;
    LPUSE_INFO_2 Buffer;
} USE_INFO_2_CONTAINER, *PUSE_INFO_2_CONTAINER, *LPUSE_INFO_2_CONTAINER;

typedef struct _USE_ENUM_STRUCT {
    DWORD   Level;
    [switch_is(Level)] union _USE_ENUM_UNION {
        [case(0)]
            LPUSE_INFO_0_CONTAINER Level0;
        [case(1)]
            LPUSE_INFO_1_CONTAINER Level1;
        [case(2)]
            LPUSE_INFO_2_CONTAINER Level2;
        [default]
            ;
    } UseInfo;
} USE_ENUM_STRUCT, *PUSE_ENUM_STRUCT, *LPUSE_ENUM_STRUCT;

unsigned long
NetrWkstaGetInfo (
    [in,string,unique] WKSSVC_IDENTIFY_HANDLE ServerName,
    [in] unsigned long Level,
    [out, switch_is(Level)] LPWKSTA_INFO WkstaInfo
);

unsigned long
NetrWkstaSetInfo (
    [in,string,unique] WKSSVC_IDENTIFY_HANDLE ServerName,
```

```
    [in] unsigned long  Level,
    [in, switch_is(Level)] LPWKSTA_INFO   WkstaInfo,
    [in,out,unique] unsigned long* ErrorParameter
);

typedef struct _WKSTA_USER_INFO_0_CONTAINER {
    unsigned long   EntriesRead;
    [size_is(EntriesRead)] LPWKSTA_USER_INFO_0 Buffer;
} WKSTA_USER_INFO_0_CONTAINER, *PWKSTA_USER_INFO_0_CONTAINER,
  *LPWKSTA_USER_INFO_0_CONTAINER;

typedef struct _WKSTA_USER_INFO_1_CONTAINER {
    unsigned long   EntriesRead;
    [size_is(EntriesRead)] LPWKSTA_USER_INFO_1 Buffer;
} WKSTA_USER_INFO_1_CONTAINER, *PWKSTA_USER_INFO_1_CONTAINER,
  *LPWKSTA_USER_INFO_1_CONTAINER;


typedef struct _WKSTA_USER_ENUM_STRUCT {
    unsigned long   Level;
    [switch_is(Level)] union _WKSTA_USER_ENUM_UNION {
        [case(0)]
            LPWKSTA_USER_INFO_0_CONTAINER Level0;
        [case(1)]
            LPWKSTA_USER_INFO_1_CONTAINER Level1;
        [default]
            ;
    } WkstaUserInfo;
} WKSTA_USER_ENUM_STRUCT,
  *PWKSTA_USER_ENUM_STRUCT,
  *LPWKSTA_USER_ENUM_STRUCT;

unsigned long
NetrWkstaUserEnum (
    [in,string,unique] WKSSVC_IDENTIFY_HANDLE ServerName,
    [in,out] LPWKSTA_USER_ENUM_STRUCT UserInfo,
    [in] unsigned long  PreferredMaximumLength,
    [out] unsigned long* TotalEntries,
    [in,out,unique] unsigned long* ResumeHandle
);

void Opnum3NotUsedOnWire(void);

void Opnum4NotUsedOnWire(void);

typedef struct _WKSTA_TRANSPORT_INFO_0_CONTAINER {
    unsigned long   EntriesRead;
    [size_is(EntriesRead)] LPWKSTA_TRANSPORT_INFO_0 Buffer;
} WKSTA_TRANSPORT_INFO_0_CONTAINER,
  *PWKSTA_TRANSPORT_INFO_0_CONTAINER,
  *LPWKSTA_TRANSPORT_INFO_0_CONTAINER;

typedef struct _WKSTA_TRANSPORT_ENUM_STRUCT {
    unsigned long   Level;
    [switch_is(Level)] union _WKSTA_TRANSPORT_ENUM_UNION {
        [case(0)]
            LPWKSTA_TRANSPORT_INFO_0_CONTAINER Level0;
        [default]
            ;
    } WkstaTransportInfo;
} WKSTA_TRANSPORT_ENUM_STRUCT, *PWKSTA_TRANSPORT_ENUM_STRUCT,
  *LPWKSTA_TRANSPORT_ENUM_STRUCT;

unsigned long
NetrWkstaTransportEnum (
    [in,string,unique] WKSSVC_IDENTIFY_HANDLE ServerName,
    [in,out] LPWKSTA_TRANSPORT_ENUM_STRUCT TransportInfo,
    [in] unsigned long  PreferredMaximumLength,
    [out] unsigned long* TotalEntries,
    [in,out,unique] unsigned long* ResumeHandle
```

```
);

unsigned long
NetrWkstaTransportAdd (
    [in,string,unique] WKSSVC_IDENTIFY_HANDLE ServerName,
    [in] unsigned long Level,
    [in] LPWKSTA_TRANSPORT_INFO_0 TransportInfo,
    [in,out,unique] unsigned long* ErrorParameter
);

unsigned long
NetrWkstaTransportDel (
    [in,string,unique] WKSSVC_IDENTIFY_HANDLE ServerName,
    [in,string,unique] wchar_t* TransportName,
    [in] unsigned long ForceLevel
);

unsigned long
NetrUseAdd (
    [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
    [in] unsigned long Level,
    [in, switch_is(Level)] LPUSE_INFO InfoStruct,
    [in, out, unique] unsigned long* ErrorParameter
);



unsigned long NetrUseGetInfo(
  [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
  [in, string] wchar_t* UseName,
  [in] unsigned long Level,
  [out, switch_is(Level)] LPUSE_INFO InfoStruct
);



unsigned long
NetrUseDel (
    [in, string, unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
    [in, string] wchar_t* UseName,
    [in] unsigned long ForceLevel
);

unsigned long
NetrUseEnum (
    [in, string, unique] WKSSVC_IDENTIFY_HANDLE ServerName,
    [in, out] LPUSE_ENUM_STRUCT InfoStruct,
    [in] unsigned long PreferredMaximumLength,
    [out] unsigned long* TotalEntries,
    [in, out, unique] unsigned long* ResumeHandle
);


void Opnum12NotUsedOnWire(void);

unsigned long
NetrWorkstationStatisticsGet(
    [in,string,unique] WKSSVC_IDENTIFY_HANDLE ServerName,
    [in,string,unique] wchar_t* ServiceName,
    [in] unsigned long Level,
    [in] unsigned long Options,
    [out] LPSTAT_WORKSTATION_0* Buffer
);

void Opnum14NotUsedOnWire(void);

void Opnum15NotUsedOnWire(void);

void Opnum16NotUsedOnWire(void);
```

```
        void Opnum17NotUsedOnWire(void);

        void Opnum18NotUsedOnWire(void);

        void Opnum19NotUsedOnWire(void);

        unsigned long
        NetrGetJoinInformation(
            [in,string,unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
            [in,out,string] wchar_t**    NameBuffer,
            [out] PNETSETUP_JOIN_STATUS BufferType
        );

        void Opnum21NotUsedOnWire(void);

    #define JOIN_OBFUSCATOR_LENGTH 8
    #define JOIN_MAX_PASSWORD_LENGTH 256

        typedef struct _JOINPR_USER_PASSWORD {
            unsigned char Obfuscator[JOIN_OBFUSCATOR_LENGTH];
            wchar_t Buffer[JOIN_MAX_PASSWORD_LENGTH];
            unsigned long Length;
        } JOINPR_USER_PASSWORD,
         *PJOINPR_USER_PASSWORD;

        typedef struct _JOINPR_ENCRYPTED_USER_PASSWORD {
            unsigned char Buffer[JOIN_OBFUSCATOR_LENGTH +
                (JOIN_MAX_PASSWORD_LENGTH * sizeof(wchar_t)) +
                sizeof(unsigned long)];
        } JOINPR_ENCRYPTED_USER_PASSWORD,
         *PJOINPR_ENCRYPTED_USER_PASSWORD;

        typedef struct _UNICODE_STRING {
            unsigned short Length;
            unsigned short MaximumLength;
            [size_is(MaximumLength / 2), length_is((Length) / 2)]
                unsigned short* Buffer;
        } UNICODE_STRING,
         *PUNICODE_STRING;

        unsigned long
        NetrJoinDomain2(
            [in] handle_t RpcBindingHandle,
            [in,string,unique] wchar_t* ServerName,
            [in,string] wchar_t* DomainNameParam,
            [in,string,unique] wchar_t* MachineAccountOU,
            [in,string,unique] wchar_t* AccountName,
            [in,unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
            [in] unsigned long Options
        );

        unsigned long
        NetrUnjoinDomain2(
            [in] handle_t RpcBindingHandle,
            [in,string,unique] wchar_t* ServerName,
            [in,string,unique] wchar_t* AccountName,
            [in,unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
            [in] unsigned long Options
        );

        unsigned long
        NetrRenameMachineInDomain2(
            [in] handle_t RpcBindingHandle,
            [in,string,unique] wchar_t* ServerName,
            [in,string,unique] wchar_t* MachineName,
            [in,string,unique] wchar_t* AccountName,
            [in,unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
            [in] unsigned long Options
        );
```

```
    unsigned long
    NetrValidateName2(
        [in] handle_t RpcBindingHandle,
        [in,string,unique] wchar_t* ServerName,
        [in,string] wchar_t* NameToValidate,
        [in,string,unique] wchar_t* AccountName,
        [in,unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
        [in] NETSETUP_NAME_TYPE NameType
    );

    unsigned long
    NetrGetJoinableOUs2(
        [in] handle_t  RpcBindingHandle,
        [in,string,unique] wchar_t*  ServerName,
        [in,string] wchar_t*  DomainNameParam,
        [in,string,unique] wchar_t*  AccountName,
        [in,unique] PJOINPR_ENCRYPTED_USER_PASSWORD Password,
        [in,out] unsigned long*  OUCount,
        [out,string,size_is(,*OUCount)] wchar_t*** OUs
    );

    unsigned long
    NetrAddAlternateComputerName(
        [in] handle_t  RpcBindingHandle,
        [in,string,unique] wchar_t*  ServerName,
        [in,string,unique] wchar_t*  AlternateName,
        [in,string,unique] wchar_t*  DomainAccount,
        [in,unique] PJOINPR_ENCRYPTED_USER_PASSWORD EncryptedPassword,
        [in] unsigned long  Reserved
    );

    unsigned long
    NetrRemoveAlternateComputerName(
        [in] handle_t  RpcBindingHandle,
        [in,string,unique] wchar_t*  ServerName,
        [in,string,unique] wchar_t*  AlternateName,
        [in,string,unique] wchar_t*  DomainAccount,
        [in,unique] PJOINPR_ENCRYPTED_USER_PASSWORD EncryptedPassword,
        [in] unsigned long  Reserved
    );

    unsigned long
    NetrSetPrimaryComputerName(
        [in] handle_t  RpcBindingHandle,
        [in,string,unique] wchar_t*  ServerName,
        [in,string,unique] wchar_t*  PrimaryName,
        [in,string,unique] wchar_t*  DomainAccount,
        [in,unique] PJOINPR_ENCRYPTED_USER_PASSWORD  EncryptedPassword,
        [in] unsigned long  Reserved
    );

    typedef struct _NET_COMPUTER_NAME_ARRAY {
        unsigned long EntryCount;
        [size_is(EntryCount)] PUNICODE_STRING ComputerNames;
    } NET_COMPUTER_NAME_ARRAY, *PNET_COMPUTER_NAME_ARRAY;

    unsigned long
    NetrEnumerateComputerNames(
        [in,string,unique] WKSSVC_IMPERSONATE_HANDLE ServerName,
        [in] NET_COMPUTER_NAME_TYPE  NameType,
        [in] unsigned long  Reserved,
        [out] PNET_COMPUTER_NAME_ARRAY  *ComputerNames
    );
}
```

7 (Updated Section) Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

- Windows NT operating system
- Windows 2000 operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Server 2003 R2 operating system
- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system
- Windows 10 operating system
- Windows Server 2016 operating system
- Windows Server operating system
- Windows Server 2019 operating system

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

<1> Section 1.4: Windows implementations use the **DsrGetDcNameEx2** method of the local [MS-NRPC] server to locate a domain controller when communication to a domain controller is specified.

<2> Section 1.4: Windows implementations use LDAP as a client when queries or modifications to directory objects are specified.

<3> Section 1.8: Windows only uses the values in [MS-ERREF].

<4> Section 2.1: Windows implementations use the identity of the caller to perform method-specific access checks.

<5> Section 2.2.5.8: Server implementations on Windows represent the name of the device in the form "\Device\<device_name>", where <device_name> is a local device in the NT namespace. Examples:

- \Device\NetbiosSmb
- \Device\NetBT_Tcpip_{E30E2FF4-044F-403B-9906-8E58FDFAD018}

Note that the names are driver-specific.

<6> Section 2.2.5.8: Windows NT, Windows 2000, Windows Server 2003, and Windows Server 2003 R2 implementations set the *wkti0_transport_address* parameter for the NetBIOS over TCP/IP (NetBT) transport protocol to a string that represents the IEEE 802.1 Media Access Control (MAC) address [IEEE802.1X] of the transport protocol. The string is formatted as described in IEEE 802.1 but with separator characters removed. For example, the MAC address "00-0F-FE-51-DE-3B" results in the *wkti0_transport_address* string "000FFE51DE3B".

<7> Section 2.2.5.9: On Windows implementations, the account name that was used to authenticate the user on the computer is used as the user name.

<8> Section 2.2.5.11: On Windows implementations, the *RawReadsDenied* parameter contains an indeterminate value on send and is ignored on receipt, except on Windows NT, on which the value is undefined.

<9> Section 2.2.5.11: On Windows implementations, the *RawWritesDenied* parameter contains an indeterminate value on send and is ignored on receipt, except on Windows NT, on which the value is undefined.

<10> Section 3.2.1.1: The **NetSecurityDescriptor** security descriptor is not defined on the following versions of Windows: Windows NT, Windows 2000, Windows XP operating system Service Pack 1 (SP1), and Windows Server 2003 before the release of Windows Server 2003 operating system with Service Pack 1 (SP1).

<11> Section 3.2.1.2: Windows versions implement **alternate-computer-names** as defined in section 3.2.1.2, except on Windows NT, and Windows 2000, where the default state for this list is empty.

<12> Section 3.2.1.3: Windows versions retrieve **OtherDomainsInitialization**, as defined in section 3.2.1.3, from the registry and return an empty list. Windows 2000, Windows XP, Windows Server 2003, Windows Server 2003 R2, Windows Vista, and Windows Server 2008 servers retrieve **OtherDomainsInitialization** from local registry values.

<13> Section 3.2.1.6: Domains on Windows NT do not have DNS names, so **DomainName.FQDN** is not established in that case.

<14> Section 3.2.3: Server implementations on Windows initialize this value to 1023. Windows 2000, Windows XP, Windows Server 2003, Windows Server 2003 R2, Windows Vista, and Windows Server 2008 servers initialize this value to 45.

<15> Section 3.2.3: Server implementations on Windows initialize this value to 600 seconds.

<16> Section 3.2.3: Server implementations on Windows initialize this value to 50.

<17> Section 3.2.3: Server implementations on Windows initialize this value to 0x000001F4, which is the value for Windows, as specified in section 2.2.5.1.

<18> Section 3.2.3: Server implementations on Windows initialize this value to 60 seconds.

<19> Section 3.2.3: Server implementations on Windows initialize this value to the major version number of the server operating system.

<20> Section 3.2.3: Server implementations on Windows initialize this value to the minor version number of the server operating system.

<21> Section 3.2.3: Server implementations on Windows set this to hexadecimal string representation of the 6-byte physical address of the interface.

<22> Section 3.2.3: Server implementations on Windows use the name of the transport device name associated with the transport, as specified in the **WKSTA_TRANSPORT_INFO_0** (section 2.2.5.8) structure.

<23> Section 3.2.3: Server implementations on Windows set this to any value.

<24> Section 3.2.3: Server implementations on Windows set this to TRUE for NetBIOS transports.

<25> Section 3.2.4: Windows: The underlying security subsystem is used to determine the permissions for the caller.

<26> Section 3.2.4: Gaps in the opnum numbering sequence apply to Windows as follows:

Opnum	Description
3	Only used locally by Windows, never remotely.
4	Only used locally by Windows, never remotely.
8	Only used locally by Windows, never remotely.
9	Only used locally by Windows, never remotely.
10	Only used locally by Windows, never remotely.
11	Only used locally by Windows, never remotely.
12	Only used locally by Windows, never remotely.
14	Only used locally by Windows, never remotely.
15	Only used locally by Windows, never remotely.
16	Just returns ERROR_NOT_SUPPORTED. It is never used.
17	Just returns ERROR_NOT_SUPPORTED. It is never used.
18	Just returns ERROR_NOT_SUPPORTED. It is never used.
19	Just returns ERROR_NOT_SUPPORTED. It is never used.
21	Just returns ERROR_NOT_SUPPORTED. It is never used.

Windows error codes are specified in [MS-ERREF] section 2.2.

<27> Section 3.2.4: Windows implementations do not establish SMB sessions or SMB share connections during message processing, except on Windows NT and Windows 2000.

<28> Section 3.2.4.1: Server implementations on Windows require that the caller be a member of the Administrators group if the *Level* parameter is equal to 0x00000066 or 0x000001F6. If the caller is not a member of the Administrators group, the server fails the method with ERROR_ACCESS_DENIED.

<29> Section 3.2.4.1: Server implementations on Windows set **wki502_dormant_file_limit** to zero, except on Windows NT.

<30> Section 3.2.4.2: If the *Level* is invalid, server implementations on Windows fail the call with ERROR_INVALID_PARAMETER.

<31> Section 3.2.4.2: Windows implementations use the values of these members to configure the SMB redirector, except where noted. The server stores the value and returns it when the client requests it.

<32> Section 3.2.4.2: Windows requires that the caller is a member of the Administrators group; otherwise, the server fails the method with ERROR_ACCESS_DENIED.

<33> Section 3.2.4.3: Windows implementations return the zero-based index of the user to be enumerated from the list of currently logged-on users.

<34> Section 3.2.4.3: Windows requires that the caller is a member of the Administrators group.

<35> Section 3.2.4.3: Server implementations on Windows identify every active user by a logon number. Logon numbers are monotonically increasing in order. Active users are enumerated in increasing order of logon number. *ResumeHandle* stores the logon number of the last user returned to the client. If **NetrWkstaUserEnum** (section 3.2.4.3) is called with a nonzero *ResumeHandle*, then the server only enumerates those active users who have a logon number greater than the *ResumeHandle*.

<36> Section 3.2.4.4: Windows implementations return the zero-based index of the transport protocol to be enumerated from the list of currently enabled transport protocols.

<37> Section 3.2.4.4: Server implementations on Windows do not enforce this security measure.

<38> Section 3.2.4.4: Windows implementations maintain an array of transport protocols currently enabled for use by the SMB network redirector. Currently enabled transport protocols are enumerated starting at the beginning of this array. *ResumeHandle* stores the position in this array of the last transport protocol returned to the client. If **NetrWkstaTransportEnum** (section 3.2.4.4) is called with a non-zero *ResumeHandle*, then the server begins enumerating the array from one position ahead of the *ResumeHandle*. If transport protocols are added or deleted from this array in-between calls to **NetrWkstaTransportEnum**, then some transports might not be enumerated at all, or some transports might be enumerated multiple times.

<39> Section 3.2.4.4: Windows implementations set the *TotalEntries* value to zero.

<40> Section 3.2.4.5: This method is deprecated on Windows releases. If the *Level* parameter is set to zero and the caller belongs to the Administrators group, server implementations on Windows return ERROR_INVALID_FUNCTION without any further processing.

This method is not deprecated on Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<41> Section 3.2.4.5: Windows NT requires that the caller be a member of the Administrators group.

<42> Section 3.2.4.6: This method is deprecated on Windows releases. If the *ForceLevel* parameter is set to 0x00000000, 0x00000001, or 0x00000002, and the caller belongs to the Administrators group, Windows implementations return ERROR_INVALID_FUNCTION.

This method is not deprecated on Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<43> Section 3.2.4.6: Windows NT requires that the caller be a member of the Administrators group.

<44> Section 3.2.4.7: Although Windows implementations, expose this RPC call to remote callers, it is called only by processes on the local machine. Windows clients do not issue this RPC call to a remote machine.

<45> Section 3.2.4.7: Windows requires the caller to be a member of either the Administrators or standard user group.

<46> Section 3.2.4.8: Although server implementations on Windows expose this RPC call to remote callers, it is intended to be called only by processes on the local machine. Windows clients do not issue this RPC call to a remote machine.

<47> Section 3.2.4.8: Windows requires the caller to be a member of the Administrators or standard user group.

<48> Section 3.2.4.9: Server implementations on Windows expose this RPC call to remote callers, but it is intended to be called only by processes on the local machine. Windows clients do not issue this RPC call to a remote machine.

<49> Section 3.2.4.9: Windows callers are members of either the Administrators or standard user group.

<50> Section 3.2.4.10: Server implementations on Windows expose this RPC call to remote callers, but it is intended to be called only by processes on the local machine. Client implementations on Windows do not issue this RPC call to a remote machine.

<51> Section 3.2.4.10: Windows requires the caller to be a member of either the Administrators or standard user group.

<52> Section 3.2.4.10: Windows implementations return the zero-based index of the user to be enumerated from the list of currently logged-on users.

<53> Section 3.2.4.11: Windows requires that the caller is a member of the Administrators or standard user group.

<54> Section 3.2.4.11: Windows implementations use these implementation-specific values as counters for the number of I/Os performed for each type. For information on paging, and the I/O system for background on these values, see [WININTERNALS] chapters 7 and 9.

<55> Section 3.2.4.12: Windows implementations enforce the verification of the proper RPC protocol sequence, except in the following versions: Windows NT, Windows 2000, and Windows XP without service packs.

<56> Section 3.2.4.13: This method is not available on Windows NT.

<57> Section 3.2.4.13: Apart from the values mentioned in the table in section 3.2.4.13, the following option is applicable only to server implementations on Windows.

Value/code	Meaning
NETSETUP_WIN9X_UPGRADE 0x00000010	The join operation occurs as part of an upgrade from Windows 95 operating system, Windows 98 operating system, or Windows Millennium Edition operating system to Windows NT, Windows 2000, or Windows XP.

<58> Section 3.2.4.13: Windows NT and Windows 2000 do not implement this behavior.

<59> Section 3.2.4.13: Windows implementations do not update the **DnsHostName** and service principal name (SPN) properties on the computer during message processing when **NETSETUP_DEFER_SPN_SET** is specified. The values are updated in a subsequent call to **NetrRenameMachineInDomain2** (section 3.2.4.15).

Windows NT and Windows 2000 do not implement this behavior.

<60> Section 3.2.4.13: Windows implementations do not update the DnsHostName and SPN properties on the computer during message processing when **NETSETUP_DEFER_SPN_SET** is

specified. The values are updated in a subsequent call to **NetrRenameMachineInDomain2** (section 3.2.4.15).

Windows NT and Windows 2000 do not implement this behavior.

<61> Section 3.2.4.13: Windows implementations continue message processing of a domain join when the domain-object already exists, and that object is a domain controller account and **NETSETUP_JOIN_DC_ACCOUNT** is specified.

Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2 do not implement this behavior.

<62> Section 3.2.4.13: Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2 implementations do not support **NETSETUP_JOIN_DC_ACCOUNT**.

Client implementations on Windows releases pass **NETSETUP_JOIN_DC_ACCOUNT**, which servers on Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2 ignore. Clients on Windows NT do not support this behavior.

When setting this flag, client implementations locate a domain controller on a server. The location mechanism is specified in the **DsrGetDcNameEx2** method ([MS-NRPC] section 3.5.4.3.1); the flag description for **DS_FULL_SECRET_DOMAIN_6_FLAG** is specified in [MS-ADTS] section 6.3.3.2.

The use of **DS_FULL_SECRET_DOMAIN_6_FLAG** ensures the location of a domain controller on server implementations on applicable Windows Server releases. Server implementations on Windows 2000 Server operating system, Windows Server 2003, and Windows Server 2003 R2 do not support this behavior.

<63> Section 3.2.4.13: Windows implementations use the most recently set computer name during a domain join when **NETSETUP_JOIN_WITH_NEW_NAME** is specified. Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2 do not implement this behavior.

In the Windows implementation of a computer rename, a computer is restarted before a new name can be used. This flag allows the new name to be used for a join before a restart. For example, processing a **NetrRenameMachineInDomain2** (section 3.2.4.15) message would change the persisted abstract state **ComputerName** (section 3.2.1.2), but the change would not be effective until after a machine restart. Specifying this flag would cause the join operation to use the new **ComputerName** when joining the domain.

<64> Section 3.2.4.13: Windows implementations indicate that concurrent calls to this method are not supported. Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2 do not implement this behavior.

<65> Section 3.2.4.13.1: Windows implementations pass **NETSETUP_MACHINE_PWD_PASSED** or **NETSETUP_DEFER_SPN**. This flag is ignored by Windows NT and Windows 2000 implementations.

<66> Section 3.2.4.13.1: Windows implementations define **NETSETUP_IGNORE_UNSUPPORTED_FLAGS** but do not set the flag. This flag is ignored by Windows NT, Windows 2000, and Windows XP server implementations.

<67> Section 3.2.4.13.1: Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2 do not support **NETSETUP_JOIN_DC_ACCOUNT**.

When setting the **NETSETUP_JOIN_DC_ACCOUNT** flag, clients on server implementations on Windows locate a domain controller on other server implementations on Windows. Windows NT Server operating system, Windows 2000 Server, Windows Server 2003, and Windows Server 2003 R2 do not support this behavior.

Using the DS_FULL_SECRET_DOMAIN_6_FLAG flag ([MS-ADTS] section 6.3.3.2) ensures locating a domain controller with that version.

<68> Section 3.2.4.13.1: Windows implementations enforce the verification of the proper RPC protocol sequence, except on Windows NT, Windows 2000, Windows XP, Windows XP SP1, and Windows Server 2003 before Windows Server 2003 with SP1. If the server identifies a previous RPC call that is modifying the identity of the machine, the server returns **RPC_S_CALL_IN_PROGRESS**.

<69> Section 3.2.4.13.1: Windows Vista Home Basic and Windows Vista Home Premium implementations return ERROR_NOT_SUPPORTED if this method is invoked.

<70> Section 3.2.4.13.2: Windows implementations find and set the described state through the following sequence:

1. The computer account object is created at a writable domain controller (writable DC) using the **SamrCreateUser2InDomain** method ([MS-SAMR] section 3.1.5.4.4).

2. The LDAP attributes **userAccountControl** and **unicodePwd** ([MS-ADA3] section 2.342 and [MS-ADA3] section 2.332, respectively) are set using the **SamrSetInformationUser2** method of [MS-SAMR] section 3.1.5.6.4.

3. The LDAP attributes **dNSHostName** and **servicePrincipalName** ([MS-ADTS] section 3.1.1.3.2.4 or [MS-ADA1] section 2.185, and [MS-ADA3] section 2.253, respectively) are set using the LDAP protocol ([RFC2252] and [RFC2253]).

<71> Section 3.2.4.13.3: Windows implementations send an **LsarOpenPolicy2** request to a domain controller and, using the returned policy handle [MS-LSAD], query for the domain name and SID by sending an **LsarQueryInformationPolicy2** request for InformationClass **PolicyDnsDomainInformation**, followed by sending an **LsarQueryInformationPolicy** request for InformationClass **PolicyPrimaryDomainInformation**.

<72> Section 3.2.4.13.3: Windows implementations use the algorithm specified in [FIPS186-2] for generating each byte of the machine password. [FIPS186-2] Appendix 3.1 describes a pseudo-random number generator (PRNG) that can use either DES or SHA-1. Windows uses a SHA-1–based PRNG to satisfy FIPS 140-2 level 2 cryptographic module certification requirements [FIPS140].

In the PRNG description of [FIPS186-2] Appendix 3.1, G is constructed from SHA-1 with the first parameter as the initial value for the SHA-1 registers, whereas the second parameter is the data input to be hashed.

Integer *b* is replaced with 160.

XKEY is determined by a call to an RC4-based PRNG.

The variable *q* is not used in the general-purpose version of [FIPS186-2] (see Appendix 6.9 General Purpose Random Number Generation).

$XSEED_j$ is also determined by a call to an RC4-based PRNG for every block output by the [FIPS186-2] PRNG.

The variable *m* is the number of blocks that can be output by the [FIPS186-2] PRNG before a non-NULL value is passed to $XSEED_j$. The Windows implementation sets it to the shortest possible value, which is 1.

<73> Section 3.2.4.13.3: Windows implementations store the Internet host name of the computer locally based on a local, configurable setting, which, by default, is set to store the name.

<74> Section 3.2.4.14: This method is not available on Windows NT.

<75> Section 3.2.4.14: Windows NT, Windows 2000, and Windows XP do not support this flag.

<76> Section 3.2.4.14: Windows implementations save the original state in memory for the duration of message processing before making any changes, and when message processing encounters an

error, the original state is restored before returning to the caller. This state is not persisted or retained beyond the processing duration of a call.

Persisted state manipulations are performed using local services or other network protocols as referenced in the message processing section. This is done on a best-effort basis; if an error is encountered during the restoration process, the computer is left in a different state than it was immediately before the call was processed.

<77> Section 3.2.4.14: Windows implementations enforce the verification of the proper RPC protocol sequence, except on Windows NT, Windows 2000, Windows XP, Windows XP SP1, and Windows Server 2003 before Windows Server 2003 with SP1.

<78> Section 3.2.4.14: Windows implementations update the Internet host name of the computer locally based on a local configurable setting, which, by default, is set to store the name.

<79> Section 3.2.4.15: This method is not available on Windows NT.

<80> Section 3.2.4.15: Only Windows 2000 implementations return this error.

<81> Section 3.2.4.15: Windows implementations save the original state in memory for the duration of message processing prior to making any changes, and when message processing encounters an error the original state is restored prior to returning to the caller. This state is not persisted or retained beyond the processing duration of a call.

Persisted state manipulations are performed using local services or other network protocols as referenced in the message processing section. This is done on a best-effort basis: if an error is encountered during the restoration process, the server is left in a state different than it was in immediately prior to the call being processed.

<82> Section 3.2.4.15: Windows implementations enforce the verification of the proper RPC protocol sequence, except on Windows NT, Windows 2000, Windows XP, Windows XP SP1, and Windows Server 2003 before Windows Server 2003 with SP1.

<83> Section 3.2.4.15: Windows uses a syntactic/textual conversion. This conversion limits computer names to be the common subset of the names. Specifically, the name's leftmost label is truncated to 15-bytes of OEM characters in uppercase.

<84> Section 3.2.4.15: Windows populates the DNS suffix using Group Policy from the domain. Group Policy updates the following registry key:

- HKLM\Software\Policies\Microsoft\System\DNSclientNV PrimaryDnsSuffix

If this setting is not available, the TCP/IP setting for the domain is queried and is used as the DNS suffix.

If that is not available either, the fully qualified domain name (FQDN) of the domain is used as the DNS suffix.

<85> Section 3.2.4.16: This method is not available on Windows NT.

<86> Section 3.2.4.16: Windows implementations enforce the verification of the proper RPC protocol sequence, except on Windows NT, Windows 2000, Windows XP, Windows XP SP1, and Windows Server 2003 before Windows Server 2003 with SP1.

<87> Section 3.2.4.16: Windows implementations verify that the caller is local and return **RPC_E_REMOTE_DISABLED** if not, except on Windows NT, Windows 2000, Windows XP, and Windows XP SP1.

<88> Section 3.2.4.16: Windows implementation: The name is added as a NetBIOS group name. If the operation succeeds, the name is verified as valid; the name [RFC1001] is deleted to undo the addition of the name. Otherwise, the name is not valid; ERROR_INVALID_PARAMETER is returned.

<89> Section 3.2.4.16: Windows implementation: The name is added as a NetBIOS unique name. If the operation succeeds, the name is verified as valid and not in use; the name [RFC1001] is deleted to undo the addition of the name. Otherwise, the name is not valid; ERROR_DUP_NAME is returned if the name [RFC1001] is already in use.

<90> Section 3.2.4.16: Windows implementation: The **DsrGetDcNameEx2** method of the local [MS-NRPC] server is used to verify that a domain controller can be found for the domain. If the locator succeeds in finding any domain controller in the domain, this condition is verified. Otherwise, ERROR_NO_SUCH_DOMAIN is returned.

<91> Section 3.2.4.16: Windows implementation: The **DsrGetDcNameEx2** method of the local [MS-NRPC] server is used to determine if a domain controller can be found for the domain. If the locator finds a domain controller in the domain, ERROR_DUP_NAME is returned. Otherwise, this condition is verified.

<92> Section 3.2.4.17: This method is not available on Windows NT.

<93> Section 3.2.4.17: Windows implementations return this error to indicate that the password is incorrect.

<94> Section 3.2.4.17: Windows implementations enforce the verification of the proper RPC protocol sequence, except on Windows NT, Windows 2000, Windows XP, Windows XP SP1, and Windows Server 2003 before Windows Server 2003 with SP1.

<95> Section 3.2.4.17: Windows implementations verify that the caller is local, except on Windows NT, Windows 2000, Windows XP, and Windows XP SP1.

<96> Section 3.2.4.17: Windows implementations fail this call on a domain controller, and NERR_InvalidAPI is returned.

<97> Section 3.2.4.18: This method is not available on Windows NT and Windows 2000.

<98> Section 3.2.4.18: Windows NT, Windows 2000, and Windows XP implementations do not check the NET_IGNORE_UNSUPPORTED_FLAGS bit.

<99> Section 3.2.4.18: Windows NT, Windows 2000, Windows XP, and Windows Server 2003 do not indicate that concurrent calls to this method are not supported.

<100> Section 3.2.4.18: Windows implementations save the original state in memory for the duration of message processing prior to making any changes, and when message processing encounters an error the original state is restored prior to returning to the caller. This state is not persisted or retained beyond the processing duration of a call. Persisted state manipulations are performed using local services or other network protocols as referenced in the message processing section. This is done on a best-effort basis: if an error is encountered during the restoration process, the computer is left in a different state than it was before the call was processed.

<101> Section 3.2.4.18: Windows implementations enforce the verification of the proper RPC protocol sequence. If the server identifies a previous RPC call that is modifying the identity of the machine, the server returns **RPC_S_CALL_IN_PROGRESS**.

<102> Section 3.2.4.18: Windows clients return ERROR_NOT_SUPPORTED if this method is invoked.

<103> Section 3.2.4.18: Windows uses a syntactic/textual conversion. This conversion limits the names of computers to be the common subset of the names. Specifically, the leftmost label of the name is truncated to 15 bytes of OEM characters in uppercase.

<104> Section 3.2.4.19: This method is not available on Windows NT and Windows 2000.

<105> Section 3.2.4.19: Windows NT, Windows 2000, and Windows XP implementations do not check the NET_IGNORE_UNSUPPORTED_FLAGS bit.

<106> Section 3.2.4.19: Windows implementations indicate that concurrent calls to this method are not supported:

<107> Section 3.2.4.19: Windows implementations save the original state in memory for the duration of message processing prior to making any changes, and when message processing encounters an error, the original state is restored prior to returning to the caller. This state is not persisted or retained beyond the processing duration of a call.

Persisted state manipulations are performed by using local services or other network protocols as referenced in the message processing section. This is done on a best-effort basis: If an error is encountered during the restoration process, the computer is left in a different state than it was before the call was processed.

<108> Section 3.2.4.19: Windows implementations enforce the verification of the proper RPC protocol sequence. If the server identifies a previous RPC call that is modifying the identity of the machine, the server returns **RPC_S_CALL_IN_PROGRESS**.

<109> Section 3.2.4.19: Windows client return ERROR_NOT_SUPPORTED if this method is invoked.

<110> Section 3.2.4.19: Windows uses a syntactic/textual conversion. This conversion limits the names of computers to the common subset of the names. Specifically, the leftmost label of the name is truncated to 15-bytes of OEM characters in uppercase.

<111> Section 3.2.4.20: This method is not available on Windows NT and Windows 2000.

<112> Section 3.2.4.20: Windows NT, Windows 2000, and Windows XP implementations do not check the NET_IGNORE_UNSUPPORTED_FLAGS bit.

<113> Section 3.2.4.20: Windows implementations indicate that concurrent calls to this method are not supported.

<114> Section 3.2.4.20: Windows implementations save the original state in memory for the duration of message processing prior to making any changes, and when message processing encounters an error, the original state is restored prior to returning to the caller. This state is not persisted or retained beyond the processing duration of a call.

Persisted state manipulations are performed by using local services or other network protocols as referenced in the message processing section. This is done on a best-effort basis: If an error is encountered during the restoration process, the computer is left in a different state than it was before the call was processed.

<115> Section 3.2.4.20: Windows implementations enforce the verification of the proper RPC protocol sequence. If the server identifies a previous RPC call that is modifying the identity of the machine, the server returns **RPC_S_CALL_IN_PROGRESS**.

<116> Section 3.2.4.20: Windows clients return **ERROR_NOT_SUPPORTED** if this method is invoked.

<117> Section 3.2.4.20: Windows uses a syntactic/textual conversion. This conversion limits the names of computers to the common subset of the names. Specifically, the leftmost label of the name is truncated to 15 bytes of OEM characters in uppercase.

<118> Section 3.2.4.21: This method is not available on Windows NT and Windows 2000.

<119> Section 3.2.4.21: Windows NT, Windows 2000, and Windows XP implementations do not check the NET_IGNORE_UNSUPPORTED_FLAGS bit.

<120> Section 3.2.4.21: Windows implementations enforce the verification of the proper RPC protocol sequence.

<121> Section 3.2.4.21: When this method is invoked, Windows implementations return **ERROR_NOT_SUPPORTED**.

8 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Description	Revision class
7 Appendix B: Product Behavior	Added Windows Server 2019 to the list of applicable products.	Major

9 Index

A

Abstract data model
  client 44
  server 45
Applicability 19

C

Capability negotiation 19
Change tracking 155
Client
  abstract data model 44
  initialization 44
  local events 44
  message processing 44
  sequencing rules 44
  timer events 44
  timers 44
Common Message Processing method 125

D

Data model - abstract
  client 44
  server 45
Data types 21
Decoding passwords 38
Decryption 37
Directory service schema elements 43
Domain join
  message processing 83

E

Elements - directory service schema 43
Encoding passwords 35
Encryption 37
Enumerations 21
Events
  local - client 44
  local - server 130
  timer - client 44
  timer - server 130
Examples
  netrjoindomain2 example 132
  netrwkstagetinfo example 131
  netrwkstauserenum example 131

F

Fields - vendor-extensible 19
Full IDL 136

G

Glossary 8

I

IDL 136

Implementer - security considerations 135
Implementer considerations - security 135
Index of security parameters 135
Informative references 15
Initialization
  client 44
  server 50
Introduction 8

J

JOIN_MAX_PASSWORD_LENGTH 20
JOIN_OBFUSCATOR_LENGTH 20
JOINPR_ENCRYPTED_USER_PASSWORD structure 35
JOINPR_USER_PASSWORD 37
JOINPR_USER_PASSWORD structure 34

L

Local events
  client 44
  server 130
LPSTAT_WORKSTATION_0 30
LPUSE_ENUM_STRUCT 42
LPUSE_INFO_0 39
LPUSE_INFO_0_CONTAINER 41
LPUSE_INFO_1 39
LPUSE_INFO_1_CONTAINER 42
LPUSE_INFO_2 41
LPUSE_INFO_2_CONTAINER 42
LPUSE_INFO_3 41
LPWKSTA_INFO_100 24
LPWKSTA_INFO_101 25
LPWKSTA_INFO_1013 28
LPWKSTA_INFO_1018 28
LPWKSTA_INFO_102 25
LPWKSTA_INFO_1046 28
LPWKSTA_INFO_502 26
LPWKSTA_TRANSPORT_ENUM_STRUCT 34
LPWKSTA_TRANSPORT_INFO_0 29
LPWKSTA_TRANSPORT_INFO_0_CONTAINER 33
LPWKSTA_USER_ENUM_STRUCT 33
LPWKSTA_USER_INFO_0 29
LPWKSTA_USER_INFO_0_CONTAINER 32
LPWKSTA_USER_INFO_1 29
LPWKSTA_USER_INFO_1_CONTAINER 33

M

MAX_PREFERRED_LENGTH 20
Message processing
  client 44
  server 51
Messages
  data types 21
  enumerations 21
  processing (section 3.2.4.13.1 81, section 3.2.4.13.3 83, section 3.2.4.13.4 87)
  structures 24
  syntax 20
  transport 20
  unions 23
Methods
  Common Message Processing 125
  NetrAddAlternateComputerName (Opnum 27) 101
  NetrEnumerateComputerNames (Opnum 30) 122

NetrGetJoinableOUs2 (Opnum 26) 98
NetrGetJoinInformation (Opnum 20) 76
NetrJoinDomain2 (Opnum 22) 78
NetrRemoveAlternateComputerName (Opnum 28) 108
NetrRenameMachineInDomain2 (Opnum 24) 90
NetrSetPrimaryComputerName (Opnum 29) 114
NetrUnjoinDomain2 (Opnum 23) 87
NetrUseAdd (Opnum 8) 66
NetrUseDel (Opnum 10) 71
NetrUseEnum (Opnum 11) 73
NetrUseGetInfo (Opnum 9) 68
NetrValidateName2 (Opnum 25) 95
NetrWkstaGetInfo (Opnum 0) 53
NetrWkstaSetInfo (Opnum 1) 55
NetrWkstaTransportAdd (Opnum 6) 63
NetrWkstaTransportDel (Opnum 7) 64
NetrWkstaTransportEnum (Opnum 5) 61
NetrWkstaUserEnum (Opnum 2) 60
NetrWorkstationStatisticsGet (Opnum 13) 75

N

NET_COMPUTER_NAME_ARRAY structure 39
NET_COMPUTER_NAME_TYPE enumeration 22
NetrAddAlternateComputerName (Opnum 27) method 101
NetrAddAlternateComputerName method 101
NetrEnumerateComputerNames (Opnum 30) method 122
NetrEnumerateComputerNames method 122
NetrGetJoinableOUs2 (Opnum 26) method 98
NetrGetJoinableOUs2 method 98
NetrGetJoinInformation (Opnum 20) method 76
NetrGetJoinInformation method 76
NetrJoinDomain2 (Opnum 22) method 78
Netrjoindomain2 example example 132
NetrJoinDomain2 method 78
NetrRemoveAlternateComputerName (Opnum 28) method 108
NetrRemoveAlternateComputerName method 108
NetrRenameMachineInDomain2 (Opnum 24) method 90
NetrRenameMachineInDomain2 method 90
NetrSetPrimaryComputerName (Opnum 29) method 114
NetrSetPrimaryComputerName method 114
NetrUnjoinDomain2 (Opnum 23) method 87
NetrUnjoinDomain2 method 87
NetrUseAdd (Opnum 8) method 66
NetrUseAdd method 66
NetrUseDel (Opnum 10) method 71
NetrUseDel method 71
NetrUseEnum (Opnum 11) method 73
NetrUseEnum method 73
NetrUseGetInfo (Opnum 9) method 68
NetrUseGetInfo method 68
NetrValidateName2 (Opnum 25) method 95
NetrValidateName2 method 95
NetrWkstaGetInfo (Opnum 0) method 53
NetrWkstaGetInfo example 131
Netrwkstagetinfo example example 131
NetrWkstaGetInfo method 53
NetrWkstaSetInfo (Opnum 1) method 55
NetrWkstaSetInfo method 55
NetrWkstaTransportAdd (Opnum 6) method 63
NetrWkstaTransportAdd method 63
NetrWkstaTransportDel (Opnum 7) method 64
NetrWkstaTransportDel method 64
NetrWkstaTransportEnum (Opnum 5) method 61
NetrWkstaTransportEnum method 61

NetrWkstaUserEnum (Opnum 2) method 60
NetrWkstaUserEnum example 131
Netrwkstauserenum example example 131
NetrWkstaUserEnum method 60
NetrWorkstationStatisticsGet (Opnum 13) method 75
NetrWorkstationStatisticsGet method 75
NETSETUP_JOIN_STATUS enumeration 21
NETSETUP_NAME_TYPE enumeration 22
Normative references 14

O

Overview (synopsis) 16

P

Parameters - security index 135
Password
  decoding 38
  encoding 35
PJOINPR_ENCRYPTED_USER_PASSWORD 35
PJOINPR_USER_PASSWORD 34
PNET_COMPUTER_NAME_ARRAY 39
Preconditions 19
Prerequisites 19
Product behavior 144
Protocol Details
  overview 44
PSTAT_WORKSTATION_0 30
PUNICODE_STRING 38
PUSE_ENUM_STRUCT 42
PUSE_INFO_0 39
PUSE_INFO_0_CONTAINER 41
PUSE_INFO_1 39
PUSE_INFO_1_CONTAINER 42
PUSE_INFO_2 41
PUSE_INFO_2_CONTAINER 42
PUSE_INFO_3 41
PWKSTA_INFO_100 24
PWKSTA_INFO_101 25
PWKSTA_INFO_1013 28
PWKSTA_INFO_1018 28
PWKSTA_INFO_102 25
PWKSTA_INFO_1046 28
PWKSTA_INFO_502 26
PWKSTA_TRANSPORT_ENUM_STRUCT 34
PWKSTA_TRANSPORT_INFO_0 29
PWKSTA_TRANSPORT_INFO_0_CONTAINER 33
PWKSTA_USER_ENUM_STRUCT 33
PWKSTA_USER_INFO_0 29
PWKSTA_USER_INFO_0_CONTAINER 32
PWKSTA_USER_INFO_1 29
PWKSTA_USER_INFO_1_CONTAINER 33

R

References 14
  informative 15
  normative 14
Relationship to other protocols 16

S

Schema elements - directory service 43
Security

implementer considerations 135
parameter index 135
Sequencing rules
client 44
server 51
Server
abstract data model 45
Common Message Processing method 125
initialization 50
local events 130
message processing 51
NetrAddAlternateComputerName (Opnum 27) method 101
NetrEnumerateComputerNames (Opnum 30) method 122
NetrGetJoinableOUs2 (Opnum 26) method 98
NetrGetJoinInformation (Opnum 20) method 76
NetrJoinDomain2 (Opnum 22) method 78
NetrRemoveAlternateComputerName (Opnum 28) method 108
NetrRenameMachineInDomain2 (Opnum 24) method 90
NetrSetPrimaryComputerName (Opnum 29) method 114
NetrUnjoinDomain2 (Opnum 23) method 87
NetrUseAdd (Opnum 8) method 66
NetrUseDel (Opnum 10) method 71
NetrUseEnum (Opnum 11) method 73
NetrUseGetInfo (Opnum 9) method 68
NetrValidateName2 (Opnum 25) method 95
NetrWkstaGetInfo (Opnum 0) method 53
NetrWkstaSetInfo (Opnum 1) method 55
NetrWkstaTransportAdd (Opnum 6) method 63
NetrWkstaTransportDel (Opnum 7) method 64
NetrWkstaTransportEnum (Opnum 5) method 61
NetrWkstaUserEnum (Opnum 2) method 60
NetrWorkstationStatisticsGet (Opnum 13) method 75
sequencing rules 51
timer events 130
timers 50
Standards assignments 19
STAT_WORKSTATION_0 structure 30
Structures 24
Syntax
data types 21
enumerations 21
overview 20
structures 24
unions 23

T

Timer events
client 44
server 130
Timers
client 44
server 50
Tracking changes 155
Transport 20

U

UNICODE_STRING structure 38
Unions 23
USE_ENUM_STRUCT structure 42
USE_INFO_0 structure 39
USE_INFO_0_CONTAINER structure 41
USE_INFO_1 structure 39
USE_INFO_1_CONTAINER structure 42

USE_INFO_2 structure 41
USE_INFO_2_CONTAINER structure 42
USE_INFO_3 structure 41

V

Vendor-extensible fields 19
Versioning 19

W

WKSTA_INFO_100 structure 24
WKSTA_INFO_101 structure 25
WKSTA_INFO_1013 structure 28
WKSTA_INFO_1018 structure 28
WKSTA_INFO_102 structure 25
WKSTA_INFO_1046 structure 28
WKSTA_INFO_502 structure 26
WKSTA_TRANSPORT_ENUM_STRUCT structure 34
WKSTA_TRANSPORT_INFO_0 structure 29
WKSTA_TRANSPORT_INFO_0_CONTAINER structure 33
WKSTA_USER_ENUM_STRUCT structure 33
WKSTA_USER_INFO_0 structure 29
WKSTA_USER_INFO_0_CONTAINER structure 32
WKSTA_USER_INFO_1 structure 29
WKSTA_USER_INFO_1_CONTAINER structure 33
Workgroup join - message processing 87